# Table of Contents

- **Introduction** 19

- **Chapter 1: What's New?** 21
  - New Features in Version 10.1.0 21
    - Configuring a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing 21
  - Changed Features in Version 10.1.0 21
    - TM1 Turbo Integrator Delivery Module 21
  - New Features in Version 8.4 22
    - DataStage Node 22
    - New Bulk Loader Delivery Modules 22
    - Multiple Data Sources in IBM Cognos Connection 22
  - Changed Features in Version 8.4 22
    - Credentials 22
    - Email Nodes 23
  - New Features in Version 8.3 23
    - Support for Windows Vista 23
    - Support for IBM Cognos Data Sources 23
    - Enhanced Unmatched Member Support 23
    - Enhanced Semi-Aggregate Handling 23
    - Load Control 23
    - Enhanced Error Reporting for Relational Delivery Exceptions 23
  - Changed Features in Version 8.3 24
    - Error Checking in SQLTXT Files 24
    - Aggregation Exception Handling 24

- **Chapter 2: IBM Cognos Data Manager** 25
  - Building the Data Foundation for the IBM Cognos Business Intelligence Solution 25
  - Business Dimensions 26
  - Conformed Data Warehouses 27
  - Scalable Dimension Architecture 28
  - IBM Cognos Data Manager Environments 28
  - IBM Cognos Data Manager Designer 28
  - The IBM Cognos Data Manager Engine 29
  - IBM Cognos Data Manager Language 29
  - Application Limits 30
  - Multi-platform 31

- **Chapter 3: Getting Started with IBM Cognos Data Manager Designer** 33
  - The Tree Pane 33
  - The Builds and JobStreams Folder 33
  - The Metadata Folder 34
  - The Library Folder 34
  - Set Up User-Defined Folders 34
  - Delete a User-Defined Folder 35
Change the Default Folder Settings 35
Select Multiple Components 36
The Visualization Pane 36
  Change the Default Settings in the Visualization Pane 37
  Change the Size of the Information in the Visualization Pane 37
  Reveal or Hide the Visualization Pane 37
  Copy the Visualization Pane to the Clipboard 38
Set Up Window and Dialog Box Preferences 38
Locate Components and Their Dependencies 38
  Locate a Component 38
  View and Locate the Dependencies of a Component 39
Create a Similar Component 40
Case Sensitivity 40
Cognos SQL 40
Change Fonts 40
Quoted Identifiers for Multibyte Character Sets 41

Chapter 4: IBM Cognos Data Manager Catalogs 43
  Create an IBM Cognos Data Manager Catalog 44
  Open an IBM Cognos Data Manager Catalog 44
  Change the Properties of a Catalog 45
  Document an IBM Cognos Data Manager Catalog 45
  Create an IBM Cognos Data Manager Catalog Schema 46
  Close a Catalog 48

Chapter 5: Upgrading a Catalog 49
  Perform an Upgrade 49
  Upgrade Failure 51
  Restore From an IBM Cognos DecisionStream Series 7 Backup 52
  Import an IBM Cognos DecisionStream Series 7 Package 52
  Post Upgrade Considerations 52
    Command Lines in Scripts 52
    IBM Cognos Data Manager SQL.TXT Designer Definition Files 52
    Source Code Control Projects 52
    Metadata Deliveries 53
    DB-LIB Connections and Microsoft SQL Server Bulk Loader Connections 53
    IBM Cognos Data Manager Network Services 53

Chapter 6: SQLTerm and SQL Helper 55
  Create SQL Statements 55
    Add a SELECT Statement to an SQL Query 55
    Add a Table Name or Column Name to an SQL Query 57
    Add Other Statements to an SQL Query 58
    Additional SQLTerm Commands 58
  Test an SQL Query 60
    Include Substitution Variables in an SQL Query 61
    View the Executed Query 61

Chapter 7: Connections 63
  Create a Database Connection 63
Chapter 11: Reference Structures 87

View Reference Structures 87
  View Reference Structures in the Tree Pane 87
  View Reference Structures in the Visualization Pane 87
Use the Hierarchy Wizard 88
  Create the Hierarchy from the Columns of One Table (Star Schema) 89
  Create the Hierarchy from the Rows of One Table 89
  Create the Hierarchy from Multiple Tables (Snowflake Schema) 90
Hierarchies 91
  Create a Hierarchy 91
  Create Hierarchy Levels and Define the Attributes 92
  Reposition the Hierarchy Levels 93
  Create Recursive Levels 93
Auto-level Hierarchies 95
  Top Parent ID 96
  Create an Auto-level Hierarchy and Define the Attributes 97
Lookups 98
  Create a Lookup and Define the Attributes 99
Set Up Data Access 99
  Map the DataStream Items to Level Attributes 101
  Populate the Reference Structure 102
Foster Parents 103
  Specify a Name for a Foster Parent 104
  Disable Fostering 104
Null ID Values 105
Set Error Handling 105
  Foster Orphaned Members 106
  Multiple Parents 106
  Non-Unique IDs 107
  Duplicate Rows at a Level 108
  Null/Empty Captions 109
  Non-Unique Captions 109
  Unbalanced Hierarchy 110
  Circular References 110
Create a Date Hierarchy 112
Separate Reference Data and Fact Data 113
  Example Scenarios 114
Explore and Test a Reference Structure 117
  The Reference Explorer Window 118
Delete a Reference Structure 119

Chapter 12: Constructing and Transforming Hierarchies 121

Hierarchy Types 121
  Balanced Hierarchies 121
  Unbalanced Hierarchies 121
  Ragged Hierarchies 122
Balance a Hierarchy 122
Link a New Level to the Bottom of an Unbalanced Hierarchy 124
Flatten a Hierarchy 126
Chapter 13: Fact Builds 131

View Fact Builds 131
  View Fact Builds in the Tree Pane 131
  View Fact Builds in the Visualization Pane 131

The Fact Build Wizard 133
  IBM Cognos BI Mart (Star) and IBM Cognos BI Mart (Snowflake) 134
  Relational Datamart 134
  Data Transfer 134
  Summary of Fact Build Types in the Fact Build Wizard 134
  Use the Fact Build Wizard 135
  Automatic Source Table Joins Using the Fact Build Wizard 135

Create a Fact Build Manually 135
  Add a Fact Build 136
  Define the DataStream 136
  Define the Transformation Model 136
  Add Deliveries to the Fact Build 137
  Execute the Fact Build 137

Rename a Fact Build 137

Set the Date Source for Timestamping Data 137

Handle Duplicate Data 138

Delete a Fact Build 139

Chapter 14: DataStreams 141

View DataStreams 141
  View DataStreams in the Tree Pane 141
  View Fact Build DataStreams in the Visualization Pane 142

Set Up a DataStream 142

Manage Data Sources 142
  Add a Data Source for a SQL-based Source Data 143
  Add Published FM Package Data Source 145
  Specify Prepare or Parse 148
  Test a Data Source 149
  Change the Properties of a Data Source 149
  Disable and Enable a Data Source 150
  Define Literal Values in a Data Source 150
  Delete a Data Source 151

Manage DataStream Items 151
  Create a DataStream Item 151
  Map Data Source Columns to the DataStream 152
  Delete a DataStream Item 154

Derivations 155
  Where to Perform a Derivation 155
  Expressions in Derivations 155
  Add a Derivation 156

Reduce the Amount of Input Data 157
  Sample the Data 157
  Apply a Maximum Row Limit 158
  Apply Sample Frequency and a Maximum Row Limit 158
Chapter 15: Transformation Models 163

View Transformation Models 164
  View Transformation Models in the Tree Pane 164
  View Transformation Models in the Visualization Pane 164
Add an Attribute Element to a Fact Build 164
Add a Derivation Element to a Fact Build 165
Add a Dimension Element to a Fact Build 166
Add a Derived Dimension Element to a Fact Build 167
  Example 168
Add a Measure Element to a Fact Build 170
Associate a Dimension or Derived Dimension Element with a Reference Item 171
Set Input and Output Levels for a Dimension or Derived Dimension 173
  Set Input Levels 173
  Set Output Levels for a Dimension or Derived Dimension 175
  Exclude Input Data from Fact Build Output 176
Cache Reference Data on Demand 177
Add a Calculation to a Derivation or Derived Dimension 178
Dynamic and Reference Domains 179
  Dynamic Domains 179
  Reference Domains 183
  Specify the Domain Details 183
Aggregate Data 183
  Memory Considerations 184
  Determine the Aggregation Levels 184
  Specify that Aggregation is to be Performed 184
  Calculate a Derivation Before Aggregation 185
  Set Up the Regular Aggregate 185
  Set Up an Aggregate Rule 188
  Aggregate Duplicate Data 192
  Regular Aggregate Examples 193
  Aggregate Rules Example 195
Process Late Arriving Facts 197
  Late Arriving Fact Example 199
Merge Data for a Measure or an Attribute 200
Map the DataStream to the Transformation Model 201
  Map the DataStream to Existing Transformation Model Elements 202
  Map the DataStream and Create Transformation Model Elements 203
  Map the DataStream to Transformation Model Elements Automatically 204
Add an Element to the Deliveries 204
Change an Element Type 205
Reposition Elements 205
Delete an Element 205

8 IBM Cognos Data Manager
Chapter 16: Delivering Fact Data Using a Fact Build 207
Fact Delivery Modules 207
   Table Delivery Modules 207
   Non-table Delivery Modules 208
Add a Fact Delivery 208
Define the Properties for a Table Delivery 209
Define the Index Properties 211
Add Control Attributes to a Table Delivery 212
Deliver to Multiple Tables by Subscribing to Elements 212
Define the Properties for a Subscribed Element 214
Example 214
Change the Column Names of Delivered Data 215
Add Missing Columns to the Target Table 215
Handle Failed Rows in a Relational Table Delivery 216
Set Up Delivery Filters 217
   Create a Delivery Filter 218
   Use a Level Filter to Deliver Specific Dimensions and Hierarchy Levels 219
   Use an Output Filter to Deliver Specific Data Rows 220
   Apply a Delivery Filter to a Fact Delivery 221
   Delete a Delivery Filter 221
Partition Columns Horizontally Across a Table 221
Disable and Enable a Fact Delivery 224
Change the Delivery Module for a Fact Delivery 225
Delete a Fact Delivery 225

Chapter 17: Delivering Dimension Data Using a Fact Build 227
Dimension Delivery Modules 227
Add a Dimension Delivery 227
Add a Table to a Dimension Delivery 228
Disable and Enable a Dimension Delivery 229
Delete a Dimension Delivery 230

Chapter 18: Dimension Builds 231
View Dimension Builds 232
   View Dimension Builds in the Tree Pane 232
   View Dimension Builds in the Visualization Pane 232
Use the Dimension Build Wizard 232
Schema Types 233
   Star Schema 233
   Optimal Star Schema 233
   Snowflake Schema 234
   Optimal Snowflake Schema 234
   Parent-Child Schema 234
Create a Dimension Build Manually 235
   Add a Dimension Build 235
   Add a Table to a Dimension Build 236
   Column Tokens 237
Track Attribute Changes 238
   Overwrite Attribute Changes (Type 1 Changes) 239
Track Attribute Changes (Type 2 Changes)  239
Corrective Changes (Type 0 Changes)  240
Dimensional History  241
Set Up Dimensional History  242
Reject Late Arriving Dimension Details  244
Specify the Source of the Effective Start Date in Initial Records  246
Specify Effective Date Options  247
Define the Index Properties  248
Set the Date Source for Timestamping Data  249
Delete a Dimension Build  250
Examples of how IBM Cognos Data Manager Maintains Dimension Tables  250
Example 1: Simple Dimension Table With No Tracking Behavior  250
Example 2: Dimension Table With Attribute Change Tracking Behavior  251

Chapter 19: Managing Builds Using JobStreams  255
View JobStreams  257
View JobStreams in the Tree Pane  257
View JobStreams in the Visualization Pane  257
Create a JobStream  258
Define Logging and Audit Information for a JobStream  259
Define JobStream Variables  259
Add a Fact Build, Dimension Build, or JobStream Node to a JobStream  259
Add a Procedure or Condition Node to a JobStream  261
Add an SQL Node to a JobStream  262
Add an Alert Node to a JobStream  263
Add an Email Node to a JobStream  264
Add a DataStage Node to a JobStream  265
Delete a Node from a JobStream  266
Link Nodes in a JobStream  267
Change the True and False Status for a Condition Node  268
Split Large JobStreams into Smaller JobStreams  268
Specify Error Handling in JobStream Nodes  269
Specify the Load Required to Process a Fact Build or Dimension Build Node  269

Chapter 20: Publishing Data Movement Tasks  271

Chapter 21: Executing Builds and JobStreams  273
Build Execution Modes  273
Execute a Fact Build or Dimension Build on your Local Computer  273
Check the Target Table Prior to Executing a Build  275
Fine-Tune the Data Structure  276
Execute a JobStream on your Local Computer  276
Exclude Nodes from Processing in a JobStream  278
Execute Procedure and SQL Nodes as Separate Processes  278
Restart Processing a JobStream  279
Execute a Build or JobStream on a Remote Computer  279
Configure a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing  281
Set the Default Execution Method  281
Execution Log Files  282
Change the Default Location for Log Files 282
Specify Log File Details 283
Specify the Message Frequency in a Build Log File 283
Include User Log Messages for JobStream Nodes 284
View the Log Files 284
Save Audit Details 285
  The Audit History 285
  The Audit Trail 287
  The Audit Messages 289
  The JobStream Details 290
  Stop Execution 291
  The Audit Tables 291
Create a Log File for Expression and Script Tracing 292
  Specify Trace Log File Details 292
  Example Trace Log Files 293

Chapter 22: Exporting Metadata 295
  View Metadata 295
    View Metadata in the Tree Pane 295
    View Metadata in the Visualization Pane 295
  Create a Metadata Dimension 297
  Create a Metadata Collection 298
    Add a Star Model to Metadata Collection 298
  Export Metadata 301

Chapter 23: User-Defined Functions 305
  Internal User-Defined Functions 305
    Create an Internal User-Defined Function 305
  External User-Defined Functions 307
    Declare an External User-Defined Function 308
    Special Data Types for External User-Defined Functions 309
    Allocating and Freeing Memory 313
    Execution Modes for External User-Defined Functions 314
    Macros in External User-Defined Functions 314
    Example External User-Defined Functions 315
    Register External User-Defined Functions 315
    Cross-Platform Portability 316
  Maintain User-Defined Functions 316
    Use a User-Defined Function 316
    Delete a User-Defined Function 317
    Establish the Dependencies of a User-Defined Function 317

Chapter 24: Variables 319
  User-Defined Variables 320
    User-Defined Variable Names 320
    Notation of Variables and Substitution Variables 320
    Order of Precedence 320
    Add a User-Defined Variable 321
    Reposition User-defined Variables 323
    Edit or Delete a User-Defined Variable 323
Table of Contents

Property Variables 323
  System Variables 324
  Run-time Variables 329
  Debug-Related Variables 330
  Audit-related Variables 332
  Performance-related Variables 333
  DBMS-related Variables 335

Chapter 25: Testing Expressions and Scripts 341
  Test an Expression or Script 342
  Scope of Expressions and Scripts 342
  Data Types in Expressions and Scripts 343
  Default Values in Expressions and Scripts 345

Chapter 26: Multi-Developer Support 347
  Source Code Control 347
    Guidelines for Planning a Source Code Control Project 348
    Enable Source Code Control 348
    Add a Catalog to Source Code Control 348
    Create a Shared Copy of a Catalog from Source Code Control 349
    View the Source Code Control Properties of a Component 350
    Refresh the Status of Components 350
    Obtain the Latest Version of a Catalog 350
    Check Out Catalog Components 351
    Check a Catalog into Source Code Control 352
    Undo a Catalog Check Out 352
    Open the Source Code Control System from IBM Cognos Data Manager 352
    Detach a Catalog from Source Code Control 353
    Create a Standalone Copy of a Catalog from Source Code Control 353
    Temporary Files Used By Source Code Control 354
  Component Packages 354
    Create a Component Package 354
    Import a Component Package 355
    Combine the Effort of Several Developers 359
  Backup and Restore Catalogs 359
    Backup an IBM Cognos Data Manager Catalog 360
    Restore an IBM Cognos Data Manager Catalog 360

Chapter 27: Using Pivots to Rotate Data 363
  Create Pivots 363
  Examples 364
    Example 1: Create a Single Pivot 365
    Example 2: Create a Double Pivot 367
    Example 3: Create Multiple Pivot Values 369

Chapter 28: Handling Data Rejected by a Fact Build 371
  Rejected Source Data 371
    Reject Duplicate Source Data 371
    Reject Unmatched Members 372
    Save Rejected Source Data 372
Rejected Delivery Data 373
  Use a Level Filter to Reject Data 373
  Use an Output Filter to Reject Data 374
  Save Data Rejected During Delivery 374
  Specify the Encoding to Use for Reject Files 374

Chapter 29: Managing Memory 377
  Structure Data 377
  Dimension Domains 377
  The Hash Table 377
  The Page Pool 378
  The Page Table 378
  Dimension Breaks 378
    Determine the Dimensions on which to Break 379
    Set Dimension Breaks 379
    Specify that Sorting is to be Performed 379
    Specify When to Break 380
  Processing Data Sources in Series or in Parallel 380
  Memory Limits 381
    Maximum Memory Usage 381
    Maximum Page Size 382
    Initial Size of the Page Table 382
    Initial Size of the Hash Table 382
    Set Memory Limits 383

Chapter 30: IBM Cognos Data Manager SQLTXT Designer 385
  SQLTXT Definition Files 385
  SQLTXT Text File Format 386
    Wildcards in SQLTXT File Names 386
    SQLTXT Column Names 386
    SQLTXT Record Types 386
    SQLTXT Record Delimiters 386
    SQLTXT Column Delimiters 387
    Quotation Characters in SQLTXT Delimited Records 388
    Special Characters as Delimiters 388
    SQLTXT Data Types 389
    Date String Formats in SQLTXT Files 394
    Error Checking in SQLTXT 394
    Null Values in SQLTXT 395
    Change the Font in IBM Cognos Data Manager SQLTXT Designer 396
  Open a SQLTXT Database to Access Text Files 397
  Import a Table Definition using the Import Wizard 397
    Import Delimited Records with Delimited Columns 397
    Import Delimited Records With Fixed Width Columns 399
    Import Nondelimited Records 401
  Define a Table Manually 402
    Add Columns to a Table 403
    Create a Similar Table 404
    Edit a Column or a Table 404
    Delete a Column or a Table 404
<table>
<thead>
<tr>
<th>Delivery Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essbase Direct Feed Delivery Module</td>
<td>458</td>
</tr>
<tr>
<td>Essbase File Feed Delivery Module</td>
<td>460</td>
</tr>
<tr>
<td>Informix LOAD Delivery Module</td>
<td>460</td>
</tr>
<tr>
<td>Microsoft SQL Server BCP Command Delivery Module</td>
<td>462</td>
</tr>
<tr>
<td>Netezza NZLoad Delivery Module</td>
<td>465</td>
</tr>
<tr>
<td>ORACLE SQL*Loader Delivery Module</td>
<td>468</td>
</tr>
<tr>
<td>Red Brick Loader (TMU) Delivery Module</td>
<td>471</td>
</tr>
<tr>
<td>Relational Table Delivery Module</td>
<td>474</td>
</tr>
<tr>
<td>SQL Server Bulk Copy via API Delivery Module</td>
<td>489</td>
</tr>
<tr>
<td>Sybase ASE BCP Command Delivery Module</td>
<td>490</td>
</tr>
<tr>
<td>Sybase IQ LOAD Delivery Module</td>
<td>493</td>
</tr>
<tr>
<td>Teradata Fastload Delivery Module</td>
<td>495</td>
</tr>
<tr>
<td>TM1 Turbo Integrator Delivery Module</td>
<td>498</td>
</tr>
<tr>
<td>Teradata Multiload Delivery Module</td>
<td>502</td>
</tr>
<tr>
<td>Teradata TPump Delivery Module</td>
<td>505</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Properties</td>
<td>456</td>
</tr>
<tr>
<td>Element Properties</td>
<td>458</td>
</tr>
<tr>
<td>Module Properties</td>
<td>458</td>
</tr>
<tr>
<td>Element Properties</td>
<td>459</td>
</tr>
<tr>
<td>Module Properties</td>
<td>460</td>
</tr>
<tr>
<td>Element Properties</td>
<td>460</td>
</tr>
<tr>
<td>Module Properties</td>
<td>461</td>
</tr>
<tr>
<td>Element Properties</td>
<td>462</td>
</tr>
<tr>
<td>Module Properties</td>
<td>463</td>
</tr>
<tr>
<td>Element Properties</td>
<td>465</td>
</tr>
<tr>
<td>Module Properties</td>
<td>465</td>
</tr>
<tr>
<td>Element Properties</td>
<td>467</td>
</tr>
<tr>
<td>Module Properties</td>
<td>468</td>
</tr>
<tr>
<td>Element Properties</td>
<td>471</td>
</tr>
<tr>
<td>Module Properties</td>
<td>472</td>
</tr>
<tr>
<td>Element Properties</td>
<td>473</td>
</tr>
<tr>
<td>Module Properties</td>
<td>474</td>
</tr>
<tr>
<td>Update/Insert Delivery Method</td>
<td>476</td>
</tr>
<tr>
<td>Update Delivery Method</td>
<td>483</td>
</tr>
<tr>
<td>Element Properties</td>
<td>489</td>
</tr>
<tr>
<td>Module Properties</td>
<td>490</td>
</tr>
<tr>
<td>Element Properties</td>
<td>490</td>
</tr>
<tr>
<td>Module Properties</td>
<td>491</td>
</tr>
<tr>
<td>Element Properties</td>
<td>492</td>
</tr>
<tr>
<td>Module Properties</td>
<td>493</td>
</tr>
<tr>
<td>Element Properties</td>
<td>495</td>
</tr>
<tr>
<td>Module Properties</td>
<td>496</td>
</tr>
<tr>
<td>Element Properties</td>
<td>498</td>
</tr>
<tr>
<td>Module Properties</td>
<td>499</td>
</tr>
<tr>
<td>Element Properties</td>
<td>501</td>
</tr>
<tr>
<td>Module Properties</td>
<td>503</td>
</tr>
<tr>
<td>Element Properties</td>
<td>505</td>
</tr>
<tr>
<td>Module Properties</td>
<td>506</td>
</tr>
<tr>
<td>Element Properties</td>
<td>508</td>
</tr>
</tbody>
</table>
Table of Contents

Text File Delivery Module 508
Module Properties 509
Element Properties 510
Specify Executables to Deliver Data 511
Set the Name of the LoaderExecutable in a Module Property 511
Set the Name of the Executable in an Environment Variable 512
Specify the Location of Executables 513

Appendix B: Database Drivers 515
IBM Cognos Data Source Driver 516
DB2 DBMS Driver 516
  Data Type Conversions 516
  Date Format Conversions 518
DTS Package Driver 518
Essbase DBMS Driver 518
INFORMIX (via Connect) DBMS Driver 519
  Data Type Conversion 519
  Date Format Conversion 519
ODBC DBMS Driver 520
  SQL Syntax 520
  Network Data Sources 520
  Connecting to ODBC Data Sources 520
  Data Type Conversion 520
  Date Format Conversion 521
Oracle (via SQL*Net) DBMS Driver 521
  Data Type Conversion 521
  Date Format Conversion 522
Published FM Package Driver 522
Microsoft SQL Server (via OLE-DB) DBMS Drivers 522
  Data Type Conversions 522
  Date Format Conversions 524
SAP R/3 DBMS Driver 524
  Data Type Conversion 524
  Date Format Conversion 525
SQLTXT DBMS Driver 525
  Data Type Conversion 525
  Date Format Conversion 528
Sybase (via CT-Library) DBMS Driver 528
  Data Type Conversion 528
  Date Format Conversion 529
TM1 DBMS Driver 529

Appendix C: Audit Tables 531
  Component Run Table (dsb_component_run) 531
  Run Context Table (dsb_run_context) 532
  Delivery History Table (dsb_delivery_hist) 533
  Audit Trail Table (dsb_audit_trail) 533
  Audit Message Table (dsb_audit_msg) 534
  Audit Message Line Table (dsb_audit_msg_line) 534
  Job Node Run Table (dsb_jobnode_run) 535
## Appendix D: Date, Time, and Interval Formats 537

### Date Formats 537
- Examples 539
- Default Century Behavior 539

### Time Formats 540
- Examples 541

### Interval Formats 541
- Day to Second Intervals 541
- Year to Month Intervals 542

## Appendix E: Data Types 543

### Rules for Performing Calculations on Numbers 545
- Example Calculations on Numbers 547

## Appendix F: Buttons and Icons 549

### Toolbar Buttons 549
- Standard Toolbar Buttons 549
- JobStream Toolbar Buttons 550
- SQLTerm and Data Source Toolbar Buttons 551
- Reference Explorer Toolbar Buttons 552

### Fact Build Visualization Icons 552

### Dimension Build Visualization Icons 554

### Metadata Visualization Icons 555

### Reference Structure Visualization Icons 555

### JobStream Visualization Icons 556

### Source Code Control Icons 557

### Audit History Icons 557

## Appendix G: Cross Platform Deployment 559

### Use IBM® Cognos® Data Manager Designer to Deploy on UNIX® 559
- File Names to Use on UNIX 559
- Database Connections 559

### Execute a Fact Build under UNIX 562
- Example 563

## Appendix H: Problems When Using IBM Cognos Data Manager 565

### Source Code Control 565

### Writing of Large Objects to DB2 via Bulk Loader 565

### 32 bit and 64 bit Database Clients 565

### Error Message Codes 565

### Bind Error When Using SQL Server Bulk Copy via API Delivery Module 565

### Netezza ODBC 3.0 Driver Incompatible With IBM Cognos Data Manager 566

### Limited Shared Memory on AIX 566

### Additional Permissions Required for DB2 Connection 566

### TM1 Turbo Integrator delivery via the Data Movement service fails when sharing a local folder 566

## Glossary 567

## Index 581
Introduction

This document is intended for use with IBM® Cognos® Data Manager.

If you are using Data Manager for the first time, we recommend that you first read IBM® Cognos® Data Manager Getting Started and perform the lessons that it contains.

The examples in this document use Bakus-Naur Form (BNF) to describe the syntax of the Data Manager language. The BNF syntax consists of the following:

- A set of terminal symbols that are the words, commands, or punctuation of the language and command-line interface.

- A set of non-terminal symbols that are essentially place holders. They appear in angled brackets < >, for example, <refdata_file>. A definition of each non-terminal symbol may appear elsewhere in the syntax definition. However, not all non-terminal symbols are defined. For a complete definition of the scripting language, including the description of all non-terminal symbols, see the IBM® Cognos® Function and Scripting Reference Guide.

- A set of rules that you apply when interpreting the BNF definitions:
  - ::= means 'is defined to be'.
  - Square brackets [ ] indicate that the enclosed symbols are optional.
  - Braces { } indicate that the enclosed symbols may be repeated zero or more times.
  - The piping symbol | indicates that you should choose only one of the items that it separates.

The following example defines the <options> symbol to be an optional -C, followed by a <var_list> symbol. It then defines the <var_list> symbol to be zero, one, or more instances of -V followed by a <name>=<value> pair:

```plaintext
<options> ::= [-C] <var_list>
<var_list> ::= { -V <name> = <value>
```

**Audience**

You should be familiar with Microsoft® Windows® and SQL. You should also have an understanding of multi-dimensional data analysis or Business Intelligence.

**Finding information**

To find IBM® Cognos® product documentation on the web, including all translated documentation, access one of the IBM Cognos Information Centers at http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp. Updates to Release Notes are published directly to Information Centers.

You can also read PDF versions of the product release notes and installation guides directly from IBM Cognos product disks.
Accessibility features
This product does not currently support accessibility features that help users with a physical disability, such as restricted mobility or limited vision, to use this product.

Forward-looking statements
This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer
The Great Outdoors Company, GO Sales, any variation of the Great Outdoors name, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values is coincidental. Other sample files may contain fictional data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.
Chapter 1: What’s New?

This section contains a list of new and changed features for this release. It also contains a cumulative list of similar information for previous releases. It will help you plan your upgrade and application deployment strategies and the training requirements for your users.

For information about new features for this release, see the IBM® Cognos® New Features Guide.

For changes to previous versions, see:
- New Features in Version 8.4
- Changed Features in Version 8.4

To review an up-to-date list of environments supported by IBM® Cognos® products, such as operating systems, patches, browsers, Web servers, directory servers, database servers, and application servers, visit the IBM Cognos Customer Center (http://www-01.ibm.com/software/data/cognos/customercenter/).

New Features in Version 10.1.0

Listed below are the new features since the last release. Links to directly-related topics are included.

Configuring a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing

It is now possible to utilise IBM® Cognos® Business Intelligence load balancing for executing fact build and dimension build nodes in a JobStream by choosing to execute these nodes as individual data movement tasks. For more information, see "Configure a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing" (p. 281).

Changed Features in Version 10.1.0

Listed below are changes to features since the last release. Links to directly-related topics are included.

TM1 Turbo Integrator Delivery Module

The following enhancements have been made to the TM1® turbo integrator delivery module:
- It is now possible to deliver data directly to IBM® Cognos® TM1, instead of using staged files.
- When delivering data, you can truncate an existing TM1 cube by resetting its values to zero, and specifying which view of a TM1 cube to truncate (if required).
- It is possible to set a variable to ignore minor errors while delivering data.
New Features in Version 8.4

Listed below are the new features since the last release. Links to directly-related topics are included.

DataStage Node

A new JobStream node has been added that allows you to execute IBM® DataStage® jobs within a JobStream. For more information, see "DataStage Node" (p. 257).

New Bulk Loader Delivery Modules

New bulk loader delivery modules have been added

- Netezza NZLoad
- Sybase ASE BCP Command
- Sybase IQLoad
- TM1 Turbo Integrator

Multiple Data Sources in IBM Cognos Connection

A new environment variable named DM_DATASOURCE_CONNECTION_SIGNONS has been introduced that allows multiple data source connections when using IBM® Cognos® Connection. This variable specifies the data source or data sources to use. For more information, see "DM_DATASOURCE_CONNECTION_SIGNONS" (p. 324).

Changed Features in Version 8.4

Listed below are changes to features since the last release. Links to directly-related topics are included.

Credentials

The way in which catalog credentials are stored has been improved.

When you are managing catalog credentials within IBM® Cognos® Data Manager, if there are no catalog credentials stored, or they are invalid, a status message appears to indicate this. The user whose credentials you want to store must log on and store or renew them.

In addition, if catalog credentials are stored, but they do not match the current Designer credentials that you have used to log on, a status message appears to indicate this. If required, you can store your current Designer credentials as the catalog credentials.

For more information, see "Manage Credentials" (p. 71).
Email Nodes

A BCC box has been added to the email JobStream node. This allows you to send a copy of an email to a recipient, without the recipient’s name being visible to other recipients of the message.

In addition, a new environment variable named DM_EMAIL_VIA_SERVICE has been introduced that allows you to specify that the IBM® Cognos® 8 delivery service is used for the delivery of email nodes rather than MAPI. For more information, see "DM_EMAIL_VIA_SERVICE" (p. 326).

New Features in Version 8.3

Listed below are new features since the last release. Links to directly-related topics are included.

Support for Windows Vista

IBM® Cognos® Data Manager is supported on Microsoft® Windows® Vista. With Windows® Vista, Microsoft® introduced security enhancements to further protect the Program Files directory. This necessitated changes to environment variables and the directories where user profiles are stored. For more information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Support for IBM Cognos Data Sources

You now have the ability to utilize IBM® Cognos® data sources in IBM® Cognos® Data Manager. For more information, see "Create an IBM Cognos Data Source Connection" (p. 66).

Enhanced Unmatched Member Support

IBM® Cognos® Data Manager now supports the writing back of unmatched members to the same dimension source when a dimension is used simultaneously by multiple fact builds. For more information, see "Distributed Handling of Unmatched Members" (p. 181).

Enhanced Semi-Aggregate Handling

You now have the ability to allow multiple aggregate rules for a measure element, and to extend the available set of aggregate functions. For more information, see "Aggregate Data" (p. 183).

Load Control

To prevent overloading a system, you now have the ability to restrict the execution of fact build and dimension build nodes in a JobStream. For more information, see "Specify the Load Required to Process a Fact Build or Dimension Build Node" (p. 269).

Enhanced Error Reporting for Relational Delivery Exceptions

You now have the ability to configure IBM® Cognos® Data Manager to report both the error reason and the failed row of data when a relational delivery exception is encountered. The failed row of data can be delivered to a file or to a relational table. For more information, see "Handle Failed Rows in a Relational Table Delivery" (p. 216).
Chapter 1: What's New?

**Changed Features in Version 8.3**

Listed below are changes to features since the last release. Links to directly-related topics are included.

**Error Checking in SQLTXT Files**

The way in which IBM® Cognos® Data Manager handles errors in SQLTXT files has changed. Previously, the columns validated by SQLTXT depended on the action specified by the user. Now, SQLTXT only validates those columns specified in the SELECT statement, and ignores any other columns included in the SQLTXT definition file. For more information, see "Error Checking in SQLTXT" (p. 394).

**Aggregation Exception Handling**

Aggregation exception handling has been superseded by aggregate rules. When you open a catalog that contains aggregation exceptions created in a previous version of IBM® Cognos® Data Manager, the aggregation exceptions are automatically converted to aggregate rules.
The main purpose of IBM® Cognos® Data Manager is to create data warehouses and data repositories for reporting, analysis, and performance management. Data Manager does this by

- extracting operational data from multiple sources
- transforming and merging the data to facilitate enterprise-wide reporting and analysis
- delivering the transformed data to coordinated data marts

Data Manager can be used to transfer data into single database tables and, where more complex transformations are required, to populate data warehouse fact tables and dimension tables.

Data Manager integrates with other IBM® Cognos® Business Intelligence products by delivering metadata to IBM® Cognos® Framework Manager. This allows target data warehouse and data repositories to be modeled and used in IBM® Cognos® Business Intelligence and Performance Management projects.

Building the Data Foundation for the IBM Cognos Business Intelligence Solution

IBM® Cognos® Data Manager can deliver data to any supported target database into any design of database schema. However, Data Manager is best suited to delivering data into dimensional data warehouses and coordinated data marts. This approach has several advantages over non-dimensional data warehouses and uncoordinated data marts.

Dimensional data warehouses are typically made up of a series of coordinated data marts. The key to coordinated data marts is that common dimensions are shared across subject areas. This offers a common view of your business, while allowing flexibility and rapid incremental deployment. This approach is well suited to business intelligence because it allows a fast and iterative development approach to help the successful delivery of projects.

Data Manager supports this development approach by providing many of the tools and functionality required for the delivery of subject orientated data warehouses. The Data Manager dimensional framework allows the shared dimensions, known as conformed dimensions, to be constructed and then shared across all related subject areas. This reuse of the dimensional framework helps ensure that each subject area can be designed incrementally, but integrated consistently with any existing subject areas within the data warehouse.

You can choose to build enough dimensions and fact tables to complete one or more subject areas and then deploy the model to IBM® Cognos® Framework Manager for further modeling. When the model has been published, it can be used for reporting and analysis. Data Manager allows you to build on the model you have created and extend it to additional areas while maintaining maximum reuse of any completed work.
Data Manager provides all of the techniques required to easily build a scalable data foundation for your Business Intelligence project.

**Business Dimensions**

Business dimensions are the core components or categories of a business, anything that you want to analyze in reports. Business dimensions are fundamental to data organization and provide context for numeric data items, or measures.

For example, a retail chain-store may categorize its sales data by the products that it sells, by its retail outlets, and by fiscal periods. This organization has the business dimensions Product, Location, and Time. The measures of the business, such as how much it sells, lie at the intersection of these dimensions. The following illustration shows these dimensions as the axis of a three-dimensional space. The cube at the center of this space represents 100 units of widgets sold in Montana during July.

You can derive summary information by aggregating data along one or more dimensions. For example, the following illustration shows the aggregation of data along the Location dimension to give the total sales of widgets during July.
To enable aggregation, business dimensions must have a hierarchy structure.

### Conformed Data Warehouses

Each set of business dimensions and fact tables can be referred to as a subject oriented data mart because it can be used to address the reporting and analysis requirements for a single subject area of the business. This approach allows for data structures that are very well suited to analysis by business users and are fast to report against and analyze.

By combining multiple subject areas using shared common dimensions it becomes possible to navigate across multiple subject areas while maintaining a consistent view of the business. This combination of multiple dimensional subject areas is known as a dimensional data warehouse.

The key is to coordinate data marts by sharing common dimensions across many subject areas. This strategy offers a common view of your business, which provides flexibility and allows rapid, incremental deployment.

Coordinated data marts rely on a shared dimensional structure to model the key aspects of your business. The key aspects of your business may be, for example, your products and customers, and the time periods over which you account for sales. Because all data marts share this structure, each dimension provides a single view of the business aspect that it represents. Such dimensions are known as conformed dimensions and provide the basis for integration of dependent data marts.

This structure forms the framework on which you build separate, but coordinated, data marts. As each new data mart comes online, it integrates consistently with the existing data marts. Eventually, enough data marts exist to provide an integrated, corporate data warehouse. The IBM® Cognos® Data Manager dimensional framework is key to ensuring that all subject dimensional areas are built using a common dimensional structure and that maximum reuse is made of shared dimensions. This allows for rapid, but incremental development of the dimensional data warehouse.
Scalable Dimension Architecture

IBM® Cognos® Data Manager is an extremely flexible and responsive data transformation tool. It uses an innovative dimensional reference model to coordinate the management of data marts of all shapes and sizes and across many different platforms. In addition, Data Manager can perform massive-scale data merging and aggregation in a fraction of the time possible with hand-made solutions or with SQL-based data warehouse loaders.

IBM Cognos Data Manager Environments

IBM® Cognos® Data Manager has two environments:

- Data Manager Designer
- the Data Manager engine, or run-time environment

IBM Cognos Data Manager Designer

In IBM® Cognos® Data Manager Designer, you use builds to specify a set of processing rules that determine how Data Manager acquires data from the source databases, transforms the data, and delivers it to the target database. This information is stored in a Data Manager catalog. You then execute the builds using the Data Manager engine that can be either on UNIX® or Microsoft® Windows® operating systems.

The following illustration shows a model of the Data Manager architecture.
To build a data warehouse or data mart using Data Manager you typically

- create a Data Manager catalog
- set up database connections
- create reference dimensions and templates
- define the reference structures (that is, the hierarchies, auto-level hierarchies, and lookups)
- create the required fact builds and dimension builds
- add and define the fact deliveries and the dimension deliveries
- execute builds
- create a JobStream if required to coordinate the builds with other procedures related to managing the data mart

The IBM Cognos Data Manager Engine

The IBM® Cognos® Data Manager engine consists of a number of programs that you can invoke either from Data Manager Designer (with Microsoft® Windows® operating system) or directly on the command line (with Windows® and UNIX® operating systems).

For most applications, IBM® Cognos® recommends that you design and prototype using Data Manager Designer on a Windows® computer. You can then deploy your builds to either a Windows® or a UNIX® computer that has the Data Manager engine installed. However, you can also program the Data Manager engine directly in the Data Manager language.

IBM Cognos Data Manager Language

The IBM® Cognos® Data Manager language is declarative; that is, you specify what must be rather than what Data Manager must do. A Data Manager project controls the acquisition, transformation, and delivery of data. Data Manager lets you store projects

- in text files
- in a catalog that resides in a database
Text-based Projects

In a text-file specification, a project consists of up to three text files, with each text file controlling some aspects of the overall process:

- database alias definition file
- reference structure definition file
- build definition file

You can use each definition file for more than one project. For example, you can define one alias definition file and one reference structure definition file that you use with all your projects.

Catalog-based Projects

In a catalog-based specification, the project information is stored in special tables in a database. To edit the contents of a catalog, you must either use IBM® Cognos® Data Manager Designer or export the definition to a temporary text file, which you edit, and then import the modified definition to the catalog.

Like text-based projects, each catalog-based project consists of a set of aliases, reference structures, and a build. Each performs the same function as it does in a text-based project.

Packaged-based Projects

In a package-based specification, the components in the package are saved in a non-editable.pkg file.

You can use a package to move components between catalogs, partially backup and restore a catalog, combine the efforts of several developers, and distribute catalogs or components.

From the command line interface, you can execute fact builds, dimension builds, and JobStreams from within a package rather than from a catalog. This means that the IBM® Cognos® Data Manager engine can operate in an embedded environment without using a Data Manager catalog.

Application Limits

The following table gives limits on IBM® Cognos® Data Manager features. Where a value is given as unlimited, available memory, disk space, or processor specification may impose a practical limit.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum levels in a hierarchy.</td>
<td>Unlimited.</td>
</tr>
<tr>
<td>Maximum attributes in a reference structure or template.</td>
<td>Unlimited.</td>
</tr>
<tr>
<td>Feature</td>
<td>Limit</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Maximum transformation model elements in a fact build.</td>
<td>Unlimited. However, the maximum number of elements that Data Manager can deliver to a target table is limited by the target DBMS.</td>
</tr>
<tr>
<td>Maximum columns in a query.</td>
<td>Limited by the source or target DBMS.</td>
</tr>
<tr>
<td>Maximum deliveries in a fact build.</td>
<td>Unlimited.</td>
</tr>
<tr>
<td>Maximum size of temporary files.</td>
<td>Limited by the operating system or file system.</td>
</tr>
</tbody>
</table>

**Multi-platform**

For information, see [http://www.ibm.com/software/data/support/cognos_crc.html](http://www.ibm.com/software/data/support/cognos_crc.html).
Chapter 3: Getting Started with IBM Cognos Data Manager Designer

The IBM® Cognos® Data Manager Designer window consists of

- the menu bar from which you can select an option relevant to the function that you are performing
- the toolbar, which provides quick access to the main features of Data Manager
  For information about the toolbar buttons, see "Toolbar Buttons" (p. 549).
- the Tree pane
- the Visualization pane

The Tree Pane

The Tree pane is on the left of IBM® Cognos® Data Manager Designer window. At the top level, it has folders named Builds and JobStreams, Metadata, and Library.

At the bottom of the Tree pane there are a number of tabs:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Tab name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Folder Icon]</td>
<td>Catalog</td>
<td>Shows the full catalog</td>
</tr>
<tr>
<td>![Folder Icon]</td>
<td>Fact Builds</td>
<td>Shows only the fact builds contained in the catalog</td>
</tr>
<tr>
<td>![Folder Icon]</td>
<td>Dimension Builds</td>
<td>Shows only the dimension builds contained in the catalog</td>
</tr>
<tr>
<td>![Folder Icon]</td>
<td>JobStreams</td>
<td>Shows only the JobStreams contained in the catalog</td>
</tr>
<tr>
<td>![Folder Icon]</td>
<td>Library</td>
<td>Shows only the dimensions, connections, and functions contained in the catalog</td>
</tr>
<tr>
<td>![Folder Icon]</td>
<td>Metadata</td>
<td>Shows only the metadata dimensions and collections contained in the catalog</td>
</tr>
</tbody>
</table>

The Builds and JobStreams Folder

This folder contains the fact builds, dimension builds and JobStreams in the current catalog.

For more information, see "Fact Builds" (p. 131), "Dimension Builds" (p. 231), and "Managing Builds Using JobStreams" (p. 255).
It can also contain user-defined folders that you set up. For more information see "Set Up User-Defined Folders" (p. 34).

The Metadata Folder

This folder contains the metadata dimensions and collections of star schemas that represent the conformed model that you are building.

For more information, see "Exporting Metadata" (p. 295).

The Library Folder

This folder contains all the connections, reference dimensions (including hierarchies, auto-level hierarchies, lookups, and templates), and user-defined functions in the current catalog.

For more information, see "Connections" (p. 63), "Reference Dimensions" (p. 75), "Templates" (p. 77), "Reference Structures" (p. 87), and "User-Defined Functions" (p. 305).

Set Up User-Defined Folders

You can group together facts builds, dimension builds, and JobStreams into user-defined folders. The advantage of using folders is the improved usability of catalogs that contain a large number of builds or JobStreams.

It is also possible to set up a sub folder within a folder, up to a maximum of 32.

Note: User-defined folders are shown on the catalog tab only.

Steps to Create a User-Defined Folder

1. Click the Builds and JobStreams folder.
2. From the Insert menu, click Folder.
3. In the Name box, type a unique name for the folder. If you require, type a business name and description.
4. Click OK.

Steps to Create a Sub Folder

1. Click the required user-defined folder.
2. From the Insert menu, click Folder.
   
   The Folder Properties dialog box appears.
3. In the Name box, type a unique name for the folder. If you require, type a business name and description.
4. Click OK.
IBM® Cognos® Data Manager adds a sub folder to the user-defined folder.

**Steps to Move Builds and JobStreams Into a Folder**
1. Click the required build or JobStream.
2. From the Actions menu, click Move to Folder.
   The Move to Folder dialog box appears.
3. Select the folder to which you want to add the build or JobStream.
   - To create a folder rather than use an existing one, click New, and then type a unique name for the folder.
   - To move a build or JobStream from a user-defined folder to the Builds and JobStreams folder, click the catalog node.
4. Click OK.

**Tips**
- You can move a build, JobStream, or folder by dragging and dropping it into the required folder.
- You can select multiple items to move into a single folder by holding down the Shift key or the Ctrl key as appropriate as you make your selections.

**Delete a User-Defined Folder**
You can delete user-defined folders that you no longer require.

*Note:* When you delete a user-defined folder, any builds or JobStreams that it contains are also deleted. Ensure you move all required builds and JobStreams out of a folder before deleting it.

**Steps**
1. Click the required user-defined folder.
2. From the Edit menu, click Delete.
   If the folder is not empty, a message appears prompting you to confirm whether you want to delete the selected folder.
3. Click Yes.

**Change the Default Folder Settings**
By default, when a catalog is opened, the top level folders are shown open on the Catalog tab. However, you can change this.

**Steps**
1. From the Tools menu, click Options.
2. Click the General tab.

3. Clear the Expand root folders when catalog is opened check box.

4. Click OK.

Select Multiple Components
You can select more than one component in the Tree pane. This is useful if you want to move more than one component, perhaps to a user-defined folder.

Step
- Hold down the Shift key or the Ctrl key as appropriate, and then click the required components.

The Visualization Pane
The Visualization pane shows information about the selected fact build, dimension build, reference structure, metadata export, or JobStream.

You can right-click an item in the Visualization pane to show a menu that allows you to perform many of the actions that are also available from the Tree pane or the menu bar. For example, you can
- add data sources to a fact build or reference structure
- change the element type in the transformation model
- add levels to a hierarchy
- add delivery modules to a delivery
- export metadata
- execute DataStreams

For more information, see "View Fact Builds in the Visualization Pane" (p. 131), "View Dimension Builds in the Visualization Pane" (p. 232), "View Reference Structures in the Visualization Pane" (p. 87), "View JobStreams in the Visualization Pane" (p. 257), or "View Metadata in the Visualization Pane" (p. 295).

IBM® Cognos® Data Manager uses default settings to determine certain information in the Visualization pane. You can
- change the default settings
- change the size of the information shown
- reveal or hide the pane
- copy the information shown to the clipboard
Change the Default Settings in the Visualization Pane

You can change the default settings in the Visualization Pane.

Steps
1. From the Tools menu, click Options.
2. Click the General tab.
3. Enter values as required
   - Show transformation model elements in the visualization by default.
   - Maximum number of elements to show in the visualization
     Note: Limiting the number of elements shown in the Visualization pane, may mean that mapping connections on other visualization tabs cannot be shown.
     By default, IBM® Cognos® Data Manager shows a maximum of 100 elements.
   - Show build details in the visualization by default.
     Note: This tab also allows you to change the JobStream Visualization pane default settings.
     For information, see View JobStreams in the Visualization Pane.
4. Click OK.

Change the Size of the Information in the Visualization Pane

By default, IBM® Cognos® Data Manager shows information in the Visualization pane at 100%, but you can change this if you require.

Step
- From the View menu, click Zoom and then click the required percentage.
  Tip: You can also double-click the Visualization pane to enlarge the objects so that they fill the available space.

Reveal or Hide the Visualization Pane

By default, the Visualization pane shows information about the object selected in the Tree pane. However, you can hide this information if you require.

Step
- From the View menu, click Show Visualization.
Copy the Visualization Pane to the Clipboard

You can copy the Visualization Pane to the clipboard.

Step
- From the View menu, click Copy to Clipboard.

Set Up Window and Dialog Box Preferences

You can set up your preferences for the following:
- whether you want the Welcome dialog box to appear when you start IBM® Cognos® Data Manager
- whether you want Data Manager to remember the size and position of windows and dialog boxes when you have changed them

Steps
1. From the Tools menu, click Options.
2. Click the General tab.
3. Select or clear the following check boxes as required
   - Show the welcome screen when the application starts
   - Remember window sizes and positions
4. Click OK.

Locate Components and Their Dependencies

Sometimes, in a large catalog, it can be difficult to locate a component. IBM® Cognos® Data Manager can help you to
- locate a component
- locate the dependencies of a component

Components are connections, builds, reference dimensions, reference structures, JobStreams, templates, functions, metadata dimensions, and metadata collections.

Locate a Component

You can search for specific components in a catalog.

Steps
1. From the View menu, click Navigate.
2. Click the Find Components tab.
3. In the **Search for the text** box, type the name of the component to find.

   **Tip:** You can include the asterisk (*) and question mark (?) wildcard characters to assist you.

   **Note:** You cannot search for user-defined folders.

4. Click **Find Now**.

   IBM® Cognos® Data Manager lists all the components in the current catalog that match your search text, together with their component type and location.

5. To locate a component in the **Tree** pane and to update the **Visualization** pane, click the component name in the list.

**Tips**
- You can cancel a search at any time by clicking **Stop**.
- You can copy the search results to the Microsoft® Windows® clipboard by clicking **Copy to Clipboard**.

**View and Locate the Dependencies of a Component**

You can view the dependencies of a specific component.

**Steps**

1. In the **Tree** pane, click the component.

2. From the **View** menu, click **Navigate**.

   The **Navigator** dialog box appears showing all the dependencies for the selected component, together with their component type.

3. Click the **Navigate** tab.

   When you click a component in the navigator, it is located in the **Tree** pane, and the navigator is refreshed to show the dependencies of the currently selected component. The **Visualization** pane is also refreshed to show the selected component.

   - If a component has dependent user-defined functions, the functions are not shown.
   - You cannot view or locate user-defined folders.

   **Tip:** To move between previously selected components, click the left arrow button or the right arrow button as appropriate.

4. If you want to freeze the contents of the navigator at any point, so that only the **Tree** pane and **Visualization** pane are refreshed when you click a component in the navigator, select the **Freeze** check box.

   **Tip:** To copy the contents of the navigator to the Microsoft® Windows® clipboard, click **Copy to Clipboard**.
Create a Similar Component

If you want to create a component that is similar to an existing component, duplicate the component and then make the required changes. This is often quicker and simpler than creating a completely new component.

Steps
1. Click the component to duplicate.
2. From the Edit menu, click Duplicate.
   
   If the catalog has unsaved changes, a message appears prompting you either to save the changes and proceed or to cancel.
3. Click Save First, Then Duplicate.
   
   IBM® Cognos® Data Manager duplicates the component and assigns a unique name to it by adding a number to the original name. For example, the duplicate of FISCAL would be FISCAL:1.

Case Sensitivity

Names of objects in IBM® Cognos® Data Manager Designer are not case sensitive. For example, Data Manager considers VendorDrillThru, VENDORDRILLTHRU, and vendordrillthru to be identical.

However, column names used in Cognos® SQL queries are case sensitive and must exactly match the database column names. For example, a column named Productid is considered to be different from a column named PRODUCTID or productID.

Native SQL queries are subject to the case rules of the DBMS or external function library.

Cognos SQL

When constructing SQL statements in IBM® Cognos® Data Manager, you can specify whether you want to use native SQL or Cognos® SQL for the database you are accessing. By default, a Data Manager connection accepts any vendor-specific SQL SELECT statement in a data source, including non standard SQL extensions and hints.

Using Cognos® SQL provides a greater degree of portability between mixed database environments because a common dialect can be used. Cognos® SQL is an extension of SQL99.

Note: You cannot use Cognos® SQL for SQLTXT connections or Published FM Package connections.

Change Fonts

You can change the fonts that are used in IBM® Cognos® Data Manager. Normally, Data Manager uses a default set of fonts according to the locale settings of your computer. However, if you are
working with data that contains characters from different locales, you may want to change the font so that the characters are shown correctly.

**Steps**
1. From the Tools menu, click Options.
2. Click the Fonts tab.
3. Click the appropriate browse button for the setting that you want to change.
   - **Application Font** applies to the Tree pane, the tabs on the Visualization pane, and all window and dialog boxes
   - **Expression Font** applies to any expression that you enter, perhaps for a derivation element or user-defined function
   - **Visualization Font** applies to the Visualization pane
   The Select Font dialog box appears.
4. From the drop down lists, select the appropriate font.
   **Tip:** If you are changing expression fonts, you can restrict the list of fonts to fixed width fonts only by selecting the Fixed width only check box.
5. Click OK to close the Select Font dialog box.
   **Tip:** Click Reset to return to the default values.
6. Click OK.

**Quoted Identifiers for Multibyte Character Sets**

By default, identifiers are not enclosed in quotation marks in IBM® Cognos® Data Manager Designer. If, in the Data Manager scripting language, you want to use identifiers that contain characters that may be parsed as reserved words, then they must be enclosed in quotation marks.

You can specify that identifiers are to be automatically enclosed in quotation marks when they are dragged and dropped within the scripting windows.

**Steps**
1. From the Tools menu, click Options.
2. Click the General tab.
3. Select the Quote identifiers check box.
4. Click OK.
Chapter 3: Getting Started with IBM Cognos Data Manager Designer
Each IBM® Cognos® Data Manager catalog contains the components that define how Data Manager extracts, transforms, and delivers data.

This illustration shows the top-level objects within a Data Manager catalog. It also illustrates

- the flow of data between objects, from the data sources, and to the target data marts
- how Data Manager creates conformed models for a data mart that can be exported as metadata

A Data Manager catalog is stored in a database and uses Database Management System (DBMS) drivers to connect to the required databases. For more information, see "Database Drivers" (p. 515).

Every Data Manager catalog consists of a number of tables. Each schema within a database server can contain only one Data Manager catalog. Therefore, multiple catalogs must be assigned to distinct schemas. For more information, see "Create an IBM Cognos Data Manager Catalog Schema" (p. 46).

Although multiple instances of Data Manager can access a catalog concurrently, we do not recommend that multiple users perform updates concurrently. To detect concurrent updates, Data Manager applies data integrity checks.

For information about a collaborative environment with multiple developers working on the same project concurrently, see "Multi-Developer Support" (p. 347).
Create an IBM Cognos Data Manager Catalog

Only one IBM® Cognos® Data Manager catalog can be open at any time. If you attempt to create or open a catalog when another catalog is already open, a message appears indicating that the open catalog will be closed.

**Steps**

1. Create a data source where the Data Manager catalog is to reside. The method that you use is specific to the chosen DBMS.
   
   For information, see the documentation of the relevant software vendor. If you are creating a database using ODBC, see "Open the ODBC Data Source Administrator" (p. 71).

2. From the File menu, click New Catalog.
   
   Tip: You can also create a catalog from the Welcome dialog box.

   The New Catalog wizard appears.

3. Type a name for the new catalog and, if you require, a business name and description.

4. Click Next.

5. In the left pane, click the connection method to connect to the database that is to hold the catalog.
   
   Data Manager updates the dialog box to show fields that are appropriate for the connection method that you chose.

6. Enter the values required by the connection method.

7. Click Test Connection.
   
   If the connection is successful, a message appears indicating this.

8. Click Finish.
   
   If the database that you chose already contains a catalog, an error message appears.

   Data Manager automatically creates the tables that it requires in the database. However, you may prefer to create these manually. For more information, see "Create an IBM Cognos Data Manager Catalog Schema" (p. 46).

Open an IBM Cognos Data Manager Catalog

If you attempt to open a catalog that was created using IBM® Cognos® DecisionStream Series 7 or earlier, IBM® Cognos® Data Manager prompts you to upgrade the catalog or to create an empty catalog using the same connection. If you choose to upgrade the catalog, the Upgrade Catalog wizard appears. For more information see "Perform an Upgrade" (p. 49).
Data Manager allows only one catalog to be open at any time. If you attempt to open a catalog when another catalog is already open, a message appears indicating that the open catalog will be closed.

Steps
1. From the File menu, click Open Catalog.
2. Click the Existing tab.
   
   **Tip:** Click the Recent tab to select one of the last 20 catalogs that you opened. You can clear the recent catalogs list by clicking Options from the Tools menu, and then clicking Clear recent catalogs list.
3. In the left pane, click the connection method to connect to the database that contains the catalog.
   
   Data Manager updates the dialog box to show fields that are appropriate for the connection method that you chose.
4. Enter the values required by the connection method.
5. Click OK.
   
   If the target database does not contain catalog tables, a message appears giving you the option to create them.

Change the Properties of a Catalog
You can change the properties of a catalog.

Steps
1. In the Tree pane, click the catalog.
2. From the Edit menu, click Properties.
3. Change the properties as you require.
4. Click OK.

Document an IBM Cognos Data Manager Catalog
IBM® Cognos® Data Manager can document the contents of a catalog in HTML format and open it in your web browser. You can then view and print it.

Steps
1. From the File menu, click Document Catalog.
2. In the Document name box, type the full path and file name of the file that you want to create.
You must give the file name an extension so that Data Manager can start a Web browser in which you can view the resulting documentation. If you type a file extension other than .htm or .html, you must register it within your Web browser.

3. In the **Document type** box, click the type of document that you want to create:
   - Full Documentation provides full component details with SQL statements and also includes information about all configured delivery modules.
   - Usage Summary provides a list of all the catalog components.

4. In the **Master document** box, you may want to enter the name of a master document to use as a template for the generated documentation.

   **Note:** The master document must have HTML tags within it and an `<INSERT>` tag to mark the point of insertion for the generated document. For example, a simple master document would contain the text "<HTML><INSERT></HTML>".

5. Click **OK**.

   Your Web browser appears showing the generated documentation.

   **Note:** Facts builds are referred to as data builds in the generated document.

### Create an IBM Cognos Data Manager Catalog Schema

When you create a catalog, IBM® Cognos® Data Manager automatically sets up the tables that it requires. However, for some installations, you may prefer to manually set up these tables. For example, you may want to assign database-specific syntax, such as the containers in which tables are placed within the database.

A Data Manager catalog consists of three sets of tables:

- Run detail tables named dsb_component_run, dsb_run_context, dsb_jobnode_run, dsb_sequence, and dsb_delivery_hist.
  
  For information on these tables, see "Audit Tables" (p. 531).

- Audit message tables named dsb_audit_trail, dsb_audit_msg, and dsb_audit_msg_line.
  
  For information on these tables, see "Audit Tables" (p. 531).

- Catalog tables named dsb_catalog, dsb_component, and dsb_component_line. These tables are used internally by Data Manager.

### Tips

- Data Manager allows you to remove a previous version of a catalog. To do this, from the **Schema** box, click **DecisionStream Series 7 Catalog (V4)**, and then click **Drop**.
To remove run details or audit history from a catalog, from the Schema box, select Data Manager Catalog (V5) Run Details Only or Data Manager Catalog (V5) Audit Messages Only, and then click Drop.

**Steps**

1. From the Tools menu, click Manage Database Schema.
2. In the Schema box, click Data Manager Catalog (V5).
3. In the Database box, click the connection to which Data Manager should apply the schema.
   **Note:** You can only select connections defined in the current catalog.
4. Clear the Auto apply check box for Data Manager to actually create the schema.
5. Use the buttons at the bottom of the window to add the SQL statements for the function that you want to perform. You can view and edit the statements on the SQL tab.

<table>
<thead>
<tr>
<th>Button</th>
<th>SQL statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Creates the catalog tables. You can copy these statements, using the Microsoft® Windows® clipboard, to other applications to inform your database administrators of the tables to create.</td>
</tr>
<tr>
<td>Index</td>
<td>Creates indexes for the selected database tables.</td>
</tr>
<tr>
<td>Grant</td>
<td>Grants all permissions to all users for all tables in the schema. However, this may not apply to all databases.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the contents of all the catalog tables.</td>
</tr>
<tr>
<td>Drop</td>
<td>Drops all catalog tables removing the catalog from the database.</td>
</tr>
</tbody>
</table>

This populates the SQL tab with the correct statement and, if you selected the Auto apply check box, applies the statement.

6. Click Save if you want to save the SQL script, perhaps to use when manually creating the tables. The Save SQL dialog box appears where you can specify the path and name of the file to create.
7. Click the Apply button to apply the SQL statement, as it appears on the SQL tab, to the connection and to create the tables.
   The results of applying the SQL statement are shown on the Log tab.
8. Click Close.
IBM® Cognos® Data Manager allows only one catalog to be open at any time. If you attempt to open or create a catalog when another catalog is already open, a message appears indicating that the open catalog will be closed.

**Step**
- From the **File** menu, click **Close Catalog**.
IBM® Cognos® Data Manager introduces new features. Because of this, catalogs created using IBM® Cognos® DecisionStream Series 7 must be upgraded before you can use them with Data Manager.

Note: If your catalogs were created using DecisionStream 6.5 or earlier, you must upgrade them to DecisionStream Series 7 first.

When you upgrade a catalog, you have the option to retain the Series 7 catalog tables so that you can continue to use the catalog in DecisionStream Series 7. This is possible because Data Manager uses different catalog tables to those used in DecisionStream Series 7.

Data Manager and DecisionStream Series 7 or earlier can exist concurrently on the same computer.

Upgrading a catalog automatically

- creates catalog tables
- updates the script for each component
- creates a series of new components

In addition to upgrading a catalog, you can also

- restore a catalog backup created in DecisionStream Series 7 into a Data Manager catalog
- import a package created in DecisionStream Series 7 into Data Manager

Notes

- CATIMP files containing DecisionStream Series 7 components are upgraded and converted on import.
- External user-defined functions created in DecisionStream Series 7 will continue to work in Data Manager.
- When you have upgraded a catalog, any subsequent changes to the Series 7 catalog can be brought into Data Manager by importing a component package. For information, see "Import a Component Package" (p. 355).
- For information about performing an upgrade from the command line, see "catupgrade" (p. 432).

Perform an Upgrade

Before you upgrade a catalog to IBM® Cognos® Data Manager, ensure that

- all the computers on which the catalog may be used have Data Manager installed on them
- if the Data Manager engine is running on a server with UNIX® or another Microsoft® Windows® operating system, these are upgraded at the same time as the computers running Data Manager Designer
you list all existing CATEXP or manually created Data Manager script files, and all SQLTXT definition files

you review all command scripts that may presume that only one version of Data Manager is installed or that reference the old location

you verify which scripts may need revising that are run through scheduled services such as cron/at

you have created a backup (either a database backup or a Data Manager backup) of the catalog

For information, see "Backup and Restore Catalogs" (p. 359).

Additional checks are required for catalogs stored as source code control projects because Data Manager upgrades your personal copy of each catalog, not the master catalogs. Ensure that you have

checked in each catalog to which changes have been made

synchronized each catalog to ensure that they reflect the master catalogs in the source code control repository

Steps

1. From the File menu, click Upgrade Catalog.

   If a catalog is open, a message appears indicating that this action will disconnect the current catalog.

2. In the left pane of the Upgrade Catalog wizard, click the connection method to connect to the database that contains the catalog to upgrade.

   Data Manager updates the dialog box to show fields that are appropriate for the connection method that you chose.

3. Enter the values required by the connection method, and then click Next.

4. If the selected catalog is stored as a source code control project, a message appears asking whether you want to check in any catalog components before you continue.

   Note: You must use IBM® Cognos® DecisionStream Series 7 to check in the catalog. You may want to click Cancel at this stage, and begin the upgrade again when you have checked in the catalog.

   When you are ready to continue, select the Continue the upgrade check box, and then click Next.

5. If you want Data Manager to migrate the catalog history by copying all the audit trails currently held in the Series 7 catalog tables into the upgraded catalog, select the Migrate catalog history check box.

6. If you want Data Manager to retain the DecisionStream Series 7 catalog tables after the upgrade, ensure that the Delete prior version check box is clear.
If you delete the earlier version of the catalog, Data Manager deletes the DecisionStream Series 7 catalog tables and creates Data Manager catalog tables. You cannot access the catalog in DecisionStream Series 7 after upgrading the catalog.

If you do not delete the earlier version, Data Manager retains the DecisionStream Series 7 catalog tables, as well as creating Data Manager catalog tables. You can continue to access the catalog in DecisionStream Series 7 after upgrading the catalog.

7. If you want to keep a backup copy of the catalog, select the **Backup prior version** check box, and then type a new name for the existing catalog in the **Backup file name** box.

8. Click **Finish**.

**Upgrade Failure**

In some situations, an IBM® Cognos® Data Manager upgrade may fail leaving the Data Manager project in an inconsistent state. This may happen for a number of reasons, including lack of disk space or if the database system that contains the catalog cannot apply the required changes. It is therefore very important that you create a backup of the previous environment before you begin to upgrade.

If an upgrade fails, you can restore the previous catalog database environment by using the restore functionality for your database.

**Note:** If you retained the IBM® Cognos® DecisionStream Series 7 catalog tables when you performed the initial upgrade, you can run the upgrade process again without following the steps below. Data Manager uses the DecisionStream Series 7 catalog tables to perform the upgrade, and overwrites any existing Data Manager catalog tables.

**Steps**

1. Using your database's utilities, or the Data Manager **Database Schema** window, drop the Data Manager catalog tables.

   For information on using the **Database Schema** window, see "Create an IBM Cognos Data Manager Catalog Schema" (p. 46).

   These catalog tables are named `dsb_audit_msg`, `dsb_audit_msg_line`, `dsb_audit_trail`, `dsb_catalog`, `dsb_component`, `dsb_component_line`, `dsb_component_run`, `dsb_delivery_hist`, `dsb_jobnode_run`, `dsb_run_context`, and `dsb_sequence`.

2. Use DecisionStream Series 7 to recreate the old database tables.

   These catalog tables are named `ds_audit_msg`, `ds_audit_msg_line`, `ds_audit_trail`, `ds_catalog`, `ds_component`, `ds_component_line`, `ds_component_run`, `ds_delivery_hist`, `ds_jobnode_run`, `ds_run_context`, and `ds_sequence`.

   You can do this using the **Database Schema** window. For information, see "Create an IBM Cognos Data Manager Catalog Schema" (p. 46).

3. From within DecisionStream Series 7, restore the catalog.

4. When the problem is resolved, perform the upgrade again.
Chapter 5: Upgrading a Catalog

**Restore From an IBM Cognos DecisionStream Series 7 Backup**

If you create a catalog in IBM® Cognos® Data Manager, or if you have followed the upgrade process, you can restore a catalog backup from IBM® Cognos® DecisionStream Series 7. In this case, Data Manager updates the script for the catalog objects and creates components as necessary.

**Import an IBM Cognos DecisionStream Series 7 Package**

If you create a catalog in IBM® Cognos® Data Manager, or if you have followed the upgrade process, you can import a package created in IBM® Cognos® DecisionStream Series 7. In this case, Data Manager updates the script for the catalog objects and creates components as necessary.

For information, see "Component Packages" (p. 354).

**Post Upgrade Considerations**

After upgrading a catalog, you must consider the impact that the upgrade may have on existing components and operations.

**Command Lines in Scripts**

The default location of IBM® Cognos® Data Manager executables is \Program Files\ibm\cognos\c10. A script that is used to call Data Manager builds and JobStreams should be updated to the Data Manager location.

For more information, see "Commands" (p. 413), and the IBM® Cognos® Function and Scripting Reference Guide.

**IBM Cognos Data Manager SQLTXT Designer Definition Files**

When a SQLTXT definition file (.def) is amended and saved in IBM® Cognos® Data Manager, it cannot be opened in IBM® Cognos® DecisionStream Series 7. We recommend that you create a backup copy of all SQLTXT definition files before you use them in Data Manager.

**Source Code Control Projects**

When you upgrade an IBM® Cognos® DecisionStream Series 7 catalog that is stored as a source code control project, it is no longer attached to source code control in IBM® Cognos® Data Manager. It exists as a stand-alone catalog. To share the catalog with other developers, you must add it to source code control again.

*Note:* To ensure that you retain the full project history, you must add the catalog to a new project in your source code control system.

For more information, see "Add a Catalog to Source Code Control" (p. 348).
**Metadata Deliveries**

IBM® Cognos® Data Manager no longer delivers metadata using metadata deliveries in fact builds. Any metadata deliveries to IBM® Cognos® Series 7 Transformer, IBM® Cognos® Series 7 Architect, and Microsoft® Analysis Server that you previously created in an IBM® Cognos® DecisionStream Series 7 catalog are no longer supported. All metadata is now exported directly to IBM® Cognos® Framework Manager using the metadata export utility in Data Manager.

**DB-LIB Connections and Microsoft SQL Server Bulk Loader Connections**

Any DB-LIB connections that you previously created in an IBM® Cognos® DecisionStream Series 7 catalog are migrated to OLE-DB connections when you upgrade to IBM® Cognos® Data Manager.

In addition, if your DecisionStream Series 7 catalog contains either of the following fact deliveries with a DB-LIB connection, these connections are also migrated:

- Bulk Copy (BCP) deliveries to Microsoft® SQL Server
- Microsoft® SQL Server BCP deliveries

You should test all migrated connections in Data Manager to ensure that they function as you expect.

**IBM Cognos Data Manager Network Services**

When you upgrade to IBM® Cognos® Data Manager, you must take into consideration changes that are made to the Data Manager Network Services server and client components.

**IBM Cognos Data Manager Network Services Server Component**

In IBM® Cognos® Data Manager, the Data Manager Network Services server component has been modified so it is more in line with the rest of the IBM® Cognos® Business Intelligence infrastructure:

- There is now a common IBM® Cognos® BI service that replaces the SOAP server used in IBM® Cognos® DecisionStream Series 7.
- The Socket server now has a different name, Data Manager Network Services Socket Server, and is located on a different port to the one used in DecisionStream Series 7.

As a result, when you upgrade to Data Manager, the Data Manager Network Services server component can exist concurrently with the DecisionStream Series 7 server component.

**IBM Cognos Data Manager Network Services Client Component**

In IBM® Cognos® Data Manager, the file dsnethosts.txt no longer exists. Instead the information contained in that file is maintained using IBM® Cognos® Configuration.

As a result, any changes that you previously made to the dsnethosts.txt file in the IBM® Cognos® DecisionStream Series 7 client component are not migrated when you upgrade to Data Manager.

You must manually incorporate these changes into IBM® Cognos® Configuration as follows:

- Use a text editor to open the dsnethosts.txt file, and review the changes you have made.
Open IBM® Cognos® Configuration and update the configuration as required.

For more information on configuring Data Manager Network Services, see the IBM® Cognos® Data Manager Installation and Configuration Guide.
Chapter 6: SQLTerm and SQL Helper

Use SQLTerm and SQL Helper to

- build SQL statements to access source data
- obtain information about database objects, DBMS drivers, and data sources
- examine the contents of database tables
- test SQL extracts from source databases
- tune SQL extracts

SQLTerm and SQL Helper also support

- Data Definition Language (DDL), for example, CREATE TABLE
- Data Manipulation Language (DML), for example, INSERT INTO
- queries about the connection, for example, SOURCES

Note: Whether IBM® Cognos® Data Manager shows SQLTerm or SQL Helper depends on what you are doing when you require assistance with constructing an SQL statement. You use SQLTerm and SQL Helper in exactly the same way.

For information about using SQLTerm and SQL Helper with Published FM Packages, see "Add Published FM Package Data Source" (p. 145). For information about using SQLTerm on the command line, see "sqlterm" (p. 439).

Create SQL Statements

Using SQLTerm or SQL Helper, you can

- add a SELECT statement for all columns in the selected table or for a single column
- add a table name or column name to a statement
- add a DROP TABLE, DELETE FROM, SELECT FROM, or DESCRIBE statement for the selected table

Add a SELECT Statement to an SQL Query

In addition to using the method described here, you can also type an SQL statement directly into the box in SQLTerm or SQL Helper.

You can use only a single SELECT statement in a query. Multiple SELECT statements result in an error.

Steps

1. From the Tools menu, click SQLTerm.
2. In the box at the top of the SQLTerm window, click the database with which you want to work.

3. Use the Cognos SQL check box to specify whether you want to use the native SQL or Cognos® SQL for the database you are accessing when you construct the SQL statement. Select or clear the check box as appropriate.

Tip: Using Cognos® SQL makes it easier to port your project between database platforms.

Note: The default for the Cognos SQL check box is determined by whether you selected the Cognos SQL check box in the Connection Properties dialog box.

For more information, see "Cognos SQL" (p. 40).

4. In the Database Objects pane, expand the tree, and then click the table or column for which you want to add a SELECT statement.

Tip: To refresh the list of tables in the Database Objects pane, double-click the database object, or right-click and then click either Refresh Database or Refresh Table.

5. To generate the SQL statement use one of the following methods:

- Right-click, and then click either Add Table Select Statement, or Add Column Select Statement.

- Hold down Ctrl and drag the table or column you selected in the Database Objects pane to the Query pane.

IBM® Cognos® Data Manager clears any existing query that may be in the Query pane and generates the SQL SELECT statement for the selected table or column.

Tip: To clear the query, click the Clear the query button.

6. You are now ready to execute the SQL statement.

For more information, see "Test a Data Source" (p. 149).

Example

This example shows the result of adding a SELECT statement for the table named D_BIMart_Staff and executing the generated statement by clicking the Return all rows button.
Data Manager generates a SELECT statement for all the columns in the PRODUCT_TYPE table. At the bottom of the window, the results pane shows the number of rows and the data returned by the SQL statement.

Add a Table Name or Column Name to an SQL Query

Using SQLTerm, you can easily add table names and column names to an SQL query. You can enter the name alone or you can precede it with a comma.

Steps

1. In the box at the top of the SQLTerm window, click the database with which you want to work.

2. Use the Cognos SQL check box to specify whether you want to use the native SQL or Cognos® SQL for the database you are accessing when you construct the SQL statement. Select or clear the check box as appropriate.

   Tip: Using Cognos® SQL makes it easier to port your project between database platforms.

   Note: The default for the Cognos SQL check box is determined by whether you selected the Cognos SQL check box in the Connection Properties dialog box.

   For more information, see "Cognos SQL" (p. 40).

3. In the Database Object pane, click the table name or column name and then, either:
   - Right-click, and then click either Add Table or Add Column, or drag the selected table name or column name to the Query pane.
   - Right-click, and then click either Add Comma and Table or Add Comma and Column, or hold down Shift and drag the selected table name or column name to the Query pane.

   Tip: To clear the query, click the Clear the query button.

4. You are now ready to execute the SQL statement.
For more information, see "Test a Data Source" (p. 149).

Add Other Statements to an SQL Query

You can add statements to drop a table (DROP TABLE), delete rows (DELETE FROM), select rows (SELECT FROM), or describe a table (DESCRIBE).

Steps
1. In the box at the top of the SQLTerm window, click the connection with which you want to work.
2. In the Database Objects pane, right-click the table for which you want to add the statement, and then click the action to perform.
IBM® Cognos® Data Manager adds the selected statement.
3. You are now ready to execute the SQL statement.
For more information, see "Test a Data Source" (p. 149).

Additional SQLTerm Commands

In addition to using SQLTerm to connect to and query SQL databases, you can type commands in the SQL Query pane to obtain additional information. You can type commands to
- obtain information about database objects, DBMS drivers, and data sources
- execute SQL scripts
- obtain help on SQLTerm commands

Steps to List the Available Commands
1. In the Query pane, type Help.
2. Click the Return all rows button.
These are the available commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE</td>
<td>Lists all the tables within the current database.</td>
</tr>
<tr>
<td>DESCRIBE &lt;table_name&gt;</td>
<td>Describes the columns within the specified table. Where possible, these are the column name, its data type and length, whether the column can accept null values, and whether the column forms part of the primary key for the table.</td>
</tr>
<tr>
<td>OBJECTS &lt;TVS*&gt;&lt;schema&gt;</td>
<td>Lists all the objects within the current table views schema (TVS) and database.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DIFF &lt;tab1&gt;&lt;tab2&gt;</td>
<td>Compares two tables and lists the columns that contain differences.</td>
</tr>
<tr>
<td>DRIVERS [ALL]</td>
<td>Lists the DBMS drivers for which IBM® Cognos® Data Manager is configured. Use the ALL argument to obtain a list of all the drivers known to Data Manager.</td>
</tr>
<tr>
<td>FETCHSIZE &lt;N&gt;</td>
<td>Sets or shows the fetch row count used to query the database. This setting controls the number of rows that are queried at once by using the bulk fetch facility for your database. You can use this to tune large extractions from a source database. This setting has no effect for databases that do not support bulk fetch operations. The default fetch row size is 50.</td>
</tr>
<tr>
<td>ISOLEVEL &lt;N&gt;</td>
<td>Sets or shows the database isolation level defined for the database transaction you are using to query the data source. This setting can be used where you specifically want to control the transaction isolation level for a query transaction. This setting has no effect for database systems that do not support transaction isolation levels. The default value is determined by the source database system.</td>
</tr>
<tr>
<td>OUTPUT [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>PROGRESS &lt;N&gt;</td>
<td>Shows the select progress for every n rows.</td>
</tr>
<tr>
<td>PROMPT &lt;text&gt;</td>
<td>Shows a line of text, which is useful for debugging and inserting custom messages in SQLTerm or SQL Helper output files.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>Lists the schema from the current connection.</td>
</tr>
<tr>
<td>SCREEN [&lt;width&gt;]</td>
<td>Sets the maximum number of characters on a line before the output is wrapped. The default is to return the current width.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SOURCES [&lt;driver&gt;] [ALL]</td>
<td>Lists all the available data sources. The default is to return a list of all data sources that are available by using the driver that is currently in use. If you specify the optional ALL argument, SOURCES returns all data sources that are available through any DBMS driver. You can also list the data sources that are available through any particular driver.</td>
</tr>
<tr>
<td>STATEMENT [SELECT</td>
<td>DML</td>
</tr>
<tr>
<td>STATISTICS [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>TIME</td>
<td>Shows the current system date and time.</td>
</tr>
<tr>
<td>TYPES</td>
<td>This command shows information about the current connection. For ODBC, it also shows information about the data type conversion.</td>
</tr>
<tr>
<td>Help</td>
<td>Shows all the commands included in this table and a brief description of each.</td>
</tr>
</tbody>
</table>

**Test an SQL Query**

You can choose whether you want to return all the selected rows in the database or just a single row. You can also execute the SQL query and return all the rows.

**Step to Return All the Selected Rows in the Database**

- From the toolbar in the SQLTerm window, click the Return all rows button.

**Step to Limit Results to a Single Row**

- From the toolbar in the SQLTerm window, click the Return one row button.

**Step to Execute the Statement and Return the Rows**

- In the Test pane, click the Execute statement(s) button.

**Step to Interrupt Execution of an SQL Query**

- From the toolbar in the SQLTerm window, click the Interrupt current processing button.
Include Substitution Variables in an SQL Query

You can execute SQL queries that include substitution variables so that you can run an SQL query and view the results as the query is constructed. Run-time variables can be used to control queries, especially filters.

Note: If you include substitution variables in an SQL query for a reference structure, you must declare the variables in the appropriate fact build or dimension build.

For more information about substitution variables, see "Variables" (p. 319).

When you execute an SQL query that includes a variable, a dialog box appears for you to enter the value of each variable used in the query. IBM® Cognos® Data Manager uses these values when the query is executed.

Steps

1. In the SQLTerm or SQL Helper Query pane, construct the required SQL query that includes a substitution variable.

2. Execute the SQL query.

   For more information, see "Test a Data Source" (p. 149).

   The Values for Expression window appears listing each substitution variable included in the query.

3. In the Value column, type a suitable value for each variable.

   Tip: Click Set Default Values to enter the default value that you specified when you defined the substitution variable.

   For more information about default values, variable data type and scope, see "Testing Expressions and Scripts" (p. 341).

4. Click OK.

   The window shows the SQL statements that are generated after variable substitution has occurred and after processing any Cognos® SQL syntax or functions that are not supported by the underlying database.

View the Executed Query

If required, you can view the executed SQL query.

Step

- Click the Show executed query button.

  Tip: Select the Word wrap check box to specify that word wrapping should be used.
Chapter 7: Connections

To configure IBM Cognos® Data Manager to interact with data sources, you must establish connections to:

- identify the database
- specify the connection method that Data Manager must use to connect to the data
- provide information that the database management system (DBMS) requires when Data Manager connects to the data; for example, the username and password

Each connection definition is stored in a Data Manager catalog. This means that you define a connection once, and then use it in as many components as you require.

Having created an appropriate connection to interact with the data sources, you treat all data sources in an identical manner.

You can create a connection and change the physical database to which it relates. For more information, see "Cross Platform Deployment" (p. 559).

Connections are shown in the Library tree as follows.

Create a Database Connection

When you create a database connection, you must supply the connection parameters that the database requires. Each database type has its own particular requirements that typically include information to identify the database and a valid username and password.

You must also specify whether you want to use Cognos® SQL, which is an extension of SQL 99. Using Cognos® SQL, you have a greater degree of portability between mixed database environments because a common dialect can be used. You cannot use Cognos® SQL for SQLTXT connections.

By default, a connection accepts any vendor-specific SQL SELECT statement in a data source, including non standard SQL extensions and hints.

For information on creating an IBM® Cognos® data source connection, see "Create an IBM Cognos Data Source Connection" (p. 66).

For information on creating a Published FM Package connection, see "Create a Published FM Package Connection" (p. 67).

Steps

1. In the Library folder, click Connections.
2. From the **Insert** menu, click **Library**, and then click **Connection**.
   The **Connection Properties** dialog box appears.

3. Click the **General** tab.

4. In the **Alias** box, type an alias for the connection.
   IBM® Cognos® Data Manager uses this alias whenever it refers to the connection.

5. In the **Description** box, if required, type notes about the connection.

6. Click the **Connection Details** tab.
   
   **Note:** You can set up a connection to which you do not have access from your computer. This allows the connection to be used on another computer on which the connection method if available.

7. Click the appropriate connection method for the database to which you want to connect.
   Data Manager updates the dialog box to show fields that are appropriate for the connection method that you chose.

8. Enter the values required by the connection method.

9. In the **Collation sequence** box, type the collation sequence for the selected database.
   For more information, see "Distributed Sorting" (p. 64).

10. Select the **Cognos SQL** check box to use Cognos® SQL when you construct an SQL statement for components using this connection. If you clear this check box, you must use native SQL for the database you are accessing.
   The selection that you make here determines the default for the **Cognos SQL** check box in other components that use this connection.

11. To test the connection, click **Test Connection**.
   A message appears stating whether the test was successful. Click **OK** to close the message box.

12. Click **OK**.

---

**Distributed Sorting**

Both IBM® Cognos® Data Manager and source databases can sort data. Problems can arise if the sort order of Data Manager and the source databases differ. This often occurs if Data Manager and the source databases are operating in different locales.

To help avoid problems with compatible sort orders, you can instruct Data Manager that sorting should be performed by the Data Manager computer rather than the underlying data source.

**Note:** This only affects data sources that use Cognos® SQL.

You specify how distributed sorting is to be performed in the Collation sequence box in the Connection Properties dialog box. You can enter

- **COMPATIBLE** (default)
- NOT-COMPATIBLE
- a database-specific setting

COMPATIBLE

If the database to which you are connecting sorts data in the same order as the computer on which IBM® Cognos® Data Manager is running, enter COMPATIBLE. This causes sorting to be distributed between the computer on which Data Manager is running and the database to which you are connecting. This is the default.

NOT-COMPATIBLE

If the database to which you are connecting sorts data in a different order to the computer on which IBM® Cognos® Data Manager is running, enter NOT-COMPATIBLE. This causes all sorting to take place on the computer on which Data Manager is running.

Entering NOT-COMPATIBLE decreases the performance of queries that use distributed sorting. This difference in performance is dependent on the relative speed and power of the computer running your database versus the speed and power of the computer running Data Manager. Therefore, you should avoid unnecessary use of the NOT-COMPATIBLE setting.

Database-Specific Setting

A database-specific setting can only be used if you are accessing single-byte data.

The following lists show some example collation sequences.

EBCDIC

<table>
<thead>
<tr>
<th>Code page</th>
<th>Collation sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00037</td>
<td>EBCDIC00037</td>
<td>United States and Canada code page</td>
</tr>
<tr>
<td>00500</td>
<td>EBCDIC00500</td>
<td>International #5 code page</td>
</tr>
</tbody>
</table>

Microsoft® Access

<table>
<thead>
<tr>
<th>Collation sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESENG</td>
<td>English</td>
</tr>
<tr>
<td>ACCESSSSP</td>
<td>Spanish</td>
</tr>
<tr>
<td>ACCESSNOR</td>
<td>Nordic</td>
</tr>
<tr>
<td>ACCESSDUT</td>
<td>Dutch</td>
</tr>
</tbody>
</table>
Microsoft® SQL Server

<table>
<thead>
<tr>
<th>Code page</th>
<th>Collation sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SYBISODOCI</td>
<td>Dictionary order and not case-sensitive</td>
</tr>
<tr>
<td>850</td>
<td>SYBCP850DOC1</td>
<td>Not case-sensitive</td>
</tr>
</tbody>
</table>

IBM® DB2

<table>
<thead>
<tr>
<th>Collation sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2ENCASE</td>
<td>English and case-sensitive</td>
</tr>
</tbody>
</table>

Create an IBM Cognos Data Source Connection

An IBM® Cognos® Data Source connection allows you to access data sources set up in IBM® Cognos® Business Intelligence.

**Note:** IBM® Cognos® Data Manager can only access data from data sources that accept Cognos® SQL.

Before you can create an IBM® Cognos® Data Source connection, you must

- configure Data Manager to access the Gateway URI and the Dispatcher URI
  
  For information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

- be logged on to IBM® Cognos® Connection, the portal to IBM® Cognos® BI, which provides a single access point to all corporate data available in IBM® Cognos® BI.

  **Note:** If the server to which you are connecting accepts anonymous users, you do not have to log on. For information, see "Manage Credentials" (p. 71) and the IBM® Cognos® Connection User Guide.

**Steps**

1. In the Library folder, click Connections.

2. From the Insert menu, click Library, and then click Connection.

   The Connection Properties dialog box appears.

3. Click the General tab.

4. In the Alias box, type an alias for the connection.

   Data Manager uses this alias whenever it refers to the connection.

5. In the Description box, if required, type notes about the connection.

6. Click the Connection Details tab.
Note: You can set up a connection to which you do not have access from your computer. This allows the connection to be used on another computer on which the connection method is available.

7. Click Cognos Data Source.

8. Click the browse button.

The Select the Data Source dialog box appears listing all the available IBM® Cognos® data sources.

9. Click the data source that you want to use, and then click OK.

The name of the selected data source is shown in the Data Source Name box.

10. If required, in the Connection name box, specify the connection name defined for the data source in IBM® Cognos® BI.

11. If required, in the Signon name box, specify the signon name used to connect to the data source in IBM® Cognos® BI.

12. If required, select the Execute connection/session commands box to run the database commands associated with data source connections in IBM® Cognos® Connection.

For more information on using database commands in IBM® Cognos® Connection, see the IBM® Cognos® Administration and Security Guide.

13. Select the Cognos SQL check box to use Cognos® SQL when you construct an SQL statement for components using this connection. If you clear this check box, you must use native SQL for the database you are accessing.

The selection that you make here determines the default for the Cognos SQL check box in other components that use this connection.

14. To test the connection, click Test Connection.

A message appears stating whether the test was successful. Click OK to close the message box.

15. Click OK.

Create a Published FM Package Connection

A Published FM Package connection allows you to access query subjects from a published IBM® Cognos® Framework Manager package. Each Published FM Package connection can access one published package.

Note: You cannot use Cognos® SQL for Published FM Package connections.

Before you can create a Published FM Package connection, you must

- configure IBM® Cognos® Data Manager to access the Gateway URI and the Dispatcher URI

For information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.
be logged on to IBM® Cognos® Connection, the portal to IBM® Cognos® BI, which provides a single access point to all corporate data available in IBM® Cognos® BI.

Note: If the server to which you are connecting accepts anonymous users, you do not have to log on.

For information, see "Manage Credentials" (p. 71) and the IBM® Cognos® Connection User Guide.

Note: When using IBM® Cognos® Connection with multiple data source connections, you can use the DM_DATASOURCE_CONNECTION_SIGNONS variable to specify which data sources to use. For information, see "DM_DATASOURCE_CONNECTION_SIGNONS" (p. 324).

Steps
1. In the Library folder, click Connections.
2. From the Insert menu, click Library, and then click Connection.
   The Connection Properties dialog box appears.
3. Click the General tab.
4. In the Alias box, type an alias for the connection.
   Data Manager uses this alias whenever it refers to the connection.
5. In the Description box, if required, type notes about the connection.
6. Click the Connection Details tab.
   Note: You can set up a connection to which you do not have access from your computer. This allows the connection to be used on another computer on which the connection method if available.
7. Click Published FM Package.
8. Click the browse button.
   The Select the Package dialog box appears listing all the available published packages.
9. Click the package that you want to use, and then click OK.
   The name of the selected package is shown in the Package box.
10. To test the connection, click Test Connection.
    A message appears stating whether the test was successful. Click OK to close the message box.
11. Click OK.

Use an Alias to Connect to a Database

IBM® Cognos® Data Manager uses connection aliases to describe the connections to databases. For example, a connection named Sales may use an Oracle connection of username/password@service1 allowing specified connections to be re-used throughout the design process.
Although a connection string may be the preferred definition during the design phase of a Data Manager project, the test or production environment may need to connect to a different physical database. The database alias definition file allows multiple run environments to be defined to determine to which database Data Manager connects when you specify a particular alias. At runtime, the database alias definition file replaces the connection definitions defined as part of the Data Manager catalog. For example, to use a database alias definition file in a Data Manager database, you would type

databuild -ePRODUCTION -Ac:\workfiles\Alias.txt "ODBC" "DSN=DS_Tutorial"
CopyTable

Create a Database Alias Definition File

Database alias definition files are text files that specify to which databases IBM® Cognos® Data Manager can connect. A database alias definition file also specifies the parameters that Data Manager must use when connecting to the data. Each database alias definition file must define all the connections that a build requires.

Syntax

<runenv> <name> <driver> <driver_items> [<notes>]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;runenv&gt;</td>
<td>The run environment context for the database alias. By default, Data Manager uses &lt;DEFAULT&gt; when processing your projects. Additional run environments are created for specific purposes.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>The alias for the connection.</td>
</tr>
<tr>
<td>&lt;driver&gt;</td>
<td>The name of the database driver that Data Manager should use to connect to the data. For example, ORACLE or ODBC.</td>
</tr>
<tr>
<td>&lt;driver_name&gt;</td>
<td>The information that Data Manager should pass to the database driver to establish the connection. For example, user ID and password.</td>
</tr>
<tr>
<td>&lt;notes&gt;</td>
<td>Optional notes to describe the alias. You can only view such notes within the alias definition file.</td>
</tr>
</tbody>
</table>

Examples

The following is an example database alias definition file.

//******************
//* DEFAULT ALIASES *
//******************

<DEFAULT> 'Tutorial Catalog' ODBC 'DSN=DS Tutorial Catalog' ''
<DEFAULT> 'Tutorial DataMart' ODBC 'DSN=DS Tutorial Output' ''
<DEFAULT> 'Tutorial Reference' ODBC 'DSN=DS Tutorial Reference' ''
<DEFAULT> 'Tutorial Sales' ODBC 'DSN=DS Tutorial Sales' ''
Password Encryption

IBM® Cognos® Data Manager supports both encrypted and plain-text passwords for database aliases. For encrypted passwords, Data Manager uses its own encryption algorithm. It determines whether a password is encrypted and automatically deciphers encrypted passwords before connecting to the target database. To use encrypted passwords, you must create database connections within Data Manager Designer and then, export those connections to an alias definition file. You can export the connections using the CATEXP command, or Show Source from the Actions menu. For more information, see "catexp" (p. 425).

Change the Properties of a Connection

You can change the properties of an existing connection.

Steps
1. Click the connection for which you want to change the properties.
2. From the Edit menu, click Properties.
3. Change the properties as you require.
   
   Note: Changing the setting in the Cognos SQL check box does not change the Cognos® SQL setting in any existing data sources.
4. Click OK.

Test a Connection

You can test a connection to ensure it is set up correctly.

Steps
1. Click the connection you want to test.
2. From the Edit menu, click Properties.
   
   The Connection Properties dialog box appears.
3. Click the Connection Details tab.
4. Click Test Connection.
   
   If the test is successful, a message appears indicating this. If the test is unsuccessful, the native database driver error messages appear.
5. Click OK to acknowledge the message.
Delete a Connection

You cannot delete a connection that another IBM® Cognos® Data Manager component is using.

Tip: You can determine the components that use a connection by using the Data Manager navigator. For more information, see “Locate Components and Their Dependencies” (p. 38).

Steps
1. Click the connection you want to delete.
2. From the Edit menu, click Delete.
   A message appears prompting you to confirm that you want to delete the selected connection.
3. Click Yes.
   If another Data Manager component is using the connection, an error message appears showing all the components that use the connection. Click OK to acknowledge this message.

Open the ODBC Data Source Administrator

You may want to use the ODBC Data Source Administrator when you create a connection. You can do this from the Microsoft® Windows® operating system Control Panel or directly from IBM® Cognos® Data Manager Designer.

Step
• From the Tools menu, click ODBC Administrator.

Manage Credentials

IBM® Cognos® Connection is the portal to IBM® Cognos® Business Intelligence and provides a single access point to all corporate data available in IBM® Cognos® BI.

IBM® Cognos® BI supports authenticated and anonymous user access. To use IBM® Cognos® BI as an authenticated user, you must successfully log on. During the log on process, you must provide your credentials, such as user ID and password, as required by your organization. Anonymous users do not log on.

For more information, see the IBM® Cognos® Connection User Guide.

When using IBM® Cognos® Data Manager to access corporate data available in IBM® Cognos® BI, there are two types of credentials:

• Designer credentials are used while you are working in Data Manager Designer and are obtained from the IBM® Cognos® Connection log on window
• catalog credentials are used when you execute a build or JobStream and are read from the catalog during command line execution
This means, for example, that you could log on with Designer credentials as User A to set up a build using Data Manager Designer, and then execute the same build using the catalog credentials of User B.

**Designer Credentials**

When using IBM® Cognos® Data Manager Designer some actions require that you access IBM Cognos Connection to access an IBM® Cognos® data source or published IBM® Cognos® Framework Manager package.

You can either log on using Logon As from the Data Manager Actions menu, or you can wait until you perform an action for which you must connect to IBM® Cognos® Connection, such as creating a Published FM Package connection, when the Log on window automatically appears. If you have previously logged on during the current Designer session, you can access the data without logging on again.

**Steps to Log On to IBM Cognos Connection**

1. From the **Actions** menu, click **Logon As**.
   The IBM® Cognos® Connection Log on window appears.

2. Enter the information required to log on.
   For information, see the IBM® Cognos® Connection **User Guide**.
   **Tip:** You can also log on using the **Manage Catalog Credentials** dialog box.
   For more information, see "Catalog Credentials" (p. 72).

**Step to Log Off and End your IBM Cognos Connection Session**

- From the **Actions** menu, click **Logoff**.
  A message appears informing you that you are logged off.
  For information, see the IBM® Cognos® Connection **User Guide**.
  **Tip:** You can also log off using the **Manage Catalog Credentials** dialog box.
  For more information, see "Catalog Credentials" (p. 72).

**Catalog Credentials**

Catalog credentials are used when you execute a build or JobStream from the command line.

**Create the Stored Catalog Credentials**

If there are no catalog credentials stored, a status message appears to indicate this in the Manage Catalog Credentials dialog box. The user whose credentials you want to store must log on and store them.

**Steps**

1. From the **Actions** menu, click **Manage Execution Credentials**.
If you are not logged on to IBM® Cognos® Connection, you must do so before you can set the catalog credentials.

2. To log on, click Logon As, and enter the information required.
   For information, see the IBM® Cognos® Connection User Guide.

3. Click Create credentials, to store the catalog credentials.

**Manage the Catalog Credentials**

If the catalog credentials do not match the current Designer credentials that you have used to log on, a status message appears to indicate this in the Manage Catalog Credentials dialog box. If required, you can store your current Designer credentials as the catalog credentials.

**Steps**

1. From the Actions menu, click Manage Execution Credentials.
   If you are not logged on to IBM® Cognos® Connection, you must do so before you can set the catalog credentials.

2. To log on, click Logon As, and enter the information required.
   For information, see the IBM® Cognos® Connection User Guide.

3. Click Set to current account, to set the catalog credentials to be the same as the Designer credentials.

   Tip: To clear the catalog, click Clear catalog credentials.

**Renew the Stored Catalog Credentials**

The catalog credentials currently stored may not be valid if they have expired or the password has changed. If this occurs, the user whose credentials are stored must log on and renew them.

**Steps**

1. From the Actions menu, click Manage Execution Credentials.
   If you are not logged on to IBM® Cognos® Connection, you must do so before you can set the catalog credentials.

2. To log on, click Logon As, and enter the information required.
   For information, see the IBM® Cognos® Connection User Guide.

3. Click Renew credentials.

**Anonymous Users**

Anonymous users do not log on. However, you can choose to override the anonymous account and log on using Logon As from the IBM® Cognos® Data Manager Actions menu. This forces a logon to one of your other namespaces. You can repeat this process for as many namespaces as you have. If you have logged on to all your namespaces, and attempt to log on again, a message informs you
of this. Using anonymous access means that there is no authentication required to access IBM® Cognos® Business Intelligence data sources or published packages.
Chapter 8: Reference Dimensions

A reference dimension groups together all the reference structures (hierarchies, auto-level hierarchies, lookups) and templates that relate to a particular business dimension. Collectively, this is known as the dimensional framework.

For more information, see "Business Dimensions" (p. 26) and "Reference Structures" (p. 87).

View Reference Dimensions

You can view reference dimensions in the Tree pane from the Catalog tab or the Library tab.

The Dimensions folder contains the dimensional framework. At the top level, the folder contains all the reference dimensions used in the catalog. Each reference dimension corresponds to a single business dimension.

All reference structures and templates that relate to a reference dimension appear within the reference dimension.

Create a Reference Dimension

You must create a reference dimension to hold the required reference structures and templates.

Steps
1. In the Library folder, click Dimensions.
2. From the Insert menu, click Library, and then click Reference Dimension.
   The Dimension Properties dialog box appears.
3. Type a name for the reference dimension and, if required, a business name and description.
4. If the dimension is time-based, select the Dimension Represents Time check box.
5. Click OK.
**Change the Properties of a Reference Dimension**

You can change the properties of an existing reference dimension.

**Steps**

1. Click the reference dimension for which you want to change the properties.

2. From the **Edit** menu, click **Properties**.

3. Change the properties as you require.

4. Click **OK**.

**Delete a Reference Dimension**

You cannot delete a reference dimension if another component from the same catalog is using it.

For information about component dependencies, see "Locate Components and Their Dependencies" (p. 38).

**Steps**

1. Click the reference dimension that you want to delete.

2. From the **Edit** menu, click **Delete**.

3. In the message box, click **Yes**.

   If you attempt to delete a reference dimension that has dependent components, an error message appears.
Chapter 9: Templates

A database provides basic information about a table, principally the column names and data types. However, some columns have further special significance to IBM® Cognos® Data Manager and are an integral part of dimension table maintenance. This extra column information is referred to as the column behavior. The extent to which Data Manager uses column behavior depends on the complexity of the dimension management. This complexity can vary between a simple dimension table with a business key value and various attributes, to a dimension table which fully implements attribute change tracking.

In Data Manager, you use a template to

- define the attributes required in a reference structure
- populate the reference structures with dynamic data
- define the behavior of the attributes when delivering dimension data

By defining the behavior of attributes, Data Manager can manage surrogates and unmatched members, track attribute changes, and minimize updates to the data mart dimension tables.

For more information, see "Reference Structures" (p. 87), "Surrogate Keys" (p. 83), and "Track Attribute Changes" (p. 238). For information about unmatched members, see "Accept Source Data Containing Unmatched Dimension Values" (p. 179).

The following illustration shows how Data Manager uses templates to deliver data to a data mart.

To use a template to deliver structure data and transaction data, you must complete this process:
Create a template.

Create a reference structure to gather the structure data from the source system, enable aggregation, and to impose the structure of the data to be delivered. The reference structure uses a template to access the data and to specify the attributes for the extracted columns.

Create a dimension build and select the template to use to specify the columns to deliver. In addition, specify whether to use attribute change tracking to deliver dimension data.

Create a fact build to deliver the fact data.

Specify how unmatched members are to be handled.

You can create a number of different templates for the same reference dimension. This means that by changing the template you can assign a different behavior set to any reference structure, dimension build, or dimension delivery.

Before you can create a template, you must create a reference dimension to hold the template. See "Create a Reference Dimension" (p. 75).

Dimension Attributes

A dimension consists of members. For example, the members of a product dimension are the individual products. Members have attributes that identify them and provide further information. For example, some possible attributes for a product dimension are the product code, name, type, color, and size.

Each member of a dimension must have a business key to identify it in streams of transactional data. If the dimension is defined as a hierarchy, the lower levels of the hierarchy must also have an attribute that identifies the parent of each member. Another common attribute is the business name, which enables analysis software to make reports more comprehensible by using the business name in place of the business key.

In the environment of a data warehouse, information about each business dimension is stored in one or more dimension tables. Because it is common to store as much information as possible in the dimension tables, a dimension table may have many columns, each of which corresponds to a dimension attribute.

View Templates

All reference structures and templates that relate to a particular business dimension appear within a reference dimension in the Tree pane.
Create a Template

You must select a template for every reference structure, dimension build, and dimension delivery in your catalog.

Note: You do not have to create a separate template in each place that you select one. You can use any template, with suitable attributes defined, within the reference dimension.

When you create a template, you define attributes for it so that attributes are available when you create a reference structure.

Typically, a simple dimension table has attributes such as ID, caption, and surrogate. If you want to track attribute changes, you use attributes such as effective start date and current indicator.

Steps

1. In the appropriate Dimensions folder, click the Templates folder.
2. From the Insert menu, click Library, and then click Template.
   The Template Properties window appears.
3. Click the General tab.
4. In the Name box, type a name for the template and, if required, a business name and description.
5. Click the Attributes tab.
6. Click Import Table if the attributes you want to define for the template are already defined as columns in a database table.
   The Select Table dialog box appears.
7. Select the table from which to import attributes into the template, and then click OK.
   The database table columns are inserted into the Attribute Name column.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Behavior</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductNumber</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProductEndDate</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProductName</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProductTypeCode</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProductCost</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Margin</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PriceURL</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By default, IBM® Cognos® Data Manager allocates a behavior type of normal.
Tip: If you do not want to import a table or you want to add further attributes, you can define them manually by clicking Add.

8. In the Behavior box for an attribute, click the required behavior type.

For information about behavior types and additional property values, see "Behavior Types in Templates" (p. 80).

9. If you select Surrogate Key, Business Key, or Current Indicator as the behavior type, you must specify additional property values in the Value column. Enter appropriate values for the selected behavior type.

A template can have, at most, only one start date, end date, and current indicator. In addition, a business key can only be used by one surrogate key attribute.

10. Click OK.

Behavior Types in Templates

When you create a template, you define attributes for it. For each attribute in the template, you must assign a behavior type.

Surrogate Key Behavior

The surrogate key for the dimension table.

In the Value column, you must specify

● the column to which you have assigned business key behavior

● a start value if you want to ensure unique surrogate keys

Type the initial number to be assigned by IBM® Cognos® Data Manager when generating surrogate values.

By default, Data Manager starts each surrogate key series at 1.

For more information, see "Surrogate Keys" (p. 83).

Business Key Behavior

The identifier for the dimension table.

If a hierarchy has several levels, the template may have several business keys, one for each hierarchy level ID. However, only one of these business keys can be the primary key for the table. The lowest level business key should be defined as the primary key.

In the Value column, specify the primary key by clicking True or False.

Note: If you used a composite key to define the business ID, you must use a DataStream derivation to concatenate it. For more information, see "Derivations" (p. 155).
**Effective Start Date Behavior**

The earliest date to which the dimension data row applies. IBM® Cognos® Data Manager uses this date to track attribute changes.

The range over which the data row applies is from the effective start date to the effective end date inclusive (or the current date if there is no end date).

For more information, see "Specify Effective Date Options" (p. 247).

**Effective End Date Behavior**

The last date to which the dimension data row applies. IBM® Cognos® Data Manager uses this date to track attribute changes.

For more information, see "Specify Effective Date Options" (p. 247).

**Create Date Behavior**

The date on which IBM® Cognos® Data Manager created the dimension data row. Data Manager uses this date to track attribute changes.

**Last Update Date Behavior**

The date on which IBM® Cognos® Data Manager last updated the dimension data row. Data Manager uses this date to track attribute changes.

**Current Indicator Behavior**

Indicates whether the dimension data row contains the current value or past value for the member. IBM® Cognos® Data Manager uses this value to track attribute changes.

In the Value column, specify the current value and past value. You can specify any non-null value you require. By default, Data Manager enters Y for current value and N for past value.

**Normal Behavior**

Any other attributes.

**Change the Default Column Names for Behavior Types**

For some behavior types, IBM® Cognos® Data Manager allocates a default column name, which you can change if required. The following table shows the default column names.

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Default column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective start date</td>
<td>eff_date</td>
</tr>
<tr>
<td>Effective end date</td>
<td>end_date</td>
</tr>
<tr>
<td>Create date</td>
<td>cre_date</td>
</tr>
<tr>
<td>Behavior</td>
<td>Default column name</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Last update date</td>
<td>udt_date</td>
</tr>
<tr>
<td>Current indicator</td>
<td>curr_ind</td>
</tr>
<tr>
<td>Current value</td>
<td>Y</td>
</tr>
<tr>
<td>Past value</td>
<td>N</td>
</tr>
<tr>
<td>Surrogate key</td>
<td>skey</td>
</tr>
</tbody>
</table>

**Steps**

1. From the **Tools** menu, click **Options**.
2. Click the **Attribute Defaults** tab.
3. Enter values as required.
   - **Tip**: You can revert to the default column names by clicking **Reset Defaults**.
4. Click **OK**.

**Change the Properties of a Template**

You can change the properties of an existing template.

**Steps**

1. Click the template for which you want to change the properties.
2. From the **Edit** menu, click **Properties**.
3. Change the properties as you require.
4. Click **OK**.

**Delete a Template**

You cannot delete a template if another component from the same catalog is using it.

For information about component dependencies, see "**Locate Components and Their Dependencies**" (p. 38).

**Steps**

1. Click the template that you want to delete.
2. From the **Edit** menu, click **Delete**.
3. In the message box, click **Yes**.
Chapter 10: Surrogate Keys

Surrogate keys are numeric keys that can replace business keys in the context of a data mart. Using surrogate keys has many advantages:

- Surrogate keys are numeric.
  
  Therefore, they tend to be smaller than the keys from operational systems, which are typically text-based.

- Surrogate keys are unique.
  
  Operational keys may be unique within the database in which they reside. However, they may not be unique across a build that accesses data from multiple databases. To implement attribute change tracking, the keys must be unique.

- Surrogate keys allow you to combine data from tables with incompatible keys.

- Surrogate keys have no intrinsic meaning.
  
  Therefore, they are not subject to the same changes as operational keys. For example, product codes may expand to accommodate additional information.

The following table shows an extract of the product reference data.

<table>
<thead>
<tr>
<th>Product_code</th>
<th>Product_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRX 002 39 061</td>
<td>Trailchef water bag</td>
</tr>
<tr>
<td>PRX 003 40 067</td>
<td>Trailchef canteen</td>
</tr>
<tr>
<td>PRY 003 47 086</td>
<td>Trailchef kitchen set</td>
</tr>
</tbody>
</table>

The second table illustrates the use of surrogate keys for product reference data.

<table>
<thead>
<tr>
<th>Surrogate_id</th>
<th>Product_code</th>
<th>Product_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRX 002 39 061</td>
<td>Trailchef water bag</td>
</tr>
<tr>
<td>2</td>
<td>PRX 003 40 067</td>
<td>Trailchef canteen</td>
</tr>
<tr>
<td>3</td>
<td>PRY 003 47 086</td>
<td>Trailchef kitchen set</td>
</tr>
</tbody>
</table>

Implement Surrogate Keys

To use surrogate keys in IBM® Cognos® Data Manager, you must first implement them in the appropriate reference structures. When you specify the template to use, ensure that
the template includes an attribute that has surrogate key behavior
the surrogate key is linked to a reference structure attribute assigned as a business key

In the following template example, the attribute named skey has surrogate key behavior. It is linked to the ProductNumber attribute, which is defined as the primary business key for the template.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Behavior</th>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>skey</td>
<td>Surrogate Key</td>
<td>Business Key</td>
<td>ProductNumber</td>
</tr>
<tr>
<td>ProductNumber</td>
<td>Business Key</td>
<td>Primary Key</td>
<td>Tuo</td>
</tr>
<tr>
<td>ProductName</td>
<td>Normal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For information about attributes with surrogate key behavior, see "Behavior Types in Templates" (p. 80).

For more information about specifying a template to use for a reference structure, see "Reference Structures" (p. 87).

**Maintain Surrogate Key Values**

After you implement surrogate keys in a reference structure, you can use them in

- fact builds that deliver surrogate keys in the output
- dimension deliveries
- dimension builds

By including a dimension element (in a fact build) or a template (in a dimension delivery or dimension build) for which you have implemented surrogate key behavior, IBM® Cognos® Data Manager assigns any required surrogate key values when you execute the build.

To maintain the surrogate key values, you must specify

- the source from which the surrogate key values are assigned
- the way in which the surrogate key values are delivered

Note: In a dimension build, it is common practice for Data Manager to generate surrogate values automatically. In this case, you do not need to specify a source for surrogate key values. For information, see "Generate Surrogate Key Values Automatically" (p. 86).

**Specify the Source of Surrogate Keys in a Reference Structure**

You can use surrogate keys values from a source that you specify. For example, you may have a text file that contains surrogate key values.

**Step From the DataStream Mapping Window**

- Map a DataStream item to the surrogate key level attribute.

  For more information, see "Map the DataStream Items to Level Attributes" (p. 101).

**Step From a Reference Structure Properties Window**

- Select the Reference Surrogate check box.
Note: Use reference surrogates if you want IBM® Cognos® Data Manager to automatically generate surrogate key values in a fact build or dimension delivery. For more information, see "Reference Surrogate Keys" (p. 85).

**Deliver Sourced Surrogate Key Values**

When you execute a build, IBM® Cognos® Data Manager delivers the current surrogate value of each dimension member either from the source you specified or from the values that Data Manager generates by using reference surrogates.

You can specify that a fact build, a dimension build, or a dimension delivery is to deliver sourced surrogate key values in the output.

**Step Using a Fact Build**

- When you define the output levels for a dimension element, select the **Use surrogates when available** check box.

  For information, see "Set Output Levels for a Dimension or Derived Dimension" (p. 175).

**Step Using a Dimension Build or Dimension Delivery**

- When you define the columns for a dimension table, you must select one of these column tokens as the source for any attribute with surrogate key behavior, either <level>.$SURROGATE, $SURROGATE, or $PARENT_SURROGATE.

In the following example dimension table, two attributes have surrogate key behavior:

- **Product skey**, which is sourced from the $SURROGATE column token
- **ProductType skey**, which is sourced from the ProductType $SURROGATE column token

<table>
<thead>
<tr>
<th>Column</th>
<th>Sourced From</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product skey</td>
<td>$SURROGATE</td>
<td></td>
</tr>
<tr>
<td>Product skey</td>
<td>ProductName</td>
<td></td>
</tr>
<tr>
<td>ProductNumber</td>
<td>ProductNumber</td>
<td></td>
</tr>
<tr>
<td>ProductName</td>
<td>ProductName</td>
<td></td>
</tr>
<tr>
<td>ProductType</td>
<td>ProductType</td>
<td></td>
</tr>
<tr>
<td>ProductType</td>
<td>ProductType</td>
<td></td>
</tr>
<tr>
<td>Product skey</td>
<td>ProductType</td>
<td></td>
</tr>
<tr>
<td>ProductType</td>
<td>ProductLine</td>
<td></td>
</tr>
<tr>
<td>Product skey</td>
<td>ProductLine</td>
<td></td>
</tr>
<tr>
<td>ProductLine</td>
<td>ProductLine</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>AllProductID</td>
<td></td>
</tr>
<tr>
<td>caption</td>
<td>AllProductID</td>
<td></td>
</tr>
<tr>
<td>ProductType skey</td>
<td>ProductType $SURROGATE</td>
<td></td>
</tr>
</tbody>
</table>

For more information about sourcing dimension table columns and column tokens, see "Add a Table to a Dimension Build" (p. 236).

**Reference Surrogate Keys**

Reference surrogate keys are surrogate keys that are not automatically maintained. They are provided for backward compatibility with IBM® Cognos® DecisionStream 6.0 and to support snapshot data marts. By default, reference surrogate keys are not maintained and are regenerated each time you execute a build.
You can force IBM® Cognos® Data Manager to maintain surrogate keys values by using reference surrogates. When you deliver the dimension data, Data Manager assigns new surrogate key values for each dimension member by using the surrogate key start value defined in the associated template.

**Generate Surrogate Key Values Automatically**

You can generate surrogate key values automatically in a dimension build without sourcing them.

To specify that surrogate key values are to be generated, when you define the columns for a dimension table (in the Dimension Table Properties window), do not select a source for any attribute with surrogate key behavior.

<table>
<thead>
<tr>
<th>Column</th>
<th>Sourced From</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Surrogate Key for ProductNumber</td>
<td>☑</td>
</tr>
<tr>
<td>ProductNumber</td>
<td>Product.ProductNumber</td>
<td>☑</td>
</tr>
<tr>
<td>ProductName</td>
<td>Product.ProductName</td>
<td>☑</td>
</tr>
<tr>
<td>ProductTypeCode</td>
<td>ProductType.ProductTypeCode</td>
<td>☑</td>
</tr>
</tbody>
</table>

For information about setting up dimension table columns, see "Add a Table to a Dimension Build" (p. 236).

When you execute a dimension build, the surrogate key values are generated using the appropriate surrogate key start values defined in the associated template, and are delivered to the dimension table.

**Determine the Maximum and Minimum Surrogate Values**

You can use SQL to determine the current minimum or maximum values of a surrogate key. This may be useful before adding a level to a hierarchy.

**Example**

The dimension table named MyTable stores surrogate key values in the column named Key. To determine the maximum surrogate key value, use

```
SELECT MAX(Key) FROM MyTable;
```
Chapter 11: Reference Structures

Reference structures model the dimensions of a business and provide context for fact data. You deliver reference structures to target data marts, where they describe the dimensions of the fact data, and provide descriptive and navigational information for analysis tools. IBM® Cognos® Data Manager builds the reference structures from the structure data that you specify.

Data Manager has these types of reference structure:

- hierarchies
- auto-level hierarchies
- lookups

Reference structures, together with their associated templates, are collectively known as the dimensional framework.

The dimensional framework is crucial for coordinated data marts which have shared dimensions. These tables provide a common definition of business dimensions for the various fact tables in the warehouse.

View Reference Structures

You can view reference structures in the Tree pane or the Visualization pane.

View Reference Structures in the Tree Pane

All reference structures and templates that relate to a particular business dimension appear within a reference dimension in the Tree pane.

IBM® Cognos® Data Manager represents reference dimensions, reference structures, and templates in the Dimensions folder as follows.

View Reference Structures in the Visualization Pane

When you click a reference structure in the Tree pane, IBM® Cognos® Data Manager shows a visual representation of the whole structure in the Visualization pane.
The information shown includes

- The database from which reference data is being accessed.
- The method used to access data (a template, DataStream or both).
- Any mapping between the DataStream and the reference structure. The following methods are used to indicate the mapping state:
  - No line means that a template is used for every level.
  - An arrow with a solid line means that a DataStream is used for every level.
  - An arrow with a dotted line means that data is accessed using a template but there is also a data source with the related mapping in place, but the data source mapping is not being used as the source.

This example shows a hierarchy.

For information on each of the symbols used in the Visualization pane, see "Buttons and Icons" (p. 549).

Use the Hierarchy Wizard

The Hierarchy wizard can help you to create hierarchies and auto-level hierarchies. The wizard interprets commonly found reference data representations and creates a hierarchy and levels, a means of populating dynamic members, and a static All member to represent the top of the hierarchy tree.

When you use the Hierarchy wizard, you

- select the hierarchy structure
- define the hierarchy
- select the source data
- optionally define an all level
- define the levels
• assign the remaining attributes

When you have created a hierarchy using the Hierarchy wizard, you can amend it, and add to it. For example, attributes such as ID, parent, and caption are defined using the wizard, but you may want to add other attributes that help to describe the data.

Note: You can use the Hierarchy wizard only if the structure data is accessible to the computer on which IBM® Cognos® Data Manager is operating. If the structure data cannot be accessed directly, you must create the hierarchy manually.

The format of the wizard depends upon the structure of the source data that you are using to create the hierarchy. You can choose from three options.

Create the Hierarchy from the Columns of One Table (Star Schema)

Use this option when the source data is a single table with columns representing levels in the hierarchy.

The following example shows a fiscal hierarchy based on the relationship between columns in the same data row. The source table includes columns named fiscal_yr, fiscal_qtr, and period_no. Each year contains quarters, and each quarter contains periods.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>200501</td>
<td>200501</td>
</tr>
<tr>
<td>2005</td>
<td>200502</td>
<td>200502</td>
</tr>
<tr>
<td>2005</td>
<td>200503</td>
<td>200503</td>
</tr>
<tr>
<td>2005</td>
<td>200504</td>
<td>200504</td>
</tr>
<tr>
<td>2005</td>
<td>200505</td>
<td>200505</td>
</tr>
<tr>
<td>2005</td>
<td>200506</td>
<td>200506</td>
</tr>
</tbody>
</table>

Create the Hierarchy from the Rows of One Table

Use this option when the source data is a single table with sets of rows for each level in the hierarchy, with the rows related by a parent ID column.

The following example shows a product hierarchy based upon the relationships between rows of the same table. In relational terms, these are recursive relationships. The source table includes columns named product_type, product_cd, and parent_product_cd. Within the parent_product_cd column, each row refers to the product_cd value of its parent.
Tip: To create an auto-level hierarchy using the Hierarchy wizard, you must choose this option. On the page in the wizard where you can define an All level, in the Select the column for the level name box, click Auto-Level.

Create the Hierarchy from Multiple Tables (Snowflake Schema)

Use this option when the source data comes from multiple tables in which each table represents a single level in the hierarchy. Each row has a parent ID that relates to another row in its parent table.

In the following example, the ProductLine, ProductType, and Product levels each come from a different table.
For examples of creating hierarchies using the Hierarchy wizard, see IBM® Cognos® Data Manager Getting Started. For general information about hierarchies, see "Hierarchies" (p. 91).

Steps
1. From the Tools menu, click Hierarchy Wizard.
2. Follow the instructions.

Hierarchies
A hierarchy presents a particular view of a business dimension. It organizes the structure data into levels that represent parent-child relationships. Each hierarchy can have as many levels as you require. Each level contains a set of members.

IBM® Cognos® Data Manager can create hierarchies from any relational data, but typically from star schemas, snowflake schemas, and parent-child schemas.

You can create a hierarchy manually, or you can use the Hierarchy wizard.

To create a hierarchy manually, you must
- create a reference dimension
- create a hierarchy
- create hierarchy levels and define the attributes
- reposition the hierarchy levels
- acquire the structure data
- if the data is to be acquired using a DataStream, map the DataStream items to the level attributes
- define fostering
- define error handling

For information on using the Hierarchy wizard, see "Use the Hierarchy Wizard" (p. 88).

Create a Hierarchy
You must create a hierarchy for a dimension if you want to organize the structure data into levels that represent parent-child relationships.

Steps
1. In the Dimensions folder, click the reference dimension to which you want to add the hierarchy.
2. From the Insert menu, click Library, and then click Hierarchy.
   The Hierarchy Properties window appears.
3. Click the General tab.
4. Type a name for the new hierarchy and, if you require, a business name and description.
5. If you do not want IBM® Cognos® Data Manager to automatically provide foster parents for members without explicit parents, select the **Disable fostering** check box.

   For more information, see "Foster Parents" (p. 103).

6. If the hierarchy is to support reference surrogate keys, select the **Reference Surrogate** check box.

   For more information, see "Reference Surrogate Keys" (p. 85).

7. Click **OK**.

   You can now create the hierarchy levels.

### Create Hierarchy Levels and Define the Attributes

Each hierarchy level consists of a DataStream, a set of members containing attributes, and optionally, static members.

Members at a level can relate to their parents at the next highest level and their children at the next lowest level.

Each member of a hierarchy level must have attributes. The available attributes are determined by associating a template with the level. A level must include an ID attribute to identify each member of the level. A level usually has an attribute to provide a caption (business name), and an attribute to identify the parent of each member of the level. A level may also include attributes to explicitly support surrogate keys or attribute change tracking. You can define as many attributes as you require.

A hierarchy level can acquire structure data through the hierarchy DataStream, a level DataStream, or a level template.

If the hierarchy is to acquire structure data through a DataStream rather than a template, you must still associate a template with the hierarchy to determine the level attributes. For more information, see "Create a Template" (p. 79).

### Steps

1. Click the hierarchy to which you want to add a level.

2. From the **Insert** menu, click **Library**, and then click **Level**.

   The **Level Properties** window appears.

3. Click the **General** tab.

4. Type a name for the level and, if required, a business name and description.

5. If you want IBM® Cognos® Data Manager to sort members of the level numerically instead of alphanumerically when a build is executed, select the **Use numeric sort on ID** check box.

6. Click the **Attributes** tab.

7. In the **Template** box, click the template to use for this level.

   **Tip:** To create a template or edit an existing template, click **New** or **Edit** as appropriate.
For information about templates, see "Templates" (p. 77).

8. Move the required attributes for the level from the Available Attributes pane to the Chosen Attributes pane.

9. In the Chosen Attributes pane, select the ID check box for the attribute that provides the ID for the level.

10. If the level is not at the top of the hierarchy, and you do not want to attach all members to the foster hierarchy, you must also define a parent attribute, by selecting the Parent check box for the appropriate attribute.

11. Optionally, define a caption attribute by selecting the Caption check box for the appropriate attribute.

12. If you want to use a temporary file-based cache for attributes in order to reduce the memory used, select the Use file caching for attributes box.

13. Click OK.

After you define the level attributes, you must specify whether the level is to access source data using a template or a DataStream. If you specify a DataStream, you must map the DataStream items to the level attributes.

For more information, see "Set Up Data Access" (p. 99) and "Map the DataStream Items to Level Attributes" (p. 101).

Reposition the Hierarchy Levels

When you add a level to a hierarchy, IBM® Cognos® Data Manager places it below the other levels for the hierarchy. The order in which the levels are shown in the Tree pane, must correctly reflect the parent-child relationships, in a top-down order.

Steps
1. In the Tree pane, click the level to reposition.
2. From the Actions menu, click Move Up or Move Down as appropriate.

Create Recursive Levels

If the data for a hierarchy contains parent-child relationships, you can create a recursive hierarchy level which allows you to explicitly name the levels contained in the recursive relationships.

Note: Using recursive levels differs from using an auto-level hierarchy because it is level-based.

A recursive level can acquire structure data through a level DataStream or a level template, but not the hierarchy DataStream.

Note: If a hierarchy contains one or more recursive levels, the hierarchy DataStream becomes inactive.
When you create a recursive level, IBM® Cognos® Data Manager can use different apex detection methods to determine which nodes should be the top level parent nodes. By default, it uses nodes that have missing parents. If required, you can select a different apex detection method:

- **member values**
  Select this option if all the apex member values exist at the top of the required level, and you want to explicitly state them. Data Manager uses these values as the apex of each branch.

- **parent values (default)**
  Select this option to choose the apex members based on a set of explicit values of the parent attribute of the recursive level. By default, members with a NULL value for the parent attribute are selected.

- **parent existence**
  Select this option if all the apex member values exist at the level above the required level, but you do not know what those values are, so cannot explicitly state them.

  **Note:** You set up a separate hierarchy level at the level directly above the recursive level to use this option.

If you are interested only in a subset of the data, the apex does not have to be the top of the recursive level. By setting the apex to the top of the recursive level, if you have a very large hierarchy, Data Manager must search the entire hierarchy to work out the structure of the data. If you set the apex to a dependent level, you restrict the recursive structure to the nodes in the hierarchy from a specific point.

For example, you may have a recursive level named Continent. If you set the apex to Continent, all data that exists for each continent is included in the hierarchy. However, you may be only interested in data for a particular country. In this case, you would specify a particular country ID as the apex.

  **Note:** You can also specify more than one ID attribute as the apex by typing a comma-separated list of IDs. For example, 'A', 'B', 'C'.

For more information on hierarchy transformation, see "Constructing and Transforming Hierarchies" (p. 121).

**Steps**

1. Click the hierarchy to which you want to add a recursive level.
2. From the **Insert** menu, click **Library**, and then click **Recursive Level**.
   The **Recursive Level Properties** window appears.
3. Click the **General** tab.
4. Type a name for the level that you want to appear at the top of the recursive level and, if required, a business name and description.
5. If you want Data Manager to sort members of the level numerically instead of alphanumerically when a build is executed, select the **Use numeric sort on ID** check box.
6. Click the **Sub-Levels** tab.

7. To create a dependent recursive level, click **Add**, and then type a name for the level.
   
   **Tip:** You can add a full business name and description to a dependent level. Select the level and then click **Edit Details**.

8. Click **OK**.

9. Repeat steps 7 and 8 to create further dependent levels.
   
   - The order of the dependent levels on this tab reflects the parent-child relationships, in a top-down order. To reposition a dependent level, click the level and then click **Move Up** or **Move Down** as appropriate.
   
   - To edit the name of a dependent level, click the required level, and then click **Edit**.
   
   - To delete a dependent level, click the required level, and then click **Delete**.

10. In the **Apex detection method** box, specify the detection method you want to use.
    
    If you select **Parent values** or **Member values**, type the values in the adjacent box.

11. Click the **Attributes** tab and define the level attributes for the recursive level.
    
    For more information, see "Create Hierarchy Levels and Define the Attributes" (p. 92).

12. Click **OK**.
    
    After you define the level attributes, you must specify whether the recursive level is to access source data using a template or a DataStream. If you specify a DataStream, you must map the DataStream items to the level attributes.
    
    For more information, see "Set Up Data Access" (p. 99) and "Map the DataStream Items to Level Attributes" (p. 101).
    
    **Note:** You should perform mapping at the recursive level, not at the hierarchy level.

### Auto-level Hierarchies

Auto-level hierarchies are hierarchies from parent-child schemas where the structure data table contains no hierarchy level information. IBM® Cognos® Data Manager ascertains the number of levels required and builds the hierarchy by following the parent-child relationships between the structure data rows.

For example, the following table contains no level information. Data Manager ascertains the levels required using the values in the **Parent** column.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Rodriguez</td>
<td>00</td>
</tr>
<tr>
<td>02</td>
<td>Wilcox</td>
<td>01</td>
</tr>
</tbody>
</table>
The hierarchicial representation of the table looks like this.

![Diagram of the hierarchy]

Note: You can also set up a parent-child structure as a hierarchy with recursive levels. For more information, see "Create Recursive Levels" (p. 93).

### Top Parent ID

To help IBM® Cognos® Data Manager ascertain the structure of an auto-level hierarchy, you specify the top parent ID used in the source data. The top parent ID is the value of the parent attribute of the top-level member, not the ID of the top level member.

For example, the top parent ID for this table is 00.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Rodriguez</td>
<td>00</td>
</tr>
<tr>
<td>02</td>
<td>Wilcox</td>
<td>01</td>
</tr>
<tr>
<td>03</td>
<td>Floyd</td>
<td>01</td>
</tr>
</tbody>
</table>

To create an auto-level hierarchy manually you must

- create a reference dimension
- create an empty auto-level hierarchy and define the attributes
- acquire structure data
- define fostering
- define error handling
Tip: To create an auto-level hierarchy using the Hierarchy wizard, you must choose the Create a hierarchy from the rows of one table option on the first page of the wizard. On the page where you define the source columns, in the Select the column for the level name box, click Auto-Level.

Create an Auto-level Hierarchy and Define the Attributes

You must use an auto-level hierarchy when the structure data contains no hierarchy level information. An auto-level hierarchy consists of a single DataStream, a set of members containing attributes, and, optionally, static members.

Members relate to their parent members at the next highest level and their children members the next lowest level. Each member must have attributes. The available attributes are determined by associating a template with the auto-level hierarchy. An auto-level hierarchy must include at least an ID attribute to identify each member and a parent attribute so that IBM® Cognos® Data Manager can correctly determine the levels from the row in the data source. You can define as many attributes as you require.

An auto-level hierarchy can acquire structure data through its DataStream or template. The levels for an auto-level hierarchy are generated when the build is executed. This means that there are no level DataStreams, so data must be accessed using the auto-level hierarchy DataStream, or template.

Note: If the hierarchy is to acquire structure data through a DataStream rather than a template, you must still associate a template with the hierarchy to determine the level attributes. For more information, see “Create a Template” (p. 79).

Steps

1. In the Dimensions folder, click the reference dimension to which you want to add the auto-level hierarchy.

2. From the Insert menu, click Library, and then click Auto-Level Hierarchy. The Auto-Level Hierarchy Properties window appears.

3. Click the General tab.

4. Type a name for the new auto-level hierarchy and, if you require, a business name and description.

5. If the structure data table uses a special value in the parent column to indicate the top member, type this value in the Top parent ID box. Otherwise, leave the box empty.

   The top parent ID is the value of the parent attribute of the top-level member, not the ID of the top level member. By default the top parent ID is null because the top member has no parents.

6. If you want Data Manager to sort the members of the hierarchy numerically, select the Use numeric sort on ID check box.

   By default, Data Manager sorts IDs alphanumerically.

7. If the auto-level hierarchy is to support reference surrogate keys, select the Reference surrogates check box.
8. Click the Attributes tab.

9. In the Template box, click the template that defines the attributes for the auto-level hierarchy.
   **Tip:** To create a template, or edit an existing template, click New or Edit as appropriate. For more information, see "Templates" (p. 77).

10. Move the required attributes from the Available Attributes pane to the Chosen Attributes pane.

11. In the Chosen Attributes pane, select the ID check box for the attribute that provides the ID for the auto-level hierarchy, and the Parent check box for the attribute that provides the parent. Optionally, select the Caption check box for the appropriate attribute.

12. If you want to use a temporary file-based cache for attributes in order to reduce the memory used, select the Use file caching for attributes box.

13. Click OK.

   You must now specify whether data is to be accessed using a template or a DataStream. For information see, "Set Up Data Access" (p. 99).

   **Note:** You can balance an auto-level hierarchy, if required. For more information, see "Balance a Hierarchy" (p. 122).

---

### Lookups

A lookup is a simple, single-level reference structure with no parent-child relationships. Use a lookup when you have a set of reference members that you do not need to organize hierarchically.

Common uses for lookups are to:

- check the data integrity of fact rows against a single level of a conformed dimension
- clean complex unstructured data from source systems before hierarchies can be properly determined
- lookup specific values from a reference structure based on a lookup key
- perform late arriving fact processing
- create tables to assist in data transformation that are not required for dimensional analysis

For example, you create a currency conversion table to allow different currencies to be translated to a standard currency. The currency table will never by used for dimensional analysis, so a lookup may be more appropriate.

To create a lookup, you must

- create a reference dimension
- create an empty lookup and define the attributes
- acquire structure data
Create a Lookup and Define the Attributes

A lookup consists of a single DataStream, a set of members containing attributes, and, optionally, static members.

Each member must have attributes. The available attributes are determined by associating a template with the lookup. A lookup must include at least an ID attribute to identify each member. You can define as many attributes as you require.

Steps
1. In the Dimensions folder, click the reference dimension to which you want to add the lookup.
2. From the Insert menu, click Library, and then click Lookup.
   The Lookup Properties window appears.
3. Click the General tab.
4. Type a name for the lookup and, if you require, a business name and description.
5. If you want IBM® Cognos® Data Manager to sort the IDs of this lookup numerically, select the Use numeric sort on ID check box.
   By default, Data Manager sorts IDs alphanumerically.
6. If the lookup is to support reference surrogate keys, select the Reference surrogates check box.
   For more information, see "Reference Surrogate Keys" (p. 85).
7. Click the Attributes tab.
8. In the Template box, click the template that defines the attributes for the lookup.
   Tip: To create a template, or edit an existing template, click New or Edit as appropriate. For more information, see "Templates" (p. 77).
9. Move the required attributes from the Available Attributes pane to the Chosen Attributes pane.
10. In the Chosen Attributes pane, select the ID check box for the attribute that provides the ID for the lookup. Optionally, select the Caption check box for the appropriate attribute.
11. If you want to use a temporary file-based cache for attributes in order to reduce the memory used, select the Use file caching for attributes box.
12. Click OK.
   You must now specify whether data is to be accessed using a template or a DataStream. For information, see "Set Up Data Access" (p. 99).

Set Up Data Access

You can access data using a template or a DataStream.
Template access is most commonly used

- Against data that does not require custom SQL.
- When a dimension table was built using a dimension build and template and then the same template is used to access the data.
- If you are accessing dimensional data that you delivered to the warehouse using a template, then when you next access this data you should again use the template. This allows IBM Cognos Data Manager to access all the columns it requires and know the intended behavior of each column without this work having to be done again.

A template generates SQL against a single table, and cannot be used against multiple tables. You cannot use a template to access data if literals have been defined on the DataStream.

For a hierarchy, you must specify a template to populate each level in the hierarchy, not the hierarchy as a whole.

For more information, see "Templates" (p. 77).

DataStream access uses the SQL declared in the data source and gives you direct control over the SQL. You can join tables, use calculated fields, and use ORDER BY or GROUP BY clauses. DataStream access is commonly used against data from operational systems.

You can specify that Data Manager is to access data using a DataStream for a complete hierarchy or for each hierarchy level.

Note: If a hierarchy contains one or more recursive levels, the hierarchy DataStream becomes inactive.

Steps Using a Template

1. Click the hierarchy level, auto-level hierarchy, or lookup.
2. From the Edit menu, click Properties.
   The Properties window for the selected reference structure appears.
3. Click the Data Access tab.
4. Click Use template for data access.
   Note: The template that Data Manager will use is the one specified on the Attributes tab.
5. In the Connection box, click the connection for the database in which the structure data resides.
   Tip: To create a connection, click New. For more information, see "Create a Database Connection" (p. 63).
6. In the Table name box, enter the name of the structure data table.
7. In the Distributed lock table name box, enter the name of the lock table to use when distributed handling of unmatched members is enabled for fact builds.
   For more information, see "Distributed Handling of Unmatched Members" (p. 181).
8. If it is possible that the data is not distinct, select the Use DISTINCT in template SELECT statement check box.

Selecting this check box adds a DISTINCT clause to the database SELECT statement that returns the reference structure elements. This ensures that duplicate rows are removed before the data is returned to Data Manager.

When you access a data source using a template, all the levels defined by the hierarchy for which the template is related are read back. In this case, you should not select this check box as all the data will be retrieved anyway. In the same situation, you may want to access only higher levels of a hierarchy. In this case, selecting this check box results in fewer records being returned from the database containing the reference structure.

9. Click OK.

Steps Using a DataStream

1. Click the hierarchy, hierarchy level, auto-level hierarchy, or lookup.

2. From the Edit menu, click Properties.

   The Properties window for the selected reference structure appears.

3. Click the Data Access tab.

4. Click Use DataStream for data access.

5. Click OK.

   You must now map the DataStream items to the level attributes.

Map the DataStream Items to Level Attributes

To enable the flow of data from a data source to the level attributes, you must map the DataStream items to the level attributes.

Steps

1. Click the hierarchy, hierarchy level, auto-level hierarchy, or lookup for which you want to perform mapping.

2. From the Edit menu, click Mapping.

3. In the DataStream Item column, click the item to which you want to map an attribute.

4. In the Level Attributes pane, click the attribute to map.

   Tip: To add an attribute, click Add and then click either
   
   • Create a new attribute and type a name for the attribute to create
   
   • Subscribe to unused template attribute, and click the required attribute

5. Click Map.
Tip: You can also perform mapping by dragging the level attribute to the relevant position in the Maps To column.

6. Repeat steps 3 to 5 until you complete the required mappings.
   
   Note: You can map an attribute to more than one DataStream item.
   
   Tip: To clear the mapping, click the mapped item in the Maps To column, and click Clear.

7. Click OK.

**Populate the Reference Structure**

IBM® Cognos® Data Manager can use both static members and dynamic members to populate a reference structure.

**Static Members**

Static members

- are predefined
- often define an All member at the top of a hierarchy, which can act as a foster parent for any record that does not have a parent associated with it
- are useful for things that never change, such as months
- do not exist in the source data as they are defined and stored entirely in the catalog

Note: You must define the attributes of the reference structure before adding static members.

**Dynamic Members**

Dynamic members

- are populated using SQL statements to extract data from reference tables in a database
- change as the reference data changes
- should be used wherever possible

IBM® Cognos® Data Manager acquires dynamic structure data when a build is executed. Dynamic structure data can reside in master tables, transaction tables, or a combination of both. Master tables contain semi-static data that a transactional system uses to define valid transaction data. For example, a master table may exist to define a product range or to provide information about salespersons.

It is normal for master tables to provide structure data when these master tables describe all the associated entities, such as, all products or all salespersons.

When adequate master tables do not exist, transaction tables may provide the required structure data. This is particularly true when the transaction data contains the business keys and business names for the associated entities.

When they do not describe the complete range of the associated entities, you can supplement the master tables with transaction data.
Add a Static Member

This section describes how to add a static member.

Steps
1. In the appropriate reference structure, click Static Members.

2. From the Edit menu, click Properties.

   The Static Members window appears. This window has a column that corresponds to the attributes you defined on the Attributes tab of the Level Properties window. In addition to the column for the required ID attribute. If you are adding a static member to a hierarchy or auto-level hierarchy, there is also a column indicating whether the static member is a foster parent. Note that each level can have one foster parent at most.

   For more information, see "Create Hierarchy Levels and Define the Attributes" (p. 92), "Create an Auto-level Hierarchy and Define the Attributes" (p. 97), "Create a Lookup and Define the Attributes" (p. 99).

3. Click Add, and type a value for each attribute.

4. If the static member is to be a foster parent for orphaned members of the level below, select the check box in the Foster column and type a name for the foster parent.

   Note: You can either create a static member specifically to be the foster parent, or you can use an existing static member.

   Tip: To change the value for an attribute, click the attribute and then click Edit.

5. Click OK.

Foster Parents

In a typical hierarchy, each level contains a set of members. The members of each level, except the highest, are related to a parent member from the level above. If a member has no explicit parent, it cannot attach to the hierarchy. A foster parent is an artificially introduced member that acts as a parent for members that either have no defined parent, or whose defined parent cannot be found at the next highest level.

By default, IBM® Cognos® Data Manager creates automatic foster parents at all but the lowest level. At each level, you can create one static member to act as a foster parent for orphaned members from the level below. Using a static member as the foster parent overrides automatic fostering.

Fostering means that the hierarchy can be represented even when levels are missing for some records. This ensures that aggregations take into account all child members at a level when producing summary data.

For example, consider this hierarchy.
The data includes a member named Birmingham, for which no explicit parent or foster parent exists. Data Manager creates a foster parent using the name of the next-highest level, and prefixes it with a question mark. It adds a foster parent named ?COUNTRY, under which it places the member named Birmingham.

Fostering most commonly occurs when creating a dimension from operational source data. If possible, any fostered members should be examined and assigned to legitimate parents in the operational database.

You can change the default name, and you can disable fostering.

**Specify a Name for a Foster Parent**

You can specify a name for a foster parent rather than use the default name.

IBM® Cognos® Data Manager names foster parents by prefixing the level name with a question mark. For example, the foster parent that resides in the LOCATION level is named ?LOCATION. However, if the hierarchy is entirely numeric, Data Manager names the foster parent by prefixing the level number with a minus sign (-).

You specify a name for a foster parent by defining a static member as the foster parent and entering a suitable name. For more information, see "Add a Static Member" (p. 103).

You cannot specify a name for an automatic foster parent.

**Disable Fostering**

By default, IBM® Cognos® Data Manager automatically provides a foster parent for each orphaned hierarchy member. However, you can disable automatic fostering.

**Steps**

1. Click the hierarchy or auto-level hierarchy for which you want to disable fostering.
2. From the Edit menu, click Properties.
3. Click the General tab.
4. Select the Disable fostering check box.
You must now instruct Data Manager to take appropriate action if it encounters orphaned members.

5. Click the **Features** tab.

6. In the **Fostering** box, click **Reject**, **Reject/Warn**, or **Error** as appropriate.
   
   For information, see "Foster Orphaned Members" (p. 106).

7. Click **OK**.

### Null ID Values

IBM® Cognos® Data Manager ignores ID columns that have a null value. This is because null IDs are usually an indication that the data is dirty, or that the wrong column has been specified as the ID.

### Set Error Handling

You can specify the action that IBM® Cognos® Data Manager should take when it encounters particular situations. For each situation, you can choose one of the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts the item without warning (default)</td>
</tr>
<tr>
<td>Warn</td>
<td>Writes a message to the log file</td>
</tr>
<tr>
<td>Reject</td>
<td>Rejects the item without warning</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Rejects the item and writes a message to the log file</td>
</tr>
<tr>
<td>Error</td>
<td>Writes an error message to the log file and stops executing the build at the current item</td>
</tr>
</tbody>
</table>

By default, when you select Warn or Reject/Warn, each occurrence of the selected situation is written to the log file. However, you can specify a limit to the number of messages written.

Data Manager always writes a message to indicate that a situation has occurred, even if individual occurrences are not written.

### Steps

1. Click the reference structure for which you want to set error handling.

2. From the **Edit** menu, click **Properties**.
   
   The **Properties** window for the selected reference structure appears.

3. Click the **Features** tab.
4. In the **Actions** box adjacent to the feature, click the required option.

5. If you set the feature to either **Warn** or **Reject/Warn**, and you want to restrict the number of warning messages issued, select the **Limit** check box for the feature. Then, in the adjacent box, type the maximum number of warnings.

   When the number of warnings reaches this limit, Data Manager continues to take the required action. However, it suppresses further warning messages to the log file.

   **Note:** If you type a limit of zero, no warnings are written to the log file, and you have no indication of why a record was rejected.

6. Click **OK**.

### Foster Orphaned Members

For information about fostering, see "Foster Parents" (p. 103).

If you disable fostering, you must configure IBM® Cognos® Data Manager to take appropriate action if it encounters orphaned hierarchy members.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts the orphaned members</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts the orphaned members, and writes a message to the log file stating that foster members were encountered</td>
</tr>
<tr>
<td>Reject</td>
<td>Rejects all orphaned members</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Rejects all orphaned members, and writes a message to the log file stating that foster members were rejected</td>
</tr>
<tr>
<td>Error</td>
<td>Issues an error message and stops executing the build if an orphaned member is encountered</td>
</tr>
</tbody>
</table>

### Applicable Reference Structures

- Hierarchies and auto-level hierarchies

### Multiple Parents

If IDs are not unique, more than one data source row may contribute to a particular hierarchy member. If those rows refer to different parents, the hierarchy member has multiple parents.

If multiple data source rows contain the same ID for a hierarchy level, each data source row may refer to a different parent. IBM® Cognos® Data Manager merges the non-unique data source rows to create a single member with a single set of member attributes. However, the single member may be delivered as multiple rows to the target dimension.
By default, one row is delivered for each combination of ID and parent ID. You can configure Data Manager to allow or disallow this depending on the option that you set for multiple parents.

If you allow multiple parents, each delivered row has the same business ID and the same surrogate key but different parent IDs.

Support for multiple parents requires each fact row to potentially reference multiple dimension rows, which is a many-to-many relationship.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts multiple parents</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts multiple parents, and writes a message to the log file stating that multiple parents were encountered</td>
</tr>
<tr>
<td>Reject</td>
<td>Accepts the first parent that is encountered and rejects source data rows that refer to other parents</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Accepts the first parent that is encountered, rejects source data rows that refer to other parents, and writes a message to the log file stating that multiple parents were rejected</td>
</tr>
<tr>
<td>Error</td>
<td>Issues an error message and stops executing the build</td>
</tr>
</tbody>
</table>

**Applicable Reference Structures**

Hierarchies and auto-level hierarchies

**Non-Unique IDs**

The same ID can never exist within a hierarchy level. However, the same ID can exist at different levels within the same hierarchy.

If multiple data source rows contain the same ID within a hierarchy level, IBM® Cognos® Data Manager merges the source data to give a single member for that ID. Multiple data source rows with the same ID within a single level may also point to different parents, which may lead to complications. For more information, see Multiple Parents.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Allows the same ID to exist at different hierarchy levels</td>
</tr>
<tr>
<td>Warn</td>
<td>Allows the same ID to exist at different hierarchy levels, and writes a message to the log file</td>
</tr>
<tr>
<td>Reject</td>
<td>Accepts the first ID at any hierarchy level and rejects the remainder</td>
</tr>
</tbody>
</table>
### Applicable Reference Structures

Hierarchies and auto-level hierarchies

### Duplicate Rows at a Level

If two source data rows have the same ID and the same parent, they are considered duplicates. For example, in this table, because both items with the ID value of p010 have the same parent, they are considered duplicate rows.

<table>
<thead>
<tr>
<th>ID</th>
<th>Caption</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>p001</td>
<td>Camping equipment</td>
<td></td>
</tr>
<tr>
<td>p010</td>
<td>Lights</td>
<td>p001</td>
</tr>
<tr>
<td>p010</td>
<td>Lanterns</td>
<td>p001</td>
</tr>
</tbody>
</table>

Valid configuration are options are shown below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Assigns to each member the last caption that is accepted from the DataStream. In the example table, the caption Lantern is assigned to the member with ID p010.</td>
</tr>
<tr>
<td>Warn</td>
<td>Assigns to each member the last caption that is accepted from the DataStream, and writes a message to the log file. In the example table, the caption Lantern is assigned to the member with ID p010.</td>
</tr>
<tr>
<td>Reject</td>
<td>Accepts the first instance of each ID and parent pairing and rejects subsequent instances. The first applicable caption is assigned to each member. In the example table, the caption Lights is assigned to the member with ID p010.</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Accepts the first instance of each ID and parent pairing, rejects subsequent instances, and writes a message to the log file. The first applicable caption is assigned to each member. In the example table, the caption Lights is assigned to the member with ID p010.</td>
</tr>
</tbody>
</table>
Applicable Reference Structures
All reference structures

Null/Empty Captions
You can specify an attribute with the $CAPTION property for all reference features.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts the null/empty captions</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts the null/empty captions and writes a message to the log file stating that they were encountered</td>
</tr>
<tr>
<td>Reject</td>
<td>Rejects any data that has a null or empty value</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Rejects any data that has a null or empty value and writes a message to the log file</td>
</tr>
<tr>
<td>Error</td>
<td>Issues an error message and stops executing the build if null or empty values are encountered</td>
</tr>
</tbody>
</table>

Applicable Reference Structures
All reference structures

Non-Unique Captions
Non-unique captions may cause an internal error in some reporting tools. You can configure IBM® Cognos® Data Manager to take appropriate action if this occurs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts the non-unique captions</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts the non-unique captions and writes a message to the log file stating that they were encountered</td>
</tr>
<tr>
<td>Reject</td>
<td>Accepts the first source data row and rejects subsequent rows</td>
</tr>
</tbody>
</table>
### Applicable Reference Structures

All reference structures

### Unbalanced Hierarchy

IBM® Cognos® Data Manager cannot determine whether a hierarchy is unbalanced until it acquires all source data.

For information about unbalanced hierarchies, see "Hierarchy Types" (p. 121).

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts that the hierarchy is unbalanced</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts that the hierarchy is unbalanced and writes a message to the log file</td>
</tr>
<tr>
<td>Reject</td>
<td>Not available</td>
</tr>
<tr>
<td>Reject/Warn</td>
<td>Not available</td>
</tr>
<tr>
<td>Error</td>
<td>Issues an error message and stops executing the build if a unbalanced hierarchy is encountered</td>
</tr>
</tbody>
</table>

### Applicable Reference Structures

Hierarchies and auto-level hierarchies

### Circular References

Circular references occur when a hierarchy member is its own ancestor. For example, if member A is the parent of member B, member B is the parent of member C, and member C is the parent of member A, then each member (A, B, and C) is its own ancestor.

IBM® Cognos® Data Manager detects circular references by examining each hierarchy member in sorted order and following the child-parent references. For each member, Data Manager stops following the child-parent links when it either reaches the ultimate ancestor or proves that a circular reference exists. If a circular reference exists, the last member that was examined is attached to the foster hierarchy.

For example, in the following table, BUF is the first item in the sort order.
<table>
<thead>
<tr>
<th>ID</th>
<th>Caption</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUF</td>
<td>Buffalo</td>
<td>NY</td>
</tr>
<tr>
<td>EAST</td>
<td>Eastern Region</td>
<td>BUF</td>
</tr>
<tr>
<td>NY</td>
<td>New York</td>
<td>EAST</td>
</tr>
</tbody>
</table>

The following code is the result:

\[
{\text{?AUTO}} \rightarrow \text{[Unknown AUTO]}
\]

\[
{\text{?LEVEL2}} \rightarrow \text{[Unknown LEVEL2]}
\]

\[
{\text{?LEVEL3}} \rightarrow \text{[Unknown LEVEL3]}
\]

\[
{\text{EAST}} \rightarrow \text{[Eastern Region]}
\]

\[
{\text{NY}} \rightarrow \text{[New York]}
\]

\[
{\text{BUF}} \rightarrow \text{[Buffalo]}
\]

In the following table, EAST is the first item in the sort order.

<table>
<thead>
<tr>
<th>ID</th>
<th>Caption</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX</td>
<td>Buffalo</td>
<td>NY</td>
</tr>
<tr>
<td>EAST</td>
<td>Eastern Region</td>
<td>XXX</td>
</tr>
<tr>
<td>NY</td>
<td>New York</td>
<td>EAST</td>
</tr>
</tbody>
</table>

The following code is the result:

\[
{\text{?AUTO}} \rightarrow \text{[Unknown AUTO]}
\]

\[
{\text{?LEVEL2}} \rightarrow \text{[Unknown LEVEL2]}
\]

\[
{\text{?LEVEL3}} \rightarrow \text{[Unknown LEVEL3]}
\]

\[
{\text{NY}} \rightarrow \text{[New York]}
\]

\[
{\text{XXX}} \rightarrow \text{[Buffalo]}
\]

\[
{\text{EAST}} \rightarrow \text{[Eastern Region]}
\]

Valid configuration are options are shown below.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Accepts the circular references. When circular references are being accepted, you must not disable automatic fostering. This is because Data Manager attaches to the foster hierarchy those hierarchy branches that exhibit circular references.</td>
</tr>
<tr>
<td>Warn</td>
<td>Accepts the circular references, and writes a message to the log file stating that circular references were encountered. When circular references are being accepted, you must not disable automatic fostering. This is because Data Manager attaches to the foster hierarchy those hierarchy branches that exhibit circular references.</td>
</tr>
</tbody>
</table>
### Applicable Reference Structures

Auto-level hierarchies

### Create a Date Hierarchy

If the structure data does not contain a master table of dates, you can create a static hierarchy of dates. You can include levels for years, quarters, months, weeks, and days.

For most date hierarchy levels, the boundary of each member coincides with a boundary of one of its children. For example, the start and end of each quarter is at the start and end of a month. However, the beginning or end of a month may occur in the middle of a week.

You can choose how to consolidate weeks that go across month boundaries:

- **Weeks roll to parent start**

- **Weeks roll to parent end**

- **Weeks roll to parent start and end**
Before you create a date hierarchy, ensure that the reference dimension to which you want to add the hierarchy represents time.

**Tip:** To check, click the reference dimension, and from the **Edit** menu, click **Properties**. If necessary, select the **Dimension represents time** check box, and click **OK**.

**Steps**

1. From the **Tools** menu, click **Date Hierarchy Wizard**.
2. Follow the instructions.

**Separate Reference Data and Fact Data**

It is highly preferable to completely separate reference data and fact data. You can then use a reference structure based on reference data in a number of builds that process fact data. However, this is not always possible.

Sometimes the fact data contains some or all of the reference data for a dimension. For example, suppose you have fact data representing customer transactions, such as goods purchased. The customer dimension in the fact data need only hold the unique customer reference number. A customer hierarchy that holds the same reference numbers, but from reference data, could be associated with the dimension. However, suppose it is the practice for the reference data to contain customer telephone numbers, but for updates to the telephone numbers to appear in the fact data (perhaps because the customer provided the update as part of making their transaction). Now the fact data contains elements of reference data.

In building the hierarchy, IBM® Cognos® Data Manager could read both the reference data and the fact data. The disadvantage is that Data Manager reads the fact data twice, once in the context of the reference data, and once as fact data. The alternative is to use Data Manager functionality that maps elements of the fact data back to the reference data.
A more extreme example is a fact table that is denormalized. That is, it contains not only the member ID for the fact data, but also the member IDs for the other levels in the same dimension. For example, fact data represents monthly sales, and necessarily has a Month column, but also has Quarter and Year columns. The Month column should be treated as the dimension, and the Quarter and Year columns treated as attributes and probably marked *Never Output*. You should then map the Quarter and Year columns back in to the Quarter and Year levels of the hierarchy associated with the month dimension.

A common example of this is where the fact data relates to transactions for customers. The fact data would contain the customer reference as well as a transaction reference (that is, an order number). The reference data will only have information about previous orders that customers placed with new orders appearing in the fact data.

**Note:** There is little value in performing updates to the hierarchy in this way, unless the hierarchy itself is delivered as a dimension delivery, because the updates are lost at the end of the build. The dimension delivery should deliver data to the same table from which the hierarchy is built to so that the next time the hierarchy is used, it will contain all the updates made by previous builds.

When this technique is used to add new members to a hierarchy level, the domain size of the hierarchy increases throughout the build. This may mean that a domain size estimated for the hierarchy at the start of the build does not represent a good estimate and may cause the build to fail. In this case, a domain size should be set for the dimension in question.

If you map a transformation model element directly to a hierarchy, the transaction data contributes hierarchy members. Therefore, there can be no unmatched members of that hierarchy.

For more information, see "Example Scenarios" (p. 114).

**Example Scenarios**

**Private Data Mart**

A private data mart typically provides a snapshot view of the data. It has one or more private dimension tables and one or more fact tables. The following illustration shows the production of a private data mart.
IBM® Cognos® Data Manager first builds the hierarchy in memory. As it acquires the transaction data, Data Manager maps this to the hierarchy and transformation model, and performs the transformations that you specify. Finally, Data Manager delivers the fact and dimension data to the data mart.

Because the dimension data is not shared with other data marts, you can acquire structure and transaction data, transform that data, and deliver both the fact and dimension data in one object. Therefore, a fact build that has both fact deliveries and dimension deliveries is a good solution for this scenario. For private data marts that provide a snapshot view, it is usual to configure both fact deliveries and dimension deliveries to fully refresh the target tables.

Private data marts require one hierarchy for each business dimension to be modelled.

**Conformed Marts Without Surrogate Keys**

Conformed marts differ from private marts in that many marts share the delivered dimension data. Therefore, the delivery of dimension data is not the responsibility of any one fact build. The following are strategies for delivery of dimension data:

- All the fact builds have identical dimension deliveries, and the dimension data is delivered to the data mart each time you execute a fact build.

- None of the fact builds have dimension deliveries. Instead, a dimension build delivers the dimension data, and the fact builds deliver only fact data. In this case, it is usual to combine the dimension build and fact builds in a JobStream.

The following illustration shows the latter case.

[Diagram showing two stages of data delivery: First stage for structure data and dimension builds, Second stage for fact builds and data mart]

This approach populates the data mart in two stages. In one stage, IBM® Cognos® Data Manager creates each hierarchy in memory and delivers the corresponding dimension data through the corresponding dimension build. Each business dimension requires one hierarchy and one dimension build. In the second stage, Data Manager delivers the fact tables through the fact builds.

Because both the dimension builds and fact builds refer to the same hierarchies, only one hierarchy is required for each business dimension.
Conformed Marts With Surrogate Keys

Surrogate keys are numeric values that replace business keys in the data mart. They exist only in the data mart. The following illustration shows the transformation of a master table and related transaction table from an online transaction processing (OLTP) system to dimension and fact tables in a data mart.

Note: The business key becomes an attribute of the dimension table, and the surrogate becomes the primary key. The fact table does not contain the business keys of the transaction system, and the surrogate key replaces the business key.

Because the OLTP system contains no information about the surrogate keys that reside in the data mart, this information is obtained from the dimension tables in the data mart. However, these tables must first be populated.

The requirement to populate the dimension tables before processing the fact data implies the following stages of execution.

This approach has these characteristics:

- Two reference structures are available for each dimension.
- A primary reference structure is constructed from the structure data. It does not contain the attributes to support surrogate keys.
- A dimension build delivers the primary reference structure to the data mart. A dimension table template defines the semantics of the dimension table and delivers the surrogate keys to the data mart.
A secondary reference structure is constructed from the dimension table in the data mart. It uses the template used to deliver the dimension table. Therefore, IBM® Cognos® Data Manager can access the surrogate key values, and the fact build can deliver these values to the fact table.

It is common to combine the dimension builds and dependent fact builds in a JobStream. For information about JobStreams, see "Managing Builds Using JobStreams" (p. 255).

**Explore and Test a Reference Structure**

You can use the reference explorer to explore and test a reference structure. When you view a reference structure in the reference explorer, IBM® Cognos® Data Manager acquires the structure data and organizes it according to the specification of the reference structure.

You can also use the Find menu to search the data to locate a member by name.

**Steps**

1. From the Tools menu, click **Reference Explorer**.
   - If you have made changes to the catalog, a message appears where you choose whether you want to save the changes.
   - The Reference Explorer dialog box appears.

2. In the Dimension box, click the reference dimension in which the reference structure resides.

3. In the Reference item box, click the reference structure to explore.

4. If you want to limit the depth to which the reference explorer processes the reference structure, in the Min depth and Max depth boxes, click the minimum and maximum depth (level) respectively.

5. If you do not want the reference explorer to show foster parents that have no child members, select the Remove unused foster parents check box.

6. Click **OK**.
   - If substitution variables are included in any of the SQL statements for the reference structure you are testing, the Values for Expression window appears listing each substitution variable.

7. In the Value column, type a suitable value for each variable.
   - **Tip**: Click **Set Default Values** to enter the default value that you specified when you defined the substitution variable.
   - For more information on default values, variable data type and scope, see "Testing Expressions and Scripts" (p. 341).

8. Click **OK**.
   - The Reference Explorer window appears showing the data structure for the selected reference structure. Note that this process may take some time, particularly if there are errors in the definitions. The status messages appear at the bottom of the window as the reference structure appears.
Tips

- To interrupt processing at any time, from the View menu, click **Interrupt**.
- To refresh the information or to view another reference item, from the View menu, click **Refresh**. The **Reference Explorer** dialog box appears.

The Reference Explorer Window

The information in the reference explorer is shown in two panes. The Elements pane shows either the hierarchical relationship view or the level elements view. The Attributes pane shows detail information.

The Elements Pane

The Hierarchy Relationship View

The hierarchy relationships view in the Elements pane, shows the structure data as a true hierarchical structure with members linked to their parent.

![Hierarchy Relationship View](image)

**Tip:** To select this view, from the View menu, click **Hierarchical Relationships**.

The Elements View

The elements view shows the hierarchical members linked under the levels to which they belong.

Tip: To select this view, from the View menu, click **Elements At Each Level**. The elements view looks something like this:
The Attributes Pane

The Attributes pane shows the attributes that are relevant to the item you select in the Elements pane.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>Staff</td>
<td></td>
</tr>
<tr>
<td>No. Parents</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>No. Children</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Id</td>
<td>35</td>
<td>SelectStaffCode</td>
</tr>
<tr>
<td>Option</td>
<td>Edward Lopez</td>
<td>SelectStaffName</td>
</tr>
<tr>
<td>Attribute</td>
<td>1992-08-12</td>
<td>DateHead</td>
</tr>
<tr>
<td>Attribute</td>
<td>Branch Manager</td>
<td>Position</td>
</tr>
<tr>
<td>Parent 1</td>
<td>25</td>
<td>SelectBranchName</td>
</tr>
</tbody>
</table>

Tip: You can hide the Attributes pane by clicking the Show/hide attributes pane button. To reveal the pane, click the button again.

Delete a Reference Structure

You cannot delete a reference structure if another component from the same catalog is using it.

For information about component dependencies, see "Locate Components and Their Dependencies" (p. 38).

Steps

1. Click the reference structure that you want to delete.
2. From the Edit menu, click Delete.
3. In the message box, click Yes.
Chapter 12: Constructing and Transforming Hierarchies

You can construct different types of hierarchy using IBM® Cognos® Data Manager, depending on the structure of the data that you are working with. You can use Data Manager to work with balanced hierarchies, unbalanced hierarchies, and ragged hierarchies.

It is more difficult to work with unbalanced and ragged hierarchies than balanced hierarchies because the number of levels can vary from one branch to another. There are some techniques you can use to construct and deliver data for these hierarchy types so that it is easy to work with the data using IBM® Cognos® Business Intelligence. You can

- Balance a hierarchy.
- Link a new level to the bottom of an unbalanced hierarchy.
- Flatten a hierarchy.

Hierarchy Types

Balanced Hierarchies

In a balanced hierarchy, each branch of the tree contains the same number of levels, and the parent of every member comes from the level immediately above. This is an example of a balanced hierarchy.

Unbalanced Hierarchies

In an unbalanced hierarchy, each branch of the tree can descend to a different level, so it has leaf nodes at more than one level. The parent of every member comes from the level immediately above. In the following example, state information is available only for the USA. Mexico and Canada are leaf nodes.
Ragged Hierarchies

In a ragged hierarchy, the logical parent of at least one member does not come from the level immediately above, but a level higher up. In the following example, city information for Washington DC relates to the country level, not the state level.

Balance a Hierarchy

Consider this unbalanced hierarchy.

If you do not balance the hierarchy, only the member LOS is detected at the bottom of the hierarchy. If you want to accurately detect further valid members, you must balance the hierarchy. You can balance a hierarchy at one or more specified levels.

When you balance a hierarchy at a specified level, IBM® Cognos® Data Manager searches for all leaf nodes at the level directly above and duplicates them at the level you are balancing. Duplicate nodes share the ID and caption of the original node.

If you balance the previous hierarchy at the City level, it looks like this.
Note: Duplicate nodes, created by Data Manager, are shown in red.

In addition to balancing, you can perform dense balancing. This differs from balancing in that Data Manager duplicates every node from the level directly above, not just leaf nodes.

If you perform dense balancing on the previous hierarchy at the City level, it looks like this.

Note: Duplicate nodes that Data Manager creates as a result of performing dense balancing are shown in green.

In the following example, the State and City levels have been balanced.

You can perform balancing on all hierarchy types. Data Manager applies balancing to a hierarchy only after it has constructed it.
You can also balance an auto-level hierarchy, but this is not level-based balancing. Data Manager balances the whole hierarchy, from the highest level to the lowest level it can detect.

**Steps to Balance a Hierarchy an Auto-Level Hierarchy**

1. Click the hierarchy level you want to balance.
2. From the **Edit** menu, click **Properties**.
   The appropriate **Properties** window appears.
3. Click the **General** tab.
4. Select the **Balance level** check box.
5. If you also want to perform dense balancing, select the **Dense** option.
6. Click **OK**.
   You can now test the hierarchy by exploring it. For more information, see "Explore and Test a Reference Structure" (p. 117).

**Steps to Balance a Hierarchy With Recursive Levels**

1. Click the hierarchy you want to balance.
2. From the **Edit** menu, click **Properties**.
   The **Recursive Level Properties** window appears.
3. If you want to balance the top level of the recursive hierarchy, click the **General** tab.
4. Select the **Balance level** check box. If you also want to perform dense balancing at the top level, select the **Dense** option.
5. If you want to balance a dependent of the recursive hierarchy, click the **Sub-Levels** tab.
6. Select the dependent level you want to balance and then click the **Edit Details**.
   The **More Details** dialog box appears.
7. Select the **Balance level** check box. If you also want to perform dense balancing at the top level, select the **Dense** option.
8. Click **OK** to close the **More Details** dialog box.
9. Click **OK**.
   You can now test the hierarchy by exploring it. For more information, see "Explore and Test a Reference Structure" (p. 117).

**Link a New Level to the Bottom of an Unbalanced Hierarchy**

Sometimes, data from a separate source needs to be linked to the bottom of a hierarchy. By default, members are only added if the parent exists at the level immediately above. For example, suppose you have an unbalanced hierarchy like this.
If you were to add a Store level to this hierarchy, only stores linked directly to LOS are added by default. Suppose the data for the Store level is as follows.

<table>
<thead>
<tr>
<th>Store ID</th>
<th>Store Name</th>
<th>Parent ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Shop A</td>
<td>CAN</td>
</tr>
<tr>
<td>102</td>
<td>Shop B</td>
<td>ONT</td>
</tr>
<tr>
<td>103</td>
<td>Big Store A</td>
<td>LOS</td>
</tr>
<tr>
<td>104</td>
<td>Shop D</td>
<td>OTT</td>
</tr>
<tr>
<td>105</td>
<td>Big Store C</td>
<td>USA</td>
</tr>
</tbody>
</table>

When you link the Store level, only member 103 is correctly added to the bottom of the hierarchy. All the other stores are fostered.

IBM® Cognos® Data Manager lets you resolve this problem by providing the option to search for parent members further up the hierarchy. This technique allows you to transform a ragged hierarchy.

**Note:** You cannot use balancing to resolve this problem because balancing only occurs after hierarchy construction.

This option is named Link parent at next available level. For each node whose parent is found at a higher level of the hierarchy, Data Manager creates duplicate nodes at every intermediate level to link the data to the hierarchy. If any of the hierarchy data contains links to nodes that do not exist in the hierarchy, Data Manager still fosters it. Duplicate nodes share the ID and caption of the original nodes.

Using the example above, after the store data has been linked to the hierarchy, using the Link parent at next available level option, the hierarchy looks like this.
Duplicate nodes, created by Data Manager, are shown in red.

Notes
- Data Manager does not create any duplicates of the node MEX because there are no stores linked to it.
- Store 104 is still fostered because its parent node, OTT, does not exist in the hierarchy.

You can set up the Link parent at next available level option for all hierarchy types, except auto-level hierarchies.

Steps
1. Click the hierarchy level you want to link.
2. From the Edit menu, click Properties.
   The Level Properties window appears.
3. Click the General tab.
4. Select the Link parent at next available level check box.
5. Click OK.
   You can now test the hierarchy by exploring it. For more information, see "Explore and Test a Reference Structure" (p. 117).

Flatten a Hierarchy
If the source data for a hierarchy contains parent-child relationships, you can flatten the structure so it can be used for reporting. When you flatten a hierarchy, IBM® Cognos® Data Manager creates a set of records, with each record containing the details of a single branch of a tree.

You flatten a hierarchy by creating a hierarchy with recursive levels, which allows you to explicitly name the levels contained in the recursive relationships. The hierarchy can contain a mixture of
normal and recursive levels as required, so the recursive levels can be positioned at any level within
the hierarchy.

**Note:** You can also create an auto-level hierarchy using the source data and flatten it using staging
builds, but this requires an intermediate step.

For more information about creating a hierarchy with recursive levels, see "Create Recursive Lev-
els" (p. 93).

### Example

Suppose you have the following geographical data in a parent-child structure.

<table>
<thead>
<tr>
<th>Location ID</th>
<th>Parent ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAM</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>MEX</td>
<td>NAM</td>
</tr>
<tr>
<td>CAN</td>
<td>NAM</td>
</tr>
<tr>
<td>USA</td>
<td>NAM</td>
</tr>
<tr>
<td>ONT</td>
<td>CAN</td>
</tr>
<tr>
<td>CAL</td>
<td>USA</td>
</tr>
<tr>
<td>LOS</td>
<td>CAL</td>
</tr>
</tbody>
</table>

You want to flatten the data so that it looks like this.

<table>
<thead>
<tr>
<th>Continent ID</th>
<th>Country ID</th>
<th>State</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAM</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>NAM</td>
<td>USA</td>
<td>CAL</td>
<td>LOS</td>
</tr>
<tr>
<td>NAM</td>
<td>CAN</td>
<td>ONT</td>
<td>&lt;null&gt;</td>
</tr>
<tr>
<td>NAM</td>
<td>MEX</td>
<td>&lt;null&gt;</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

To do this, you must set up a hierarchy with recursive levels for Continent, Country, State/Province,
and City.

The following diagram illustrates this structure set up as a hierarchy with recursive levels in
IBM® Cognos® Data Manager.
Now you want to link a new level of data, called Store, to this hierarchy. The Store data to be linked is as follows.

<table>
<thead>
<tr>
<th>Store ID</th>
<th>Store Name</th>
<th>Parent ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Shop A</td>
<td>CAN</td>
</tr>
<tr>
<td>102</td>
<td>Shop B</td>
<td>ONT</td>
</tr>
<tr>
<td>103</td>
<td>Big Store A</td>
<td>LOS</td>
</tr>
<tr>
<td>105</td>
<td>Big Store C</td>
<td>USA</td>
</tr>
</tbody>
</table>

Because the Store level data is separate from the geographical data, you need set up another hierarchy level, Store, at the bottom of the North America hierarchy.

If you do not select the Link parent at next available level option for the Store level, only store 103 would be linked to the hierarchy, because LOS, the parent of store 103, is the only hierarchy member that exists at the lowest level of the hierarchy. The remaining stores would be fostered.

To resolve this issue, ensure the Link parent at next available level option is selected for the Store level, as illustrated here.

If you deliver this hierarchy using a dimension build, the data might look like this.
Note: If you exclude partially populated rows, Data Manager does not deliver the first record. For more information about excluding partially populated rows, see "Add a Table to a Dimension Build" (p. 236).

Now suppose you want to deliver fact data with reference to this hierarchy. The fact data to be delivered is as follows.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Product Name</th>
<th>Location ID</th>
<th>Monthly Sales Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product A</td>
<td>101</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>Product B</td>
<td>103</td>
<td>7000</td>
</tr>
<tr>
<td>3</td>
<td>Product C</td>
<td>MEX</td>
<td>100</td>
</tr>
</tbody>
</table>

If you want Data Manager to accurately deliver all three records, you must balance the hierarchy at the State/Province, City, and Store levels to ensure that MEX is duplicated at the bottom level of the hierarchy, as shown here.

<table>
<thead>
<tr>
<th>Continent ID</th>
<th>Country ID</th>
<th>State/Province ID</th>
<th>City ID</th>
<th>Store ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAM</td>
<td>MEX</td>
<td>MEX</td>
<td>MEX</td>
<td>MEX</td>
</tr>
<tr>
<td>NAM</td>
<td>CAN</td>
<td>CAN</td>
<td>CAN</td>
<td>101</td>
</tr>
<tr>
<td>NAM</td>
<td>CAN</td>
<td>ONT</td>
<td>ONT</td>
<td>102</td>
</tr>
<tr>
<td>NAM</td>
<td>USA</td>
<td>CAL</td>
<td>LOS</td>
<td>103</td>
</tr>
<tr>
<td>NAM</td>
<td>USA</td>
<td>USA</td>
<td>USA</td>
<td>105</td>
</tr>
</tbody>
</table>

If you do not balance the hierarchy, only records 1 and 2 are delivered, because stores 101 and 103 exist at the bottom level of the hierarchy. Record 3 is rejected.
The following diagram illustrates this hierarchy in Data Manager, where the Balance level option is selected for the State/Province, City, and Store levels.

![Diagram of hierarchy]

If you deliver the fact data with reference to this hierarchy, it looks like this.

<table>
<thead>
<tr>
<th>Continent ID</th>
<th>Country ID</th>
<th>State ID</th>
<th>City ID</th>
<th>Store ID</th>
<th>Product ID</th>
<th>Monthly Sales Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAM</td>
<td>CAN</td>
<td>CAN</td>
<td>CAN</td>
<td>101</td>
<td>1</td>
<td>1500</td>
</tr>
<tr>
<td>NAM</td>
<td>USA</td>
<td>CAL</td>
<td>LOS</td>
<td>103</td>
<td>2</td>
<td>7000</td>
</tr>
<tr>
<td>NAM</td>
<td>MEX</td>
<td>MEX</td>
<td>MEX</td>
<td>MEX</td>
<td>3</td>
<td>100</td>
</tr>
</tbody>
</table>
Chapter 13: Fact Builds

A fact build acquires transactional data from the data source, transforms the data with reference to the dimensional framework, and delivers the data to target data marts. Optionally, a fact build can also deliver dimension data.

Note: Data can also be transformed independently of the dimensional framework. This might be required when setting up one or more staging builds to cleanse or standardize data sources prior to merging them in a final fact build.

View Fact Builds

You can view fact builds in the Tree pane or the Visualization pane.

View Fact Builds in the Tree Pane

A fact build consists of several types of objects. These objects are shown grouped together in the Tree pane. Each fact build in the Builds and JobStreams folder is represented as follows.

Tip: You can view just the fact builds contained in a catalog by clicking the Fact Builds tab at the bottom of the Tree pane.

View Fact Builds in the Visualization Pane

When you click a fact build in the Tree pane, four different views of the selected fact build are shown, each on a different tab in the Visualization pane.

For information about each of the symbols used in the Visualization pane, see "Buttons and Icons" (p. 549).

The <Fact Build Name> Tab

The <fact build name> tab visually represents the whole fact build. The name of this tab is the name of the selected fact build.
Chapter 13: Fact Builds

The tab shows the processing that occurs from data acquisition to delivery of the transformed data. At the top are the reference structures associated with the fact build. Below the reference structures, from left to right, are the processes in sequential order.

![Diagram]

**The Mapping Tab**

The Mapping tab visually represents the mapping between the data source columns on the left, the DataStream items in the middle, and the transformation model on the right.

When you click a data source column, DataStream item, or a transformation model element, the mapping from the data source through to the transformation model element is highlighted.

Unmapped items are indicated by a change of background color.

![Diagram]

**The Transformation Model Tab**

The Transformation Model tab shows
- each element in the transformation model
- the reference structure to which each dimension element relates
- the hierarchy levels at which IBM® Cognos® Data Manager is to access (input) data
- the hierarchy levels at which Data Manager is to deliver (output) data

For more information about the transformation model, see "Transformation Models" (p. 163).
The Fact Delivery Tab

The Fact Delivery tab shows all the elements in the transformation model and, for each element, the column name to which each element is to be delivered.

When you click an item, the mapping between the transformation model element and the fact delivery column is highlighted. Unmapped items are indicated by a change of background color.

For more information about fact deliveries, see "Delivering Fact Data Using a Fact Build" (p. 207).

The Fact Build Wizard

You can use the Fact Build wizard to help you to create a fact build that delivers the type of data that you specify.

If you want to deliver multiple tables to different databases, or partition data to deliver to different tables, you cannot use the Fact Build wizard, you must add the deliveries manually. For information, see "Delivering Fact Data Using a Fact Build" (p. 207).

Each of these standard fact build types provide default settings for various deliveries:

- **IBM® Cognos® BI Mart (Star)**
- **IBM® Cognos® BI Mart (Snowflake)**
- **Relational Datamart**
- **Data Transfer**

Where possible, the Fact Build wizard delivers surrogate keys for all the standard types of fact builds except Data Transfer, and performs no aggregation. It delivers the fact data to a single table. However, if you decide to perform aggregation along any dimension, the Fact Build wizard creates a fact delivery for each combination of hierarchy levels. For example, there are three dimensions (A, B, and C) and you decide to perform aggregation on dimension A, which has three levels (1, 2, and 3), then the Fact Build wizard creates fact deliveries for A.1*B*C, A.2*B*C, and A.3*B*C.
IBM Cognos BI Mart (Star) and IBM Cognos BI Mart (Snowflake)

An IBM® Cognos® BI Mart (Star) fact build delivers dimension data to tables that represent a star schema. That is, one dimension table is created for each dimension in the fact build.

An IBM® Cognos® BI Mart (Snowflake) fact build delivers dimension data to tables that represent a snowflake schema. That is, one dimension table is created for each level of each dimension in the fact build.

By default, an IBM® Cognos® BI Mart delivers fact data only at the lowest level of each hierarchy. If you choose to deliver aggregations, the Fact Build wizard configures the fact build to deliver them to satellite tables.

To use the Fact Build wizard to track attribute changes, you should select IBM Cognos BI Mart (Star). Alternatively, you can select IBM Cognos BI Mart (Snowflake), and then manually add the details of attributes to be tracked.

Relational Datamart

A relational datamart fact build provides the default settings for a fact build that delivers data into a relational database.

Use this type of fact build to deliver dimension data to tables that represent a star schema. That is, create one dimension table for each dimension in the fact build.

Data Transfer

A data transfer fact build provides the default settings for copying data from one DBMS to a single fact table in another DBMS.

By default, the Fact Build wizard creates all transformation model elements as attributes. It does not create dimension data.

Summary of Fact Build Types in the Fact Build Wizard

This table summarizes the default settings for each build type.

<table>
<thead>
<tr>
<th>Default</th>
<th>Star</th>
<th>Snow</th>
<th>Rel</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surrogate keys</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Aggregation</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Elements to suit data type</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>All elements as attributes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Star schema</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Snowflake schema</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Use the Fact Build Wizard

When you use the Fact Build wizard, you

- define the purpose of the build, which includes selecting the type of build to create and the connection into which the build is to deliver data
- create the DataStream
- assign the transformation model element types
- define the dimensions by associating each dimension element with a reference structure and specifying whether aggregation is to be performed, and define the measures and attributes if present
- define the fact delivery and its properties
- define the dimension delivery and its properties

When you create a fact build using the Fact Build wizard, you can amend it and add to it as you require.

Steps

1. From the Tools menu, click Fact Build Wizard.
2. Follow the instructions.

Automatic Source Table Joins Using the Fact Build Wizard

When you select more than one source table for a fact build, IBM Cognos Data Manager attempts to determine the join conditions for those tables. However, you can edit the SQL statement that the Fact Build wizard generates.

If Data Manager can detect the primary key of a table, it searches in the other tables for the name of each primary key column, and creates join conditions where it finds a match.

Where Data Manager cannot detect a primary key, it searches the tables for any match between the names of columns, and bases join conditions on any matches found.

For example, if a Customer table has a primary key named Customer.CustID and an Order table has a column named CustID, Data Manager detects the primary key. It matches the name of the primary key column with the column named CustID from the Order table, and includes Customer.CustID = Order.CustID in the WHERE clause of the SQL for the data source.

Create a Fact Build Manually

To create a fact build manually, you must
Add a Fact Build

This section describes how to add a fact build.

Steps
1. Click the Builds and JobStreams folder.
2. From the Insert menu, click Build, and then click Fact Build.
3. Click the General tab.
4. In the Name box, type a unique name for the fact build. If you require, type a business name and description.
5. Click OK.

This is the minimum information necessary to create a fact build. You are now ready to define the DataStream, the transformation model, and the deliveries.

For more information, see "DataStreams" (p. 141), "Transformation Models" (p. 163), "Delivering Fact Data Using a Fact Build" (p. 207), "Delivering Dimension Data Using a Fact Build" (p. 227).

Define the DataStream

After you create a fact build, you define the DataStream, which includes

- defining data sources so that IBM® Cognos® Data Manager knows from where to acquire the data
- creating DataStream items for each acquired data source column
- mapping data source columns to the DataStream items

For more information, see "DataStreams" (p. 141).

Define the Transformation Model

After you specify how the source data is to be acquired, you define the transformation model. The transformation model allows you to manipulate the data in a number of ways, including merging data from different sources, and partitioning and aggregating data.

Defining the transformation model includes

- creating transformation model elements for transformed data
mapping DataStream items to the transformation model elements to enable the flow of data

For more information, see "Transformation Models" (p. 163).

Add Deliveries to the Fact Build

You must specify the data repositories to which the data is to be delivered.

For more information, see "Delivering Fact Data Using a Fact Build" (p. 207) and "Delivering Dimension Data Using a Fact Build" (p. 227).

Execute the Fact Build

After you define the fact build, you execute it to acquire, transform, and deliver the data.

For more information, see "Executing Builds and JobStreams" (p. 273).

Rename a Fact Build

When you rename a fact build, it is treated as a new component for logging and auditing purposes. However, the components of the fact build are not renamed. Therefore, renaming the fact build does not change the names of components, target tables, and so on, that the Fact Build wizard derived from the original build name.

When renaming a fact build, remember that the names of objects in IBM® Cognos® Data Manager Designer are not case sensitive. For example, VendorDrillThru, VENDORDRILLTHRU, and vendordrillthru are considered to be identical.

Steps

1. Click the fact build you want to rename.
2. From the Edit menu, click Properties.
3. In the Name and Business name boxes, type the new name and business name.
4. Click OK.

Set the Date Source for Timestamping Data

When you execute a fact build, data is timestamped as follows:

- If the fact build includes dimension deliveries, dimension records that contain create date or update date attributes are timestamped.
- If the fact build is configured to add unmatched members to the dimension reference data, dimension records that contain create date, update date, or effective start date attributes are timestamped.
- If the fact build includes the DS_DATA_TIMESTAMP variable, for example in a derivation, a timestamp is added as required.
For more information about the create date, update date, and effective start date attributes, see "Behavior Types in Templates" (p. 80).

For more information about the DS_DATA_TIMESTAMP variable, see "DS_DATA_TIMESTAMP" (p. 329).

By default, data is timestamped using the current system date and time on the computer on which the build is executed. If you require, data can be timestamped using a different date source.

**Steps**
1. Click the required fact build.
2. From the Edit menu, click Properties.
3. Click the General tab.
4. In the Data timestamp box, choose the date source to be used for timestamping data:
   - Click Client system date/time to set the date to the current system date and time on the computer on which the build is executed. (default)
   - Click Variable to use a variable to set the date and, in the adjacent box, type the variable name.
     
     **Note:** The variable must be defined on the Variables tab. For information, see "Variables" (p. 319).
   - Click Date (yyyy-mm-dd hh:mm:ss) to set an explicit date and time and, in the adjacent box, type the date and time.
5. Click OK.

**Handle Duplicate Data**

Duplicate source data rows have identical dimension elements. You can specify the action to be taken if duplicate records are found.

**Note:** When aggregating data, duplicate keys in the input data can produce different results depending on how the duplicates are handled. For more information, see "Aggregate Duplicate Data" (p. 192).

**Steps**
1. Click the fact build.
2. From the Edit menu, click Properties.
3. Click the Input tab.
4. In Duplicate key handling, click one of the following options:
   - Allow records with duplicate keys for IBM® Cognos® Data Manager to accept the duplicated records. You should use this when duplicates are unimportant or when you know that duplicates will not occur. (default)
• **Reject records with duplicate keys** for Data Manager to accept the first data row and reject subsequent rows to the reject file. For information, see "Handling Data Rejected by a Fact Build" (p. 371).

• **Merge records with duplicate keys** for Data Manager to track the duplicated records and merge all dimension values using the merge functions that you specify. For information, see "Aggregate Duplicate Data" (p. 192).

• **Aggregate records with duplicate keys** to track the duplicated records and aggregate all dimension values using the aggregate function you specify. For information, see "Aggregate Duplicate Data" (p. 192).

**Note:** If you are aggregating data using aggregate rules, you cannot allow records with duplicate keys. You must reject, merge or aggregate them.

**Tip:** Merging or rejecting duplicates works well with breaking to minimize memory usage. For information, see "Dimension Breaks" (p. 378).

5. Click OK.

The build details are updated in the **Visualization** pane to graphically show how duplicates are to be handled. For information, see "Fact Build Visualization Icons" (p. 552).

### Delete a Fact Build

You can delete a fact build that is no longer required.

**Steps**

1. Click the required fact build.

2. From the **Edit** menu, click **Delete**.

3. In the message box, click **Yes**.
Chapter 14: DataStreams

Use DataStreams to access source data from a database. IBM® Cognos® Data Manager automatically adds an empty DataStream to each fact build and reference structure that you create. If you create a hierarchy, an additional DataStream is added to each level that you create. You can therefore use a DataStream to provide data for the whole hierarchy or for each hierarchy level.

Within a DataStream, you must define the necessary data sources that specify from where, and how, you will access data. A DataStream can contain as many data sources as you require.

After you define the data sources, you must create DataStream items to map the data source columns to the transformation model elements or reference structure level attributes.

You should use a DataStream to
- access data for a fact build
- access data from multiple data sources
- access structure data for multiple levels of a hierarchy
- perform derivations on source data
- add literal values

For reference structures, an alternative to using a DataStream to access data is to use a template. For information on when to use a DataStream or template to access data for a reference structure, see "Set Up Data Access" (p. 99).

You can apply a DataStream filter to remove unwanted data rows before they reach the transformation model. For information, see "Filter Source Data" (p. 159).

You can execute a DataStream to examine the source data prior to transformation. For information, see "Execute a DataStream" (p. 160).

View DataStreams

You can view DataStreams in the Tree pane or the Visualization pane.

Note: You can only view DataStreams for fact builds in the Visualization pane.

View DataStreams in the Tree Pane

In the Tree pane, each DataStream is represented as follows.
View Fact Build DataStreams in the Visualization Pane

To view the DataStream for a fact build, click the DataStream and then click the Mapping tab in the Visualization pane. This tab shows the mapping between the data source columns on the left, the DataStream items in the middle, and the transformation model on the right.

When you click on a data source column, a DataStream item, or a transformation model element, the mapping from the data source through to the transformation model element is highlighted.

Unmapped items are indicated by a change of background color.

Set Up a DataStream

To set up a DataStream, you

- add data sources and add an SQL statement to return columns of data
- add literal values to the data sources if required
- add derivations to the data sources or DataStream if required
- create DataStream items for mapping
- map data source columns returned by the data sources to the DataStream items
- map the DataStream items to either the transformation model elements or the reference structure level attributes as appropriate
- add a filter if required

Manage Data Sources

A data source is any location from which IBM® Cognos® Data Manager can access data. Each data source either contains an SQL based query, or a link to a query subject contained in a published IBM® Cognos® Framework Manager package.
A data source can also contain literal values and derivations. For information see "Define Literal Values in a Data Source" (p. 150) and "Derivations" (p. 155).

After you add a data source, you must map the data source columns to the DataStream items.

You can use multiple data sources to input data from more than one connection. This can be useful when you want to perform data cleansing using different queries. When you use multiple data sources, you must access the data using a DataStream because this is where the data sources are merged.

Add a Data Source for a SQL-based Source Data

Before you add a data source, ensure that you have created a connection to the required database from which you want to access source data. For information, see "Connections" (p. 63).

For information about creating a Published FM Package data source, see "Add Published FM Package Data Source" (p. 145).

Steps

1. Click the DataStream for which you want to add a data source.

2. Click the Insert menu, and
   - if you are adding a data source within a fact build, click Build, and then click Data Source.
   - if you are adding a data source within a reference structure, click Library, and then click Data Source.

3. Click the General tab.

4. Type a name for the data source and, if you require, a business name and description.
   Tip: To exclude the data source when the build is executed, clear the Enabled check box.

5. Click the Query tab.
   Note: You must specify the SQL statement before you can save the data source.

6. In the Database box, click the database in which the source data resides.
   Tip: Click New to create a connection. For information, see "Create a Database Connection" (p. 63).

7. In the Query pane, enter the SQL statement for the data source. You can type this or you can use one of the following methods:
   - Right-click the required table or column, and then click either Add table select statement, or Add column select statement as appropriate.
   - Hold down Ctrl and drag the table or column that you selected in the Database Objects pane to the Query pane.

For more information on entering SQL statements, see "SQLTerm and SQL Helper" (p. 55). IBM® Cognos® Data Manager clears any existing query that may be in the Query pane and generates the SQL SELECT statement for the selected table or column.
You can use Cognos® SQL when you construct the SQL statement. For information, see "Define Advanced Settings for a Data Source" (p. 144).

To view the executed SQL query, click the Show executed SQL query button.

To refresh the list of tables in the Database Objects pane, double-click the database object, or right-click and then click either Refresh Database or Refresh Table.

To clear the query, click the Clear the query button.

8. You are now ready to execute the SQL statement.

9. Click the Result Columns tab.

10. Populate the Result columns pane by clicking either Prepare or Parse, and then clicking Refresh. For more information, see "Specify Prepare or Parse" (p. 148).

11. Click OK.

Define Advanced Settings for a Data Source

You can use the advanced settings to specify the type of SQL to use, the number of rows to retrieve in one fetch call, and the isolation level.

Different databases support different isolation levels. Not all databases support transaction isolation levels. A transaction within a database delimits a unit of work. Multiple transactions from many users are usually performed on the database at the same time. Some users may be reading the database, and some users may be entering data into the database. Two transactions can interfere with each other.

Note: You cannot define advanced settings for a Published FM Package data source.

Steps

1. Click the data source for which you want to define the advanced settings.

2. From the Edit menu, click Properties.

3. Click the Query tab.

4. Click Advanced.

5. Select the Cognos SQL check box to use Cognos® SQL when you construct the SQL statement. If you clear this check box, you must use native SQL for the database you are accessing.

Note: Selecting the Cognos SQL check box in the Connection Properties dialog box, determines the default for the Cognos SQL check box.

For more information, see "Cognos SQL" (p. 40).

6. If your database supports a bulk fetch operation, in the Fetch row count box, type the number of rows that are queried in one fetch call using the bulk fetch facility for your database. Use this to tune large extractions from a source database. The default fetch row size is 50.
Note: There may be a limit on the number of rows that you can retrieve. If that limit is lower than the value in the Fetch row count box, that limit is used instead.

7. Use the Isolation level box to set the lowest transaction isolation level supported by your database to minimize the potential impact of long running queries on data that might be concurrently updated. Select

- **Read uncommitted** to make changes made by other transactions immediately available to a transaction.
- **Read committed** to give a transaction access only to rows that other transactions have committed.
- **Cursor stability** to stop other transactions updating the row upon which a transaction is positioned.
- **Repeatable read** to guarantee that rows selected or updated by a transaction are not changed by another transaction until the transaction is complete.
- **Phantom protection** to stop a transaction from having access to rows inserted or deleted since the start of the transaction.
- **Serializable** to ensure that a set of transactions executed concurrently produce the same result as if they were performed sequentially.

See your vendor documentation for more details about transaction and isolation levels.

8. Click OK.

**Add Published FM Package Data Source**

In addition to accessing source data stored in a database, you can also access query subjects contained in a published IBM® Cognos® Framework Manager package.

Before you can add a Published FM Package data source, you must

- create an IBM® Cognos® Data Manager connection to the package
  
  For information, see "Create a Published FM Package Connection" (p. 67).

- be logged on to IBM® Cognos® Connection, the portal to IBM® Cognos® Business Intelligence, which provides a single access point to all corporate data available in IBM® Cognos® BI.
  
  For information, see "Manage Credentials" (p. 71) and the IBM® Cognos® Connection User Guide.

  **Note:** If the server to which you are connecting accepts anonymous users, you do not need to log in.

**Steps**

1. Click the DataStream for which you want to add a Published FM Package data source.

2. Click the Insert menu, and
Chapter 14: DataStreams

- if you are adding a data source within a fact build, click Build, and then click Data Source.
- if you are adding a data source within a reference structure, click Library, and then click Data Source.

3. Click the General tab. Type a name for the data source and, if you require, a business name and description.

   **Tip:** To exclude the data source when the build is executed, clear the Enabled check box.

4. Click the Query tab. In the Database box, click the required Published FM Package connection.

   **Tip:** Click New to create a connection. For information, see "Create a Published FM Package Connection" (p. 67).

5. Populate the Query items pane by dragging and dropping a query subject or individual item in the Query items pane. There are three columns in this pane:
   - Name shows the name of the query subject or item
   - Expression shows the expression as it exists in the package
   - Sort Order shows the sort order to use

   **Tips**
   - To clear a single query item, click the Delete selected query items button.
   - To clear all the query items in the pane, click the Delete all query items button.

6. To apply a sort order for an item, right click in the Sort Order column, and select the relevant option:
   - None
   - Ascending
   - Descending

7. You can use filters from the model or you can create a filter in Data Manager.

   To use a filter from the model, in the Database objects pane, expand the Filters folder, click the required filter, and drag it to the Filters pane.

   For information on creating a filter in Data Manager, see "Create a Filter for a Published FM Package Data Source" (p. 147).

   You are now ready to return or execute the query.

8. Click the Result Columns tab.

   Populate the Result columns pane by clicking either Prepare or Parse, and then clicking Refresh.

   For more information, see "Specify Prepare or Parse" (p. 148).

9. Click OK.
Create a Filter for a Published FM Package Data Source

You can use a filter that was previously created in IBM® Cognos® Framework Manager and included in the published package, create a filter directly in IBM® Cognos® Data Manager, or use a combination of both.

Steps to Use a Filter From a Published Package

1. In the Database objects pane, open the Filters folder.

2. Click the filter that you require and drag it to the Filters pane.

3. Repeat the previous step as required.

Tips

- To edit a filter, in the Filters pane, double click the filter.

- To clear a single filter, click the filter and then click the Delete selected filters button.

- To clear all the filters in the pane, click the Delete all filters button.

Steps to Create a Filter in Data Manager

1. Click the Create a filter button.

The Edit Detail Filter window appears.

2. To create the filter, drag and drop query items and existing filters from the Query Items tab. You can also select user-defined variables, Data Manager Designer and run-time variables from the Variables tab.

3. Click OK.

Steps to Edit a Filter

1. In the Filters pane, double-click the filter to edit.

The Edit Detail Filter window appears.

2. Edit the filter as required.

3. Click OK.

View the Generated Query Specification

You can view the generated XML for the query, and if required, convert it so that you can manually author it. Note that when a query is converted you cannot return to using the model.

Note: This functionality is intended for advanced users who are familiar with the query specification syntax and the IBM® Cognos® Software Development Kit.

Steps

1. Click the Show query specification button.
The Query Source window appears.

2. Edit the query specification if required.
   
   **Tip:** Select the Word wrap check box to use word wrapping.

3. To convert the query items to the query specification, click **Convert**.
   
   **Note:** Converting the query items cannot be undone.

### View the Generated SQL or MDX

If you convert the query for manual authoring, you can view the generated SQL or, if your data is an OLAP source, the generated MDX query that will be executed when the query specification is processed. You can also convert the query specification to use the generated SQL or MDX as its data source. When a query is converted you cannot return to using the model.

**Steps**

1. Click the View SQL button.

2. If you are viewing SQL, from the drop down list, click either Native SQL or Cognos SQL as appropriate.
   
   For more information, see "Cognos SQL" (p. 40).

3. Edit the SQL or MDX if required.

4. To convert the query items to SQL or MDX, click **Convert**.
   
   **Note:** Converting the query items cannot be undone.

### Specify Prepare or Parse

After you enter the SQL statement, or for a published FM package data source, populated the Query Items pane, you must specify whether to use prepare or parse to populate the Result Columns tab. By default, all new data sources use the prepare method.

The returned columns are matched to the results of the query by the column's ordinal position. If you modify the query after you determined the columns, you should regenerate the list of columns by clicking Refresh. When this mapping is complete, the column name controls all other mappings in the DataStream.

#### Prepare

We recommend that you use the prepare method. It sends the query to the database and returns the column names. For example, the following SQL statement returns ContactCode and Name:

```sql
SELECT ContactCode,
FirstName & ' ' & LastName As Name
From VContact
```

You can use database-specific syntax. For example, SQL statements that do not begin with `SELECT` and database-specific concatenation characters.

**Tip:** To rename columns use the As clause in the SQL statement.
**Parse**

Parse does not send the query to the database, so you can use it when you cannot connect to the database. The query is parsed and the result set of columns returned as written in the query. For example, the following SQL statement returns FirstName & ' ' & LastName As Name:

```sql
SELECT ContactCode,
FirstName & ' ' & LastName As Name
From VContact
```

When you use parse, the SQL statement must begin with SELECT. Although using parse is quicker than using prepare, parse may fail if the SQL is too complex. Parse may not evaluate database-specific syntax correctly.

**Steps**

1. Click the data source for which you want to specify prepare or parse.
2. From the Edit menu, click Properties.
3. Click the Result Columns tab.
4. Populate the pane by clicking either Prepare or Parse.
5. Click Refresh to update the pane.
6. Click OK.

**Test a Data Source**

You can choose to return all the selected rows in the database in a tabular format, return just a single row in a tabular format, or execute the statement.

- To return all the rows, in the Test pane, click the Return all rows button.
- To return a single row, in the Test pane, click the Return one row button.
- To execute the statement and return the rows, in the Test pane, click the Execute statement button.
- To interrupt the execution of an SQL query, in the Test pane, click the Interrupt current processing button.

**Change the Properties of a Data Source**

If you change the SQL statement, you must check the DataStream mapping because it is not automatically updated.

If you change the SQL dialect, check the SQL statement because it may have become invalid.

**Steps**

1. Click the required data source.
2. From the Edit menu, click Properties.
3. Click the tab that contains the property that you want to change.
4. Change the property.
5. Click OK.

**Disable and Enable a Data Source**

To temporarily prevent a data source from acquiring data, disable and later enable the data source.

**Steps**

1. Click the data source.
2. From the **Edit** menu, click **Properties**.
3. Click the **General** tab.
4. Clear or select the **Enabled** check box.
5. Click **OK**.

**Tip:** You can also disable or enable a data source by right-clicking the data source in the **Tree** pane, and then clicking **Enabled**.

**Define Literal Values in a Data Source**

A literal value is a constant that IBM® Cognos® Data Manager inserts in every row of data that the DataStream returns. Use literal values to identify the source of each data row.

You can add literals to either a data source, or to the SQL statement of a data source.

When you add a literal value to the SQL statement, the DBMS must return the literal value for each row of data. This is a disadvantage as it may consume additional computer and network resources. Using Data Manager literals is the preferred method to introduce static values.

This is an example of adding a literal to the SQL statement:

```
SELECT 'ALL', Product, Price FROM Product;
```

**Steps**

1. Click the required data source.
2. From the **Edit** menu, click **Properties**.
3. Click the **Literals** tab.
4. Click **Add**, and then type the literal value.
5. Repeat step 4 until you have defined all the literal values that you require for the selected data source.

**Tip:** To reorder the list, click the literal value that you want to move, and click either **Move Up** or **Move Down**.
6. Click **OK**.
Delete a Data Source

You can delete a data source that is no longer required.

Steps
1. Click the data source that you want to delete.
2. From the Edit menu, click Delete.
3. In the message box, click Yes.

Manage DataStream Items

Use DataStream items to map the data source columns, returned by data sources, to the transformation model elements or reference structure level attributes.

For information about accessing source data from a database, see "Manage Data Sources" (p. 142).

Note: If part of a reference structure accesses data from a template rather than from a DataStream, mapping the data source columns to the reference structure attributes has no effect.

Create a DataStream Item

You must create a DataStream item for each data source column that you want to map to the DataStream.

Tip: IBM® Cognos® Data Manager can automatically add DataStream items when you map data source columns to the DataStream. For information, see "Map Data Source Columns to the DataStream" (p. 152).

Steps
1. Click the DataStream for which you want to create DataStream items.
2. From the Edit menu, click Properties.
3. Click the DataStream Items tab.
4. Click Add.
   The DataStream Item Properties window appears.
5. Type a name for the new item and, if you require, a business name and a description.
6. In the Type box, click the data type that the DataStream item is to return. For more information, see "Data Types" (p. 543).
   If you select Char, Number, or Binary as the return type, the Precision box becomes available.
7. In the Precision box, type the maximum number of characters or numbers permitted for the DataStream item.
   Note: The precision can be up to 77 for the Number data type, and limited by the operating system and computer limits for the Char and Binary data types.
If you selected **Number** as the return type, the **Scale** box becomes available.

8. In the **Scale** box, type the number of decimal places permitted for the DataStream item.
   **Note:** The scale value cannot exceed the precision value.

9. Click **OK**.
   The DataStream item is added to the **DataStream Items** pane.

10. Repeat steps 4 and 5 for each DataStream item you want to add.

11. Click **OK**.

**Tips**
- To reorder the list in the **DataStream Items** pane, click the item that you want to move, and click either **Move Up** or **Move Down**. Reordering DataStream items has no effect on processing.
- To edit the name of a DataStream item, click the item, and then click **Edit**.

**Map Data Source Columns to the DataStream**

To enable the flow of data from a data source to a fact build or reference structure, you must

- map the data source columns to the DataStream items, as described here
- map the DataStream items to the transformation model for a fact build or to the level attributes for a reference structure

The following is an example of mapping data source columns to DataStream items.

---

**Map Data Source Columns to Existing DataStream Items**

After you create the required DataStream items, you must map the data source columns to them.
Note: If you map more than one data source column to a single DataStream item for a fact build, the fact data will contain duplicate rows. A separate row is delivered for each data source column you map.

**Steps**

1. Click the appropriate DataStream.

2. From the Edit menu, click Properties.

3. In the Data Source column, click the column to map.

4. In the DataStream Items column, click the item to which you want to map the column.

5. Click Map.

   Tip: You can also perform mapping by dragging the DataStream item to the relevant position in the Maps To column.

6. Repeat steps 3 to 5 to complete the required mapping.

7. Click OK.

**Tips**

- To map a data source column to more than one DataStream item, repeat steps 3 and 4 to select the required data source column and DataStream item, and then hold down the Ctrl key and click Map.

- To clear a mapping, click the mapped item in the Maps To column, and then click Clear.

- To clear all the mappings, click Clear All.

**Map Data Source Columns and Create DataStream Items**

Rather than creating DataStream items and then mapping data source columns to the DataStream separately, you can perform both tasks together.

Note: If you map more than one data source column to a single DataStream item for a fact build, the fact data will contain duplicate rows. A separate row is delivered for each data source column that you map.

**Steps**

1. Click the appropriate DataStream.

2. From the Edit menu, click Properties.

3. In the Data Source column, double-click the data source column that you want to map to the DataStream.

   A DataStream item is created and the selected data source column is mapped to the item.

   - Ensure that nothing is selected in the DataStream Items pane because this maps the data source column to the selected DataStream item.
If a matching DataStream item already exists, a new item is not created, but the data source column is mapped to the existing one.

4. Repeat step 3 to complete the required mapping.
5. Click OK.

Tips
- To clear a mapping, select the mapped item in the Maps To column, and then click Clear.
- To clear all the mappings, click Clear All.

Map Data Source Columns Automatically
You can map all unmapped data source columns in a single step.

If DataStream items and data source columns exist with matching names, you can automatically perform mapping. For example, if you have a data source column named OrderDate, and a DataStream item named OrderDate, you can automatically map them.

If no matching DataStream item exists for a data source column, they are created during the automatic mapping process.

Steps
1. Click the appropriate DataStream.
2. From the Edit menu, click Properties.
3. Click Auto Map.
   All the unmapped data source columns are mapped.
4. Click OK.

Delete a DataStream Item
You can delete a DataStream item that is no longer required.

Steps
1. Click the DataStream for which you want to delete DataStream items.
2. From the Edit menu, click Properties.
3. Click the DataStream Items tab.
4. In the DataStream Items pane, click the DataStream item that you want to delete.
5. Click Delete.
6. Click OK.
Derivations

A derivation is a value that is calculated using an expression that you define, instead of obtained directly from the source data. You can derive values from individual columns in the source data. For example, you can multiply Price by Units_Sold to create a column named Daily_Sales_Revenue.

You can add derivations to the DataStream or data sources of a fact build or reference structure. You can also add derivations to the transformation model of a fact build. Use the following guidelines when deciding where to perform a derivation.

Where to Perform a Derivation

Perform derivations in the DataStream when

- the same calculation is used for each of the data sources, that is, the data for each data source must undergo the same derivation.
- the result of the derivation is required before the data is used in the transformation model, that is, the derivation concatenates keys before the result is used in the transformation model for a reference structure.
- the result of the derivation is to be pivoted.

Perform derivations in the data source when

- you have more than one data source, but the derivation will only be performed on one of them.
- the result of the derivation will be used as an input for other derivations in the DataStream.

Perform derivations in the transformation model when

- the values are only available after the data has been merged.
- the data is only available after pivoting has been performed.
- the data uses input values that are stored from reference structures.

For information about creating derivations in the transformation model, see "Add a Derivation Element to a Fact Build" (p. 165).

Expressions in Derivations

When you are entering the derivation expression, you can use the tree structure that is in the left pane of Calculation tab to select items from the following folders:

- Operators to select operators
- Functions to select predefined and user-defined functions
- Control to select controls
- Variables to select variables and static variables

If you are defining a DataStream derivation, you can also select items from the following folders:

- DataStream Items to select DataStream items
DataStream Derivations to select other derivations defined within the same DataStream

If you are defining a data source derivation, you can also select items from the following folders:

- Columns to select columns returned by the data source
- Literals to select literal values defined in the data source
- Data Source Items to select data source items
- Data Source Derivations to select other derivations defined within the same data source

**Add a Derivation**

This section describes how to add a derivation to a DataStream.

**Steps**

1. Click the DataStream or data source for which you want to define a derivation.

2. From the **Edit** menu, click **Properties**.

3. Click the **Derivations** tab.

   Any derivations that are already defined are listed.

   **Tip:** To edit a derivation, click the derivation, and then click **Edit**.

4. Click **Add**.

   The **Derivation Properties** window appears.

5. Click the **General** tab. Type a name for the derivation and, if you require, a business name and a description.

   **Tip:** If you are creating a DataStream derivation that you do not want to be available for mapping in the transformation model, select the **Private** check box.

6. Click the **Calculation** tab. In the right pane, enter an expression for the derivation.

   You can type the expression, and you can use the tree structure in the left pane to select the required items.

   For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305). For information about variables, see "Variables" (p. 319).

7. In the **Return Type** box, click the expected data type for the value that the expression is to return. For more information, see "Data Types" (p. 543).

   **Tip:** We recommend that you select the same data type as specified in the source data to avoid inconsistencies when the data is converted.

   **Note:** If you do not specify a return type, a default data type is assigned during execution.

   If you select **Char**, **Number**, or **Binary** as the return type, the **Precision** box becomes available.

8. In the **Precision** box, type the maximum number of digits permitted for the value.
Note: The precision can be up to 77 for the **Number** data type, and limited by the operating system and computer limits for the **Char** and **Binary** data types.

If you selected **Number** as the return type, the **Scale** box becomes available.

9. In the **Scale** box, type the number of decimal places permitted for the value.

   **Note**: The scale value cannot exceed the precision value.

10. If required, test the expression. For information, see "**Testing Expressions and Scripts**" (p. 341).

11. Click **OK**.

    The derivation is added to the **Derivations** tab in the **Properties** window.

**Tips**

- To show the full text of each expression listed, select **Show All Expression Text**.

- Derivations are resolved in the order they appear. To reorder the list, click the derivation that you want to move, and click either **Move Up** or **Move Down**.

- To delete a derivation, click the derivation, and then click **Delete**.

**Reduce the Amount of Input Data**

You may want to restrict the amount of input data to speed up processing while testing a build or to produce statistical data based on a sample of the source data.

You can use the following methods to reduce the amount of data that a DataStream makes available to other IBM® Cognos® Data Manager objects:

- data sampling

- applying a maximum row limit

You can control the order in which the data sources are processed, and the order in which each data source presents the data it accesses. This provides some control over which data rows are processed and passed to the DataStream output.

**Sample the Data**

To configure a DataStream to sample the data, you specify a sample row frequency for that DataStream. When you do this, the DataStream processes one data row in each batch of the size you specify. For example, if you specify a sample row frequency of 100, one data row is processed in every 100 available rows of source data.

The order in which the DataStream processes the data sources affects which rows it processes. For example, if the DataStream is configured to process the data sources in parallel, the sampled data may be biased toward one of the data sources, possibly to the exclusion of another data source. You can avoid this by configuring the DataStream to process the data sources in series.
For more information, see "Data Source Processing Order" (p. 159).

**Steps**
1. Click the DataStream.
2. From the **Edit** menu, click **Properties**.
3. Click the **Input** tab.
4. In the **Sample row frequency** box, type the required value.
5. Click OK.

**Apply a Maximum Row Limit**

You apply a maximum row limit to restrict the number of data source rows that the DataStream processes. For example, if you specify a maximum row limit of 2000, the DataStream processes no more than 2,000 source data rows.

In addition to the maximum row limit that you set, the following other factors affect how many rows of source data are read:
- the sample row frequency setting for the DataStream
- data source pivoting

**Steps**
1. Click the DataStream.
2. From the **Edit** menu, click **Properties**.
3. Click the **Input** tab.
4. In the **Maximum input rows to process** box, type the required value.
5. Click OK.

**Apply Sample Frequency and a Maximum Row Limit**

If you apply both a sample frequency and a row limit to a DataStream, the sample rate is applied, and then the row limit is applied to the sampled data. For example, if you specify a sample frequency of 100 and a row limit of 1,000, IBM® Cognos® Data Manager processes one row in every 100 rows until 1,000 rows are processed. Data Manager reads up to 100,000 rows from the data sources. If the data sources contain only 50,000 rows of data, the DataStream outputs only 500 rows.

If data source pivots apply to a transformation model element, the pivot is applied, and then the row limit is applied to the pivoted data. For example, consider a transformation model element pivot that results in the generation of twelve rows of data from each row of source data. If you specify a row limit of 12,000, that limit is reached after 1,000 source data rows are processed.
Data Source Processing Order

By default, data sources are read in series unless you are using breaking, in which case they are read in parallel. When data sources are read in series, each data source is opened sequentially, as they are required.

For information on breaking, see "Dimension Breaks" (p. 378).

When data sources are processed in series, the order in which they appear in the Tree pane is used. You can change this order if you require.

Steps to Change the Order of Data Sources

1. Click the data source.

2. From the Actions menu, click either Move Up or Move Down.
   Tip: You can also reorder a data source by clicking it in the Tree pane and then dragging it to the required position.

3. Repeat steps 1 and 2 until the data sources appear in the correct order.

4. Click OK.

Change the Order of Data in a Data Source

You can use either of the following methods to change the order in which a data source presents data:

- For any data source, you can include an ORDER BY clause in the SQL statement of the data source. This clause changes the order in which the DBMS, and hence the data source, presents the data.

- If the data source accesses data for a fact build, you can specify dimension breaks. This appends an appropriate ORDER BY clause to every data source of the DataStream.

   For information about reference dimension breaking, see "Dimension Breaks" (p. 378).

Filter Source Data

A DataStream filter removes unwanted data rows before they reach the transformation model. This eliminates passing unwanted rows through the transformation model, and then later, filtering them in the deliveries. Using a DataStream filter can improve the transformation capability for dimension data and fact data.

A DataStream filter consists of a logical expression that evaluates to true or false when applied to each source data row. Only the data rows for which the expression evaluates to true are made available to the transformation model. If no DataStream filter is applied, all the data rows are output. When adding an expression, in addition to using operators, functions, controls, and variables, you can also use DataStream items and DataStream derivations for the selected DataStream.

In the following example, the DataStream only allows source data rows where the quantity of units sold is greater than 80.
Steps

1. Click the DataStream.

2. From the Edit menu, click Properties.

3. Click the Filter tab.

4. In the right pane, enter an expression for the filter.
   
   You can type the expression, and you can use the tree structure in the left pane to select the required items.
   
   You can select an item from a folder by dragging it to the right pane, or by double-clicking it.
   
   For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305). For information about variables, see "Variables" (p. 319).

5. Click OK.

   Tip: You can test the filter by executing the DataStream. For more information, see "Execute a DataStream" (p. 160).

Execute a DataStream

You can execute a DataStream to examine the source data prior to transformation. By default, IBM® Cognos® Data Manager executes

- the query for each data source set up within the DataStream.
- all data source and DataStream derivations.
the first 100 data source rows, regardless of the number of data sources. For example, if you have two data sources, the first 50 rows from each data source are executed. You can, however, specify a different row limit.

You can disable any data source within a DataStream if you do not want to include it in the execution. You can also apply a sample frequency and row limit to the DataStream execution.

**Steps**

1. Click the DataStream.

2. From the **Actions** menu, click **Execute**.

   **Note:** If there are substitution variables included in any of the data source SQL statements, the **Values for Expression** window appears. If required, enter values for the substitution variables, then click **OK**.

   For more information on substitution variables, see "Include Substitution Variables in an SQL Query" (p. 61).

   The **Execute DataStream** window appears and the DataStream executes using the default settings. When it has finished, the results appear in the bottom pane.

   **Tip:** To stop a DataStream from executing before it has finished, click **Interrupt**.

3. If you want to stop a data source from executing, in the **Data Sources** pane, clear the appropriate check box.

   **Note:** This is different to disabling the data source.

   For information, see "Disable and Enable a Data Source" (p. 150).

4. If you want to apply a sample frequency to the DataStream execution, in the **Sample row frequency** box, type the required value.

   **Note:** The value you set here applies to the DataStream execution only. It has no effect on the sample row frequency set up in the DataStream Properties window.

   For more information, see "Sample the Data" (p. 157).

5. If you want to apply a row limit to the DataStream execution, in the **Maximum input rows to process** box, type the required value.

   **Note:** The value you set here applies to the DataStream execution only. It has no effect on the row limit set up in the DataStream Properties window.

   For more information, see "Apply a Maximum Row Limit" (p. 158).

6. If the DataStream has a filter, you can choose to disable it by selecting the **Disable Filter** check box.

   For information, see "Filter Source Data" (p. 159).

7. To execute the DataStream again using your new settings, click **Refresh**.

8. When you have finished checking the data, click **Close**.
Trailing Spaces in String Values

By default, trailing spaces are removed from values that are accessed. This conserves memory and improves performance, particularly with large data sets. However, you can preserve trailing spaces if they are significant.

Steps

1. Replace the column with a calculated value in the SQL for the data source.
   The calculated value should concatenate a non-space character (for example, the piping symbol) to the column value.

2. Create a derivation element in the transformation model that removes the concatenated character from the calculated value.
   For more information, see "Aggregate Data" (p. 183).

3. Deliver the derivation element, not the SQL column.

Example

Consider an Access table named Example that has a TxtVar column.

Steps

1. Replace SELECT TxtVar FROM Example with SELECT TxtVar & '' As CalcVar FROM Example

2. Create a derivation transformation model element that uses the formula RTRIM(CalcVar, '') to remove the concatenated pipe symbol.

3. Deliver the derivation to the target fact tables.
Chapter 15: Transformation Models

The transformation model is central to a fact build. You use it to manipulate the acquired source data in a number of ways, such as merging data from different sources and aggregating data.

The transformation model contains the elements of transformed data. You can create these element types:

- attributes
- derivations
- dimensions
- derived dimensions
- measures

**Attributes**

An attribute element holds additional information that is not a dimension or a measure but that may be of interest. Attributes differ from measures in that they cannot be aggregated.

Attributes are usually one of the following:

- an attribute of one of the dimensions, such as unit weight or size
- a property of the record, such as the name of the operator who entered the record or the timestamp of record creation

**Note:** Attributes only have data values at levels retrieved directly from input data SQL queries. In summary information, the value of an attribute is always null.

**Derivations**

A derivation element is a value that is calculated by using an expression that you define instead of obtained directly from the source data.

**Dimensions**

A dimension element contains data that is used to provide context for a measure element. For example, a Product_Number dimension element provides context for a Quantity measure element.

**Note:** Dimension elements provide the link to the dimensional framework.

**Derived Dimensions**

A derived dimension element allows you to use a calculated expression as the source for a dimension attribute. This expression can be the result of another dimensional element that exists in the transformation model.
Measures
A measure element is a value, frequently numeric, that holds a piece of information for analysis, such as units, revenue, or cost.

View Transformation Models
You can view transformation models in the Tree pane or the Visualization pane.

View Transformation Models in the Tree Pane
The transformation model is represented in the Tree pane as follows.

View Transformation Models in the Visualization Pane
When you click a fact build in the Tree pane, a visual representation appears on the Transformation Model tab in the Visualization pane. The tab shows
• each element in the transformation model
• the reference structure to which each dimension element relates
• the hierarchy levels at which data is accessed (input)
• the hierarchy levels at which data is delivered (output)

Add an Attribute Element to a Fact Build
An attribute element holds additional information that is not a dimension or a measure but that may be of interest. Attributes differ from measures in that they cannot be aggregated.

Tip: You can automatically add an attribute element to the fact build when you map DataStream items to the transformation model. For information, see "Map the DataStream to the Transformation Model" (p. 201).

Steps
1. In the fact build, click Transformation Model.
2. From the **Insert** menu, click **Build** and then click **Attribute**.

3. Click the **General** tab.

4. Type a name for the new element. If required, type a business name and description.

   **Tip:** If you do not want the element made available to the deliveries, select the **Never output** check box. This is useful if an element is required only for use in calculations.

5. Click the **Merge** tab to change the merge method from the default value of first non-null.

   For more information, see "Merge Data for a Measure or an Attribute" (p. 200).

6. Click **OK**.

7. If the fact build has deliveries defined, a message appears asking whether you want to add the element to existing delivery modules in the build. Click **Yes** or **No** as appropriate.

   For more information, see "Add an Element to the Deliveries" (p. 204).

---

**Add a Derivation Element to a Fact Build**

A derivation is a value that is calculated using an expression that you define, instead of being obtained directly from the source data. You can derive values from individual columns in the source data. For example, you can multiply Price by Units_Sold to create a column named Daily_Sales_Revenue.

You can add derivation elements to the transformation model of a fact build. You can also add derivations to the DataStream or data sources of a fact build or reference structure. For information, see "Derivations" (p. 155).

**Tip:** Use derivations in data sources and the DataStream when the result of the derivation is required for pivoting or for looking up a reference element.

**Steps**

1. In the fact build, click **Transformation Model**.

2. From the **Insert** menu, click **Build**, and then click **Derivation**.

3. Click the **General** tab.

4. Type a name for the derivation. If required, type a business name and description.

   - If you do not want the element made available to the deliveries, select the **Never output** check box. This is useful if an element is required only for use in subsequent calculations.

   - If you want aggregation performed for a derivation element, select the **Calculate at Source** check box. For more information, see "Calculate a Derivation Before Aggregation" (p. 185).

5. Click the **Calculation** tab.

   For information about entering calculations, see "Add a Calculation to a Derivation or Derived Dimension" (p. 178).
6. If you selected **Calculate at Source** on the **General** tab, the **Aggregation** tab is available. Click this tab to specify the aggregation details.

   For more information, see "Aggregate Data" (p. 183).

7. Click **OK**.

8. If the fact build has deliveries defined, a message appears asking whether you want to add the element to existing delivery modules in the build. Click **Yes** or **No** as appropriate.

   For more information, see "Add an Element to the Deliveries" (p. 204).

---

**Add a Dimension Element to a Fact Build**

A dimension element contains data that is used to provide context for a measure element. For example, a Product_Number dimension element provides context for a Quantity measure element.

For more information, see "Business Dimensions" (p. 26).

**Tip:** You can automatically add a dimension element to the fact build when you map DataStream items to the transformation model. For information, see "Map the DataStream to the Transformation Model" (p. 201).

**Steps**

1. In the fact build, click **Transformation Model**.

2. From the **Insert** menu, click **Build**, and then click **Dimension**.

3. In the **Properties** window, click the **General** tab.

4. Type a name for the new element. If required, type a business name and description.

   **Tip:** If you do not want the element made available to the deliveries, select the **Never output** check box. This is useful if an element is required only for use in subsequent calculations.

5. Click the **Reference** tab. You must use the **Dimension** and **Structure** boxes to provide a link between the fact table and the dimension table.

   For information, see "Associate a Dimension or Derived Dimension Element with a Reference Item" (p. 171).

6. Use the remaining tabs to specify other details for the dimension element.

   For information, see the relevant sections in this chapter.

7. Click **OK**.

8. If the fact build has deliveries defined, a message appears asking whether you want to add the element to existing delivery modules in the build. Click **Yes** or **No** as appropriate.

   For more information, see "Add an Element to the Deliveries" (p. 204).
Add a Derived Dimension Element to a Fact Build

A derived dimension allows you to use a calculated expression as the source for a dimension attribute. You can use a derived dimension to perform dimensional lookups against existing reference data. This reduces the need to stage data or use the Lookup function in a data source or DataStream derivation.

You must enter a calculated expression as the source for the derived dimension. The expression can refer to any measure, attribute, or dimension element in the transformation model, and any derived dimension that is before it in the transformation model. However, you cannot use a derivation element in the calculation because these are calculated after merging, so they are not available.

You can create derived dimensions that are nested to more than one level, that is, a lookup on a lookup on a lookup, and so on.

When IBM® Cognos® Data Manager executes the build, dimension elements are validated first, then the first derived dimension that is listed in the transformation model, followed by the second, and so on. This order is especially important for nested derived dimensions.

**Steps**

1. In the fact build, click **Transformation Model**.
2. From the **Insert** menu, click **Build**, and then click **Derived Dimension**.
3. Click the **General** tab.
4. Type a name for the new element. If required, type a business name and description. **Tip:** If you do not want the element made available to the deliveries, select the **Never output** check box. This is useful if an element is only required for use in subsequent calculations.
5. Click the **Calculation** tab. You must enter a calculation to access the reference structure where the information that you require is held. For information, see "Add a Calculation to a Derivation or Derived Dimension" (p. 178).
6. Click the **Reference** tab. Select the reference dimension and the reference structure that holds the required information. For information, see "Associate a Dimension or Derived Dimension Element with a Reference Item" (p. 171).
7. Use the remaining tabs to specify other details for the derived dimension element. For information, see the relevant sections in this chapter.
8. Click **OK**.
9. If the fact build has deliveries defined, a message appears asking whether you want to add the element to existing delivery modules in the build. Click **Yes** or **No** as appropriate. For more information, see "Add an Element to the Deliveries" (p. 204).
This example describes how you can use a derived dimension to perform a lookup on a lookup. You have sales records that, among other things, show the number of items sold, by which sales person, and on which date. The information is referenced by lookups.

The number of items sold show how many single, double, triple, or multi packs of tent pegs were sold. What you want to know is not how many packs have been sold, but how many individual tent pegs. The Product_UOM table contains a unit of measure and a multiplier. IBM® Cognos® Data Manager uses these to calculate how many tent pegs have been sold.

There is no direct join between the sales data and the ProductUOM which is contained in the Product dimension. You can use a derived dimension to link the ProductUOM to the ProductDD table, using the ProductUOMCode which exists in both lookups.

A derived dimension allows us to use the Product_UOM_Code reference attribute from the Product_Lookup as the source for the Product_UOM_Lookup.

Before you create a derived dimension, you must have created the required data sources and DataStream, performed mapping between them, and created the required reference structures.

You can work through the following steps using the ds_advanced.mdb sample catalog, and you can also explore the DerivedDimensionExample fact build.

Steps
1. Add a derived dimension element to the transformation model.
2. In the Derived Dimension Properties window, click the Calculation tab.
3. In the Expression pane, double click Reference Attributes and then PRODUCT_CODE. PRODUCT_UOM_CODE. PRODUCT_UOM_CODE is the attribute on the ProductDDLookup used to link to the Product_UOM_Lookup because it exists in both.
4. Click the Reference tab.

5. In the Dimension box, click Product, and in the Structure box, click ProductUOMLookup.

This allows Data Manager to access the ProductUOMLookup that contains the Product_UOM attribute that is the multiplier required to calculate the number of individual tent pegs.

6. Click OK.

7. Add a derivation element to the transformation model.

8. In the Derivation Properties window, click the Calculation tab.

9. In the right pane, enter the following calculation:
   
   Quantity * UOM.PRODUCT_UOM


This multiplies the quantity, which is the number of packs, with the Product_UOM value which defines the number of tent pegs in the pack.

**Tip:** UOM.PRODUCT_UOM is an attribute in the ProductUOMLookup. You can select it by expanding the Reference Attributes folder and double clicking the attribute.

10. Click OK.

### Add a Measure Element to a Fact Build

A measure element is a value, frequently numeric, that holds a piece of information for analysis, such as units, revenue, or cost.

**Tip:** You can automatically add a measure element to the fact build when you map DataStream items to the transformation model. For information, see *Map the DataStream to the Transformation Model* (p. 201).

**Steps**

1. In the fact build, click **Transformation Model**.
2. From the **Insert** menu, click **Build**, and then click **Measure**.
3. Click the **General** tab.
4. Type a name for the new element. If you require, type a business name and description.
   
   **Tip:** If you do not want the element made available to the deliveries, select the **Never output** check box. This is useful if an element is required only for use in subsequent calculations.
5. Click the **Aggregation** tab to specify the aggregation details.
For information, see "Aggregate Data" (p. 183).

6. Click the Merge tab to change the merge method from the default value of first non-null.
For more information, see "Merge Data for a Measure or an Attribute" (p. 200).

7. Click OK.

8. If the fact build has deliveries defined, a message appears asking whether you want to add the element to existing delivery modules in the build. Click Yes or No as appropriate.
For more information, see "Add an Element to the Deliveries" (p. 204).

Associate a Dimension or Derived Dimension Element with a Reference Item

You must provide a link between the fact table and the dimension table. You do this by associating each dimension element and derived dimension element with a reference dimension, and with a reference structure from within the selected reference dimension.

You make the associations from the relevant Properties window.

Note: For dimension elements, you can also make the association during the mapping process. For information, see "Steps to Associate a Dimension Element With a Reference Item During the Mapping Process" (p. 172).

Steps to Associate a Dimension or Derived Dimension Element With a Reference Item

1. In the transformation model, click the appropriate dimension element or derived dimension element.

2. From the Edit menu, click Properties.

3. Click the Reference tab.

4. In the Dimension box, click the dimension that holds the reference structure that you want to use.

5. In the Structure box, click the reference structure that you want to use.
You can choose from all the hierarchies (shown suffixed with H), auto-level hierarchies (shown suffixed with A), and lookups (shown suffixed with L) in the selected reference dimension.

If you select a hierarchy, the grid becomes available. Use this grid to set the output levels for a dimension and the input and output levels for a derived dimension. For information, see "Set Input and Output Levels for a Dimension or Derived Dimension" (p. 173).

If you select a hierarchy or an auto-level hierarchy, the Aggregate check box becomes available. For information, see "Aggregate Data" (p. 183).

6. Click OK.

**Steps to Associate a Dimension Element With a Reference Item During the Mapping Process**

1. Click the appropriate transformation model.
2. From the Edit menu, click Mapping.

   The Transformation Model Mapping window appears.

3. In the Transformation Model pane, expand the appropriate dimension element.
4. Click no reference, and then click the Browse button that appears.
5. In the Select Reference dialog box, expand the dimension that holds the reference structure you want to use, and then click the required reference structure.

6. Click OK.
You can now map a DataStream item to the dimension element at a specific attribute level. This is known as setting the input level. For more information, see "Set Input Levels" (p. 173).

**Note:** You cannot map to a derived dimension element because the source data comes from a derivation, not the DataStream.

**Set Input and Output Levels for a Dimension or Derived Dimension**

When you associate a dimension element or a derived dimension element with a hierarchy, you can also set the levels at which data is accessed (the input levels) and delivered (the output levels).

You can view the input and output levels for all the dimension and derived dimension elements in the selected build by clicking the Transformation Model tab in the Visualization pane.

**Set Input Levels**

Data can enter the fact build process only at the hierarchy input levels.

The method used to set the levels at which source data is to be accessed depends on whether you are defining a dimension element or a derived dimension element.

**Set Input Levels for a Dimension**

For a dimension, you set the levels at which source data is to be accessed when you map a DataStream item to a transformation model dimension element. Mapping a DataStream item to a dimension element differs from mapping other element types because you cannot map a DataStream item directly to the dimension element. You map the DataStream item to the attribute ID for the required input level.

By default, when you set an input level, you map to the ID attribute of the lowest level of a hierarchy.

**Notes**

- The ID attribute of the lowest level is always shown in bold.
- You can use only ID attributes to set input levels. It is not possible to map to any other attributes.

For more information, see "Map the DataStream to the Transformation Model" (p. 201).

**Steps**

1. Click the appropriate transformation model.

2. From the **Edit** menu, click **Mapping**.

   The **Transformation Model Mapping** window appears.
3. In the **Transformation Model** pane, expand the appropriate dimension and reference structure.

4. Click the attribute ID for the required input level.

5. In the **DataStream Item** column, click the item for which you want to set the input level.

6. Click **Map**.

   **Tip:** You can set more than one input level for a DataStream item. Repeat steps 3 to 5 to select the required attribute ID and DataStream item, and then hold down the Ctrl key and click **Map**.

7. Click **OK**.

**Set Input Levels for a Derived Dimension**

You set the levels at which data is to be input using the grid that appears when you associate a dimension or derived dimension with a hierarchy. In the grid

- **Level** lists the levels in the selected hierarchy.
- **Input** allows you to set the levels at which IBM® Cognos® Data Manager is to accept input data.
- **Output** allows you to set the levels at which data is to be made available to the deliveries. For information, see "Set Output Levels for a Dimension or Derived Dimension" (p. 175).
- **Dimension** shows the levels that are to be described in dimension deliveries. For information, see "Add a Table to a Dimension Delivery" (p. 228).
Steps
1. In the transformation model, click the required derived dimension.
2. From the Edit menu, click Properties.
3. Click the Reference tab.
4. In the Input column, click the appropriate check boxes to set the levels at which Data Manager is to accept input data.
5. Click OK.

Set Output Levels for a Dimension or Derived Dimension

A fact build delivers data only at the hierarchy output levels that you set.

Aggregation is used to calculate the data to be output. For example, suppose you have a fiscal hierarchy with levels of Year, Quarter, and Period, with data input at the Period level. The values are aggregated at the Period level to obtain values for the Quarter level and the Year level.

You set the levels at which data is to be delivered using the grid that appears when you associate a dimension or derived dimension with a hierarchy. In the grid

- Level lists the levels in the selected hierarchy.
- Input shows, and for a derived dimension allows you to set, the levels at which IBM Cognos Data Manager is to accept input data. For information, see "Set Input Levels" (p. 173).
- Output allows you to set the levels at which data is to be made available to the deliveries.
- Dimension shows the levels that are to be described in dimension deliveries. For information, see "Add a Table to a Dimension Delivery" (p. 228).
Steps
1. In the transformation model, click the dimension element or derivation element.

2. From the Edit menu, click Properties.

3. Click the Reference tab.

4. In the Output column, click the appropriate check boxes to set the levels at which data is to be output.

5. If Data Manager is to output data for levels for which there will be no input data, select the Aggregate check box. Aggregation is then performed to obtain the output data for any associated measures and derivations.

   For more information, see "Aggregate Data" (p. 183).

6. Select the Use surrogates when available check box to use the surrogate value in the output. For information, see "Surrogate Keys" (p. 83).

7. Click OK.

Exclude Input Data from Fact Build Output

You can specify that the fact build should not deliver any of the input data, but simply use it for aggregation. If you choose this, you can also choose to exclude only the detail level, that is, data at the lowest level of each hierarchy.

Excluding input data and data at the detail level applies only to the fact build, not to a particular dimension or derived dimension.

This table uses a fiscal hierarchy to show the levels at which data is output if you select Exclude Input Data.
This table uses a fiscal hierarchy to show the levels at which data is output if you select Exclude Detail Level Only.

### Steps

1. Click the required fact build.

2. From the Edit menu, click Properties.

3. Click the Output tab.

4. If you do not want to deliver any of the input data, select the Exclude Input Data check box.

5. If source data is obtained from a hierarchy, and you want to exclude data at the lowest level, select the Exclude Detail Level Only check box.

### Cache Reference Data on Demand

You can cache reference data for a dimension or a derived dimension.

By default, all reference data is cached at the start of a fact build execution. In some cases, this may not be the most efficient method of looking up reference elements.

For example, you may have a large customer dimension that contains records of all customers who have ever made a purchase from a company. If you load sales records on a daily basis, it is unlikely that every customer will have a sales transaction on each day.

Caching all reference data for a large dimension consumes resources. Instead, you can specify that reference data is cached on demand, so that only those reference members that appear in the DataStream of a fact build are retrieved. This means that fewer members are likely to be cached.

Once cached, reference members remain in memory so that they can be referenced again for any subsequent occurrence of a reference key.
Notes

- Caching on demand causes more input and output, as records are retrieved only when they are required. If you know that a large proportion of records need to be cached, it is faster to cache all reference data at the start of the fact build.

- To cache reference data on demand, you must have an appropriate index on the dimension table.

- To cache reference data on demand, the dimension element or derived dimension element must be associated with a lookup that uses a template to access data.

Steps

1. In the transformation model, click the dimension element or derivation element.

2. From the Edit menu, click Properties.

3. Click the Reference tab.

4. Select the On-demand caching check box.

5. Click OK.

Add a Calculation to a Derivation or Derived Dimension

The value of a derivation or derived dimension is calculated using an expression that you define.

Steps

1. Click the Calculation tab.

2. In the right pane of the Expression box, enter the derivation expression.

You can use the tree structure in the left pane to select the items from the following folders:

- Elements to select other elements defined within the transformation model
- Reference Attributes to select a dimension and member combination
- Operators to select operators
- Functions to select predefined and user-defined functions
- Control to select controls
- Variables to select variables and substitution variables

For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305). For information about variables and substitution variables, see "Variables" (p. 319).

You can select an item from a folder by dragging it to the right pane or by double-clicking it.
3. In the **Return Type** box, click the expected data type for the value that the expression is to return. For more information, see "Data Types" (p. 543).

**Note:** If you do not specify a return type, a default data type is assigned during execution.

**Tip:** We recommend that you select the same data type as specified in the source data to avoid inconsistencies when the data is converted.

If you select **Char** as the return type, the **Precision** box becomes available. Type the maximum number of characters permitted for the value. The maximum value is limited by the computer resources and the target database limits.

If you select **Number** in the **Precision** box, the **Precision** box and the **Scale** box become available. In the **Precision** box, type the maximum number of characters or numbers permitted for the value (up to 77). In the **Scale** box, type the number of decimal places permitted for the value.

4. In the **Scale** box, type the number of decimal places permitted for the value.

5. If required, test the expression. For information, see "Testing Expressions and Scripts" (p. 341).

6. Click **OK**.

   The **Add New Element** dialog box appears, prompting you to add the derivation element to fact and metadata deliveries.

   For information, see "Add an Element to the Deliveries" (p. 204).

---

**Dynamic and Reference Domains**

A domain is a set of values that exists within a dimension element or derived dimension element. There are two domain types: **dynamic domains** and **reference domains**.

**Dynamic Domains**

In IBM® Cognos® Data Manager, a dynamic domain is assembled from the data source rows. A dynamic domain may not necessarily contain all the members in the dimension. For example, suppose the structure data with which the dimension is associated has members A, B, C, D, and E, and your fact data contains values for A, B, and X. The dynamic domain would be A, B, and X. That is, the dynamic domain would not include the dimension members C, D, and E because they have no associated fact data.

The dimension value X does not appear in the structure data. For more information, see "Accept Source Data Containing Unmatched Dimension Values" (p. 179).

**Accept Source Data Containing Unmatched Dimension Values**

By default, a source data record containing any dimension value that does not match a member of that dimension’s reference data, is rejected when a fact build is executed. These dimension values are known as unmatched members.

There are three options you can set for unmatched members

- **Accept unmatched member identifiers** to include records that would normally be rejected
- **Save unmatched member details via reference structure** to add unmatched members to the dimension reference data so that the data is not unmatched in this and subsequent build executions.

  It is not necessary to include a dimension delivery in a fact build to save unmatched members. However, because a template is used to define the behavior of dimension attributes, the reference structure must use a template for data access.

  When unmatched data is added to the dimension reference data, surrogate keys, data defined in the template, and current indicator flags are also updated.

- **Commit each saved member** to commit each unmatched member individually after saving it.

  If the fact table delivered has a foreign key constraint on the dimension table, this option is required to successfully save unmatched members.

  Tip: Committing individual members may have a negative effect on performance. To overcome this problem, consider disabling the foreign key constraint while executing the fact build.

**Notes**

- **Save unmatched member details via reference structure** is available only when the dimension element or derived dimension element is associated with a lookup that uses a template for data access, or with a hierarchy where the lowest input level use a template.

- If you are running more than one fact build simultaneously, and you want to save unmatched members from each build, you must set up distributed handling of unmatched members. For more information, see "Distributed Handling of Unmatched Members" (p. 181).

- Unmatched members are treated as data at the lowest input level of the reference data.

**Steps**

1. Click the required dimension element or derived dimension element.

2. From the **Edit** menu, click **Properties**.

3. Click the **Unmatched Members** tab.

4. To accept source data that does not relate to any dimension reference data, select the **Accept unmatched member identifiers** check box.

5. To add the unmatched members to the dimension reference data, click the **Save unmatched member details via reference structure** check box.

   **Note:** If the conditions for adding unmatched members are not met, a message appears indicating this and the check box is not available.

6. To commit each unmatched member individually after saving it, select the **Commit each saved member** check box.

   **Note:** If the conditions for adding unmatched members are not met, a message appears indicating this and the check box is not available.

7. Click **OK**.
Distributed Handling of Unmatched Members

By default, if you attempt to execute more than one fact build simultaneously, and unmatched members are found for the same dimension in the builds, it is likely that the builds will fail as a result of duplicate key errors or data integrity errors.

The following example illustrates this problem. You can see the template table for the Product dimension. The Surrogate ID and Product ID columns accept unique IDs only. Three fact builds, A, B, and C, are executed simultaneously. Builds A and B both contain an unmatched member, P5, and build C contains an unmatched member, P6.

Assuming build A processes P5 before either build B or build C processes its unmatched members, this build will add a new row to the template table, as illustrated.

If build B processes next, it will report a duplicate key error on either the Surrogate ID column or the Product ID column when attempting to add the unmatched member P5 to the template table. Similarly, build C will report a duplicate key error when attempting to add the unmatched member P6 to the template table because it tries to add a new row and insert the value 5 into the Surrogate ID column.

You can overcome these problems by using the distributed handling of unmatched members feature, which involves the use of locks. When executing simultaneous builds, if IBM® Cognos® Data Manager finds unmatched members in a build, it uses a lock to prevent other builds from accessing the dimension reference data until it has finished adding unmatched members to it. It then releases the lock. When another build containing unmatched members accesses the dimension reference data, it can now match source data to the newly added member. Alternatively, if the data is still unmatched, it can add further unmatched members to the dimension reference data.

In order to add unmatched members using a lock, Data Manager performs the following checks on the template table:

- Determine the existence of an unmatched member.
• Determine the highest surrogate value.

In the example above, using a lock, build A prevents builds B and C from accessing the template table while it adds the unmatched member P5 to it. When it has finished, build B reads the template table and now finds P5, so it no longer reports any errors. In addition, build C can successfully add the unmatched member P6 to the template table.

If simultaneous builds are executed on multiple computers, a database table must be used as the lock. If simultaneous builds are executed on a single computer, a MUTEX/semaphore can be used instead of a table.

When you create a table for the lock, it must exist on the same connection as the template table. The table must comprise a single numeric column and a single row (unless you are using Netezza, in which case the table must have two columns).

This example shows a single numeric column and a single row:

```
CREATE TABLE product_lock ( lockcol integer );
INSERT INTO product_lock ( lockcol ) VALUES ( 0 );
```

This example shows how two columns could be used for Netezza:

```
CREATE TABLE product_lock ( keycol integer, lockcol integer);
INSERT INTO product_lock ( lockcol ) VALUES ( 0 );
```

Note: You can set up a single lock for all dimensions, or you can have separate locks for each one. To enable the addition of unmatched members from multiple fact builds, you must

• set the variable named DM_DISTRIBUTED_HANDLING

    For more information, see "DM_DISTRIBUTED_HANDLING" (p. 325).

• specify the name of the lock to use when adding unmatched members from distributed fact builds

    For more information, see "Set Up Data Access" (p. 99).

**Performance Considerations**

Because distributed handling of unmatched members requires additional processing, you should take into consideration how it can affect performance:

• If your source data contains a large number of unmatched members, using exclusive locking can improve performance.
If your source data contains unmatched members for different dimensions, using separate locks for each dimension, instead of a single lock, can improve performance.

Reference Domains

In IBM® Cognos® Data Manager, a reference domain is assembled from all the members in the hierarchy or lookup associated with the dimension, regardless of whether the fact data includes them.

You must always use a reference domain for dimensions that have an associated aggregation exception. This is because for each summary data value an aggregation exception returns the value of the first or last child member. Therefore, to identify the first and last child members, Data Manager must be aware of all the dimension members that exist in the domain.

Specify the Domain Details

You can specify a domain size for a dimension element or derived dimension element to size internal transactions when the fact build is executed. By default, IBM® Cognos® Data Manager estimates the domain size by referring to the reference structure associated with the dimension or derived dimension. However, if the fact data includes unmatched members, that is, data for which there is no associated reference member, the estimate may be incorrect. Equally, the reference domain may be a large overestimate of the actual domain. In either of these cases, you can enter a better estimate. The domain size should be greater than the maximum number of distinct values in the dimension. If, when the build is executed, the domain size is inadequate, a message appears indicating this, and you should increase the size.

Note: If you want to merge dimension elements that are not associated with a reference dimension, you must set the domain size manually. Set the domain size to be greater than the maximum number of distinct domain members for the dimension.

Steps

1. Click the required dimension element or derived dimension element.
2. From the Edit menu, click Properties.
3. Click the Domain tab.
4. In the Domain size box, type the required domain size.
5. In the Domain type box, click the type of domain to use. Remember, for delivery of an aggregation exception for the dimension, you must select Reference domain.
6. Click OK.

Aggregate Data

In IBM® Cognos® Data Manager, aggregation is the process of summarizing detail data to levels of a hierarchy or auto-level hierarchy. You can aggregate measure or derivation elements. If you aggregate attribute elements in summarized rows, Data Manager always returns a null value.
You can

- aggregate data to any level of any dimension
- aggregate data simultaneously over a number of levels (multi-level aggregation), a number of dimensions (multi-dimensional aggregation), or both
- exclude detail data from the output to provide compact summary data collections
- include or exclude individual levels

For example, you can include every conceivable combination of summary data of an in-depth business analysis, or just a high-level summary for management reporting.

Before you can aggregate data, you must
- determine the aggregation levels
- specify that aggregation is to be performed
- set up the regular aggregate
- set up aggregate rules, if required

**Memory Considerations**

In IBM® Cognos® Data Manager, aggregation is performed in memory, by default. You can limit the amount of memory that Data Manager uses. For example, when you aggregate large amounts of data, you may need to introduce breaking to control the aggregation and merge process. For more information about managing memory, see "Managing Memory" (p. 377).

**Determine the Aggregation Levels**

IBM® Cognos® Data Manager can build as many levels of output data as necessary. For example, a fiscal calendar where months roll to quarters and quarters roll to years means that quarterly and yearly data can be generated from the monthly data. In a multi-dimensional model, levels in all dimensions can be aggregated in this way, including all the combinations of levels across dimensions. Aggregating everything can lead to data explosion; that is, a moderate starting volume of data results in a huge final volume.

There are many factors to consider when deciding whether to aggregate to a level, many of which will be business related. For example, 'Should I aggregate to the market segment level when it gets reorganized every other day?'

**Specify that Aggregation is to be Performed**

Aggregation cannot be performed until you have specified a reference dimension and reference structure for the dimension element or derived dimension.

**Steps**

1. Click the dimension element or the derived dimension element.
2. From the Edit menu, click Properties.
3. Click the Reference tab.

4. Check that you have specified the reference dimension and reference structure to use.
   
   For information, see "Associate a Dimension or Derived Dimension Element with a Reference Item" (p. 171).

5. Select the Aggregate check box to allow aggregation to be performed.

6. Click OK.

**Calculate a Derivation Before Aggregation**

By default, a derivation element is calculated after aggregation. However, you can choose to perform calculation before aggregation; that is, from the input values.

You can also add derivations before transformation, in the data sources and DataStream of a fact build. For information, see "Derivations" (p. 155).

**Steps**

1. Click the derivation.

2. From the Edit menu, click Properties.

3. Click the General tab.

4. Select the Calculate at source check box.

5. Click OK.

**Set Up the Regular Aggregate**

The regular aggregate is applied to the appropriate measure or derivation element when aggregating over any dimension for which there is no aggregate rule defined. By default, IBM® Cognos® Data Manager aggregates over these dimensions before aggregating over any dimensions with aggregate rules.

For more information about aggregate rules, see "Set Up an Aggregate Rule" (p. 188).

You can aggregate data in a number of different ways. For example, you may want to find the sum of all detail members (SUM) or the average of all these members (AVG). Data Manager assigns SUM as the default regular aggregate.

The following aggregate functions can be used for the regular aggregate.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>Calculates a summary data value by adding together the detail values.</td>
</tr>
<tr>
<td>MIN</td>
<td>Assigns the minimum detail value to the summary data value.</td>
</tr>
<tr>
<td>MAX</td>
<td>Assigns the maximum detail value to the summary data value.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AVG</td>
<td>Calculates a summary data value by taking the average of the detail values.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Assigns the count of the number of detail values to the summary data value.</td>
</tr>
<tr>
<td>COUNT DISTINCT</td>
<td>Assigns the count of the number of unique detail values to the summary data value.</td>
</tr>
<tr>
<td>VAR</td>
<td>Calculates a summary data value by taking the variance of the detail values.</td>
</tr>
<tr>
<td>STDDEV</td>
<td>Calculates a summary data value by taking the standard deviation of the detail values.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Takes the value from the first detail row.</td>
</tr>
<tr>
<td></td>
<td>Note: This differs from using FIRST for an aggregate rule, which takes the value corresponding to the earliest row that occurs in the reference structure.</td>
</tr>
<tr>
<td>FIRST NON-NULL</td>
<td>Takes the value from the first non-null detail row.</td>
</tr>
<tr>
<td>LAST</td>
<td>Takes the value from the last detail row.</td>
</tr>
<tr>
<td></td>
<td>Note: This differs from using LAST for an aggregate rule, which takes the value corresponding to the latest row that occurs in the reference structure.</td>
</tr>
<tr>
<td>LAST NON-NULL</td>
<td>Takes the value from the last non-null detail row.</td>
</tr>
<tr>
<td>ANY</td>
<td>Sets the summary value to one if there are any detail values. Otherwise, the summary value is set to zero.</td>
</tr>
<tr>
<td>NONE</td>
<td>Sets the summary value to NULL regardless of any detail values.</td>
</tr>
</tbody>
</table>

**Example of the First and Last Aggregate Functions for the Regular Aggregate**

Suppose you have the following Time hierarchy.

Your incoming data looks like this.
If you aggregate Quantity over Time to the Quarter level using the FIRST aggregate function, these are the results.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Lantern</td>
<td>NULL</td>
</tr>
<tr>
<td>Q1</td>
<td>Tent</td>
<td>50</td>
</tr>
<tr>
<td>Q2</td>
<td>Lantern</td>
<td>10</td>
</tr>
</tbody>
</table>

Data Manager calculates the FIRST regular aggregate according to the order of the incoming data. So in Q1, Feb is the first record found for Lantern, and Mar is the first record found for Tent. Similarly, in Q2, Jun is the first record found for Lantern.

If you aggregate the same data using the FIRST NON-NULL aggregate function, the results are different.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Lantern</td>
<td>40</td>
</tr>
<tr>
<td>Q1</td>
<td>Tent</td>
<td>50</td>
</tr>
<tr>
<td>Q2</td>
<td>Lantern</td>
<td>10</td>
</tr>
</tbody>
</table>

In this case, Data Manager ignores the Feb record for Lantern because it contains a null value, and finds Jan as the first record instead. All other records remain unchanged because there are no other null values.

Similarly, aggregating Quantity over Time to the Quarter level using the LAST aggregate function produces these results.
<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Lantern</td>
<td>40</td>
</tr>
<tr>
<td>Q1</td>
<td>Tent</td>
<td>50</td>
</tr>
<tr>
<td>Q2</td>
<td>Lantern</td>
<td>30</td>
</tr>
</tbody>
</table>

In Q1, Jan is the last record found for Lantern, and Mar is the last record found for Tent. For Q2, Apr is the last record found for Lantern.

In this case, aggregating this data using the LAST NON-NULL aggregate function produces the same results because there are no null values.

**Steps**

1. Click the measure or derivation element for which to perform aggregation.

2. From the **Edit** menu, click **Properties**.

   To perform aggregation for a derivation, you must select the **Calculate at source** check box. For more information, see "Calculate a Derivation Before Aggregation" (p. 185).

3. Click the **Aggregation** tab.

4. In the **Regular aggregate** box, click the required aggregate function.

5. Click OK.

**Set Up an Aggregate Rule**

By default, the regular aggregate is applied across every dimension. However, if a measure or derivation element is semi-additive, there may be dimensions across which the function produces meaningless results, depending which aggregate function is used for the regular aggregate.

For example, *inventory* and *head count* are not additive over Time because they represent a snapshot value for a specific point in time. If SUM is used as the aggregate function for the regular aggregate, and a Time dimension is present, an alternative way of aggregating over Time is required.

In IBM® Cognos® Data Manager, an aggregate rule is the mechanism for specifying alternatives on a dimension-by-dimension basis.

To illustrate further, the following data includes an inventory value that represents the stock held at a particular time.
If you define the regular aggregate to SUM Inventory over Product, Location, and Time, the summary inventory value over time is meaningless. There are three consecutive months in which the stock information for Lantern is 50, 50, and 150 units. The sum for the quarter would be calculated as 250 units, which is much higher than the stock level at any time. This is clearly incorrect.

As an alternative, you can set up an aggregate rule for the Time dimension. For example, you could use the LAST aggregate function to select the last available month in the quarter as the summary value for the quarter.

Aggregate rules are used in addition to the regular aggregate and are applied in the following order:

1. The regular aggregate function is applied to all dimensions for which there is no aggregate rule defined.

2. Aggregate rules are applied to the specified dimensions, in sequential order.

Notes

- You can set up a separate aggregate rule for each dimension, but each dimension can only contain a single rule.

- The order of aggregate rules affects calculations because the result of each rule in a sequence is passed to the next rule.

- If an aggregate rule is set up for every dimension, the regular aggregate function is not applied.

- Aggregate rules cannot be used with multi-level input data or dimension breaks.

- If you are using aggregate rules with duplicate input data, you cannot allow records with duplicate keys. You must reject, merge or aggregate them.

  For more information, see "Handle Duplicate Data" (p. 138).

Aggregate Functions for an Aggregate Rule

The following aggregate functions can be used for an aggregate rule.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>Calculates a summary data value by adding together the detail values.</td>
</tr>
<tr>
<td>MIN</td>
<td>Assigns the minimum detail value to the summary data value.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------------------------</td>
</tr>
<tr>
<td>MAX</td>
<td>Assigns the maximum detail value to the summary data value.</td>
</tr>
<tr>
<td>AVG</td>
<td>Calculates a summary data value by taking the average of the detail values.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Takes the value corresponding to the earliest row that occurs in the reference structure.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This differs from using FIRST for the regular aggregate, which takes the value from the first detail row.</td>
</tr>
<tr>
<td>LAST</td>
<td>Takes the value corresponding to the latest row that occurs in the reference structure.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This differs from using LAST for the regular aggregate, which takes the value from the first detail row.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Assigns the count of the number of detail values to the summary data value.</td>
</tr>
<tr>
<td>COUNT DISTINCT</td>
<td>Assigns the count of the number of unique detail values to the summary data value.</td>
</tr>
<tr>
<td>VAR</td>
<td>Calculates a summary data value by taking the variance of the detail values.</td>
</tr>
<tr>
<td>STDDEV</td>
<td>Calculates a summary data value by taking the standard deviation of the detail values.</td>
</tr>
<tr>
<td>ANY</td>
<td>Sets the summary value to one if there are any detail values. Otherwise, the summary value is set to zero.</td>
</tr>
<tr>
<td>NONE</td>
<td>Sets the summary value to NULL regardless of any detail values.</td>
</tr>
</tbody>
</table>

**Example of the First and Last Aggregate Functions for the Aggregate Rule**

Suppose you have the following Time hierarchy.

![Time Hierarchy Diagram]

Your incoming data looks like this.
If you aggregate Quantity over Time to the Quarter level using the FIRST aggregate function, these are the results.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Lantern</td>
<td>NULL</td>
</tr>
<tr>
<td>Q1</td>
<td>Tent</td>
<td>50</td>
</tr>
<tr>
<td>Q2</td>
<td>Lantern</td>
<td>30</td>
</tr>
</tbody>
</table>

In Q1, Jan is the first row that occurs in the Time hierarchy for which a record exists for Lantern, and Mar is the first row that occurs for which a record exists for Tent. Likewise, in Q2, Apr is the first row for which a record exists for Lantern.

Similarly, aggregating Quantity over Time to the Quarter level using the LAST aggregate function, produces these results:

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Lantern</td>
<td>40</td>
</tr>
<tr>
<td>Q1</td>
<td>Tent</td>
<td>50</td>
</tr>
<tr>
<td>Q2</td>
<td>Lantern</td>
<td>10</td>
</tr>
</tbody>
</table>

In Q1, Feb is the last row that occurs in the Time hierarchy for which a record exists for Lantern, and Mar is the last row that occurs for which a record exists for Tent. In Q2, Jun is the last row for which a record exists for Lantern.

**Steps**

1. Click the appropriate measure or derivation element.

2. From the Edit menu, click Properties.
3. Click the Aggregation tab.

4. Click Add.

5. Select the dimension for which you want to set up an aggregate rule, and then click OK.

6. Click the required aggregate function from the Rule aggregate drop down list.

7. Repeat steps 4 to 6 to add more aggregate rules.

8. Click OK.

Tips

- The order in which aggregate rules are shown reflects the order in which they are applied. To change the position of a specific aggregate rule, select it and click either Up or Down.

- To delete an aggregate rule, select it and click Delete.

**Aggregate Duplicate Data**

When aggregating data, duplicate keys in the input data can produce different results depending on how the duplicates are handled:

- **Allow records with duplicate keys** has no cost in terms of memory or performance. It can be used when detail data is not being delivered or you know that duplicates will not occur. If it is used under other circumstances, IBM® Cognos® Data Manager may deliver multiple detail data rows rather than a single aggregated detail row.

- **Reject records with duplicate keys** can be used under any circumstances because it ensures that only one row with any duplicate key is passed to the aggregation phase.

- **Merge records with duplicate keys** is performed as a separate phase prior to the aggregation phase.

  Tip: To ensure you obtain the results you require, compare the results of merging duplicate input data with aggregating it.

- **Aggregate records with duplicate keys** passes duplicate input data directly to the aggregation phase.

For more information about duplicate data, see "Handle Duplicate Data" (p. 138).

Note: If you are aggregating data using aggregate rules, you cannot allow records with duplicate keys. You must reject, merge or aggregate them.

This example illustrates the difference between merging and aggregating input data with duplicate keys. You want to aggregate the following data to the Product Type level (Camping) using the COUNT aggregate function.

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lantern</td>
<td>40</td>
</tr>
</tbody>
</table>
If you choose to merge records with duplicate keys, Data Manager merges the three records for Lantern into a single record.

<table>
<thead>
<tr>
<th>Product</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lantern</td>
<td>70</td>
</tr>
<tr>
<td>Lantern</td>
<td>20</td>
</tr>
</tbody>
</table>

Data Manager passes the results of the merge phase to aggregation phase. Aggregating the data to the Product Type level produces the following results.

<table>
<thead>
<tr>
<th>Product</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lantern</td>
<td>3</td>
</tr>
<tr>
<td>Camping</td>
<td>1</td>
</tr>
</tbody>
</table>

If you choose to aggregate records with duplicate keys, aggregating the same data produces different results (shown below) because Data Manager performs only the aggregation phase.

<table>
<thead>
<tr>
<th>Product</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lantern</td>
<td>3</td>
</tr>
<tr>
<td>Camping</td>
<td>3</td>
</tr>
</tbody>
</table>

**Steps**

1. Click the required fact build.
2. From the Edit menu, click Properties.
3. Click the Input tab.
4. Select the option to perform on records with duplicate keys.
5. Click OK.

**Regular Aggregate Examples**

Suppose you have Time and Location hierarchies.
The detail data looks like this.

<table>
<thead>
<tr>
<th>Month</th>
<th>City</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>London</td>
<td>2,000</td>
</tr>
<tr>
<td>Jan</td>
<td>Manchester</td>
<td>1,000</td>
</tr>
<tr>
<td>Jan</td>
<td>Paris</td>
<td>1,500</td>
</tr>
<tr>
<td>Feb</td>
<td>London</td>
<td>2,500</td>
</tr>
<tr>
<td>Feb</td>
<td>Manchester</td>
<td>2,000</td>
</tr>
<tr>
<td>Feb</td>
<td>Paris</td>
<td>1,000</td>
</tr>
<tr>
<td>Mar</td>
<td>London</td>
<td>3,000</td>
</tr>
<tr>
<td>Mar</td>
<td>Paris</td>
<td>2,000</td>
</tr>
</tbody>
</table>

**Example 1**

In this example, you aggregate data to Country level using the SUM aggregate function. That is, the revenue for London and Manchester is combined to make England and, because the data only contains information about one city in France (Paris), the revenue for Paris simply becomes France.

<table>
<thead>
<tr>
<th>Month</th>
<th>Country</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>England</td>
<td>3,000</td>
</tr>
<tr>
<td>Jan</td>
<td>France</td>
<td>1,500</td>
</tr>
<tr>
<td>Feb</td>
<td>England</td>
<td>4,500</td>
</tr>
<tr>
<td>Feb</td>
<td>France</td>
<td>1,000</td>
</tr>
<tr>
<td>Mar</td>
<td>England</td>
<td>3,000</td>
</tr>
<tr>
<td>Mar</td>
<td>France</td>
<td>2,000</td>
</tr>
</tbody>
</table>
**Example 2**

In this example, you aggregate data to the Quarter level and the European level using the SUM aggregate function. That is, the revenue for Jan, Feb, and Mar is combined to obtain a revenue value for each level at Q1.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Location</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>London</td>
<td>7,500</td>
</tr>
<tr>
<td>Q1</td>
<td>Manchester</td>
<td>3,000</td>
</tr>
<tr>
<td>Q1</td>
<td>Paris</td>
<td>4,500</td>
</tr>
<tr>
<td>Q1</td>
<td>England</td>
<td>10,500</td>
</tr>
<tr>
<td>Q1</td>
<td>France</td>
<td>4,500</td>
</tr>
<tr>
<td>Q1</td>
<td>Europe</td>
<td>15,000</td>
</tr>
</tbody>
</table>

**Aggregate Rules Example**

You have Location, Product, and Time hierarchies.

The detail data looks like this.

<table>
<thead>
<tr>
<th>Month</th>
<th>Store</th>
<th>Product</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Shop A</td>
<td>Lantern</td>
<td>50</td>
</tr>
<tr>
<td>Jan</td>
<td>Shop B</td>
<td>Lantern</td>
<td>80</td>
</tr>
<tr>
<td>Jan</td>
<td>Shop A</td>
<td>Tent</td>
<td>150</td>
</tr>
<tr>
<td>Jan</td>
<td>Shop B</td>
<td>Tent</td>
<td>10</td>
</tr>
<tr>
<td>Mar</td>
<td>Shop A</td>
<td>Lantern</td>
<td>40</td>
</tr>
<tr>
<td>Mar</td>
<td>Shop B</td>
<td>Lantern</td>
<td>30</td>
</tr>
<tr>
<td>Mar</td>
<td>Shop A</td>
<td>Tent</td>
<td>60</td>
</tr>
</tbody>
</table>

You want to aggregate this data as follows:
1. Find the total number of products sold in each region.
2. Calculate the average number of products sold for each product type.
3. Determine the maximum quantity of each product sold in each quarter.

To do this, you set up a regular aggregate using the SUM aggregate function. You also need to set up two aggregate rules: one to aggregate the Product dimension using the AVG aggregate function, and a second to aggregate the Time dimension using the MAX aggregate function.

The regular aggregate (aggregate Quantity over Location to the Region level using the SUM aggregate function) is the first calculation to be performed.

<table>
<thead>
<tr>
<th>Month</th>
<th>Region</th>
<th>Product</th>
<th>Total Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>London</td>
<td>Lantern</td>
<td>130</td>
</tr>
<tr>
<td>Jan</td>
<td>London</td>
<td>Tent</td>
<td>160</td>
</tr>
<tr>
<td>Mar</td>
<td>London</td>
<td>Lantern</td>
<td>70</td>
</tr>
<tr>
<td>Mar</td>
<td>London</td>
<td>Tent</td>
<td>60</td>
</tr>
</tbody>
</table>

These results are now used in all subsequent calculations.

The first aggregate rule (aggregate Quantity over Product to the Product Type level using the AVG aggregate function) is the next calculation.

<table>
<thead>
<tr>
<th>Month</th>
<th>Store</th>
<th>Product Type</th>
<th>Average Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>Shop A</td>
<td>Camping</td>
<td>100</td>
</tr>
<tr>
<td>Jan</td>
<td>Shop B</td>
<td>Camping</td>
<td>45</td>
</tr>
<tr>
<td>Jan</td>
<td>London</td>
<td>Camping</td>
<td>145</td>
</tr>
<tr>
<td>Mar</td>
<td>Shop A</td>
<td>Camping</td>
<td>50</td>
</tr>
<tr>
<td>Mar</td>
<td>Shop B</td>
<td>Camping</td>
<td>30</td>
</tr>
<tr>
<td>Mar</td>
<td>London</td>
<td>Camping</td>
<td>65</td>
</tr>
</tbody>
</table>

These results are passed through to the final aggregate rule (aggregate Quantity over Time to the Quarter level using the MAX aggregate function).

<table>
<thead>
<tr>
<th>Month</th>
<th>Store</th>
<th>Product Type</th>
<th>Maximum Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Shop A</td>
<td>Lantern</td>
<td>50</td>
</tr>
<tr>
<td>Month</td>
<td>Store</td>
<td>Product Type</td>
<td>Maximum Quantity</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Q1</td>
<td>Shop B</td>
<td>Lantern</td>
<td>80</td>
</tr>
<tr>
<td>Q1</td>
<td>Shop A</td>
<td>Tent</td>
<td>150</td>
</tr>
<tr>
<td>Q1</td>
<td>Shop B</td>
<td>Tent</td>
<td>10</td>
</tr>
<tr>
<td>Q1</td>
<td>London</td>
<td>Lantern</td>
<td>130</td>
</tr>
<tr>
<td>Q1</td>
<td>London</td>
<td>Tent</td>
<td>160</td>
</tr>
<tr>
<td>Q1</td>
<td>Shop A</td>
<td>Camping</td>
<td>100</td>
</tr>
<tr>
<td>Q1</td>
<td>Shop B</td>
<td>Camping</td>
<td>45</td>
</tr>
<tr>
<td>Q1</td>
<td>London</td>
<td>Camping</td>
<td>145</td>
</tr>
</tbody>
</table>

Notice that the maximum quantity for some rows is higher than the quantity sold for any product in the input data. For example, in row 6, the maximum quantity of Tents sold is 160. This value represents the maximum of the total quantity of Tents sold.

**Process Late Arriving Facts**

IBM® Cognos® Data Manager normally delivers fact data by using the business key and retrieves only the current reference record from the reference data. If the fact data contains late arriving facts, the result may be that fact records are delivered with incorrect reference data.

For example, if sales records are delivered at the end of a month, and a reference record was updated during that month, fact records will be delivered using the current reference data.

By using late arriving fact processing, data can be delivered using the surrogate key instead of the business key. This has the advantage that, if the reference data tracks attribute changes, the history of changes to the data can be retrieved.

**Note:** You must load the dimensional history before you can process late arriving facts. For information, see "Set Up Dimensional History" (p. 242).

You can specify how late arriving facts are processed, so that fact records can be delivered using the appropriate surrogate key:

- If the timestamp in a fact record falls within the range of dates for the corresponding reference data, the appropriate data is selected from the reference data and the record is processed. You do not need to specify any further action to be performed on the record.

- If the timestamp in a fact record contains a null value, you can choose one of the following actions to perform on the record:
  - Reject fact record (default)
• Use current reference member

• If the timestamp in a fact record is outside of the effective date range for the reference data, you can select one of the following actions to perform on the record:
  • Reject fact record (default)
  • Use current reference member
  • Use closest reference member

To allow late arriving fact processing on a dimension, the following conditions must be met:
• The fact build must not merge data.
• The fact build must not include any dimension deliveries for the dimension.
• The dimension must use a lookup.
• The lookup must include an attribute with effective start date behavior.
• The lookup must use a template for data access.

**Steps**
1. Click the dimension element or derivation element for which you want to specify how to process late arriving facts.
2. From the **Edit** menu, click **Properties**.
3. Click the **Late Arriving Facts** tab.

![Late Arriving Facts Tab](image)

**Note:** If the conditions for processing late arriving facts are not met, a message appears indicating this, and the following steps are not available.

4. Select the **Enable late arriving fact processing** check box.
5. In the **Transaction date element** box, click the transformation model element that corresponds to the timestamp value in the fact records.
6. In the **Transaction date value actions** box, specify the action to perform on records containing late arriving facts.

7. The **Cache reference data history since** box allows you to limit the records processed for late arriving facts to a specific date range. This restricts the domain of reference members to include all current attribute change tracking records and those non-current records that are later than the specified date. Click
   - **All available** to specify that all available records should be processed
   - **Current record only** to specify that only current records should be processed
   - **Variable** to use the variable from which to obtain the date and time, and then in the adjacent box, type the variable name

   **Note:** The variable must be defined on the **Variables** tab. For information, see "Variables" (p. 319).

   - **Date (yyyy-mm-dd hh:mm:ss)** to specify an explicit date and time from which records should be processed, and then in the adjacent box, type the date and time

8. Click **OK**.

### Late Arriving Fact Example

Monthly sales figures for product P1 contain the date of each sale.

<table>
<thead>
<tr>
<th>Period</th>
<th>Product</th>
<th>Units Sold</th>
<th>Date of Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>200510</td>
<td>P1</td>
<td>1</td>
<td>2005/10/31</td>
</tr>
<tr>
<td>200601</td>
<td>P1</td>
<td>5</td>
<td>2006/01/31</td>
</tr>
<tr>
<td>200602</td>
<td>P1</td>
<td>13</td>
<td>2006/02/27</td>
</tr>
<tr>
<td>200603</td>
<td>P1</td>
<td>1</td>
<td>&lt;null&gt;</td>
</tr>
</tbody>
</table>

The following table illustrates Product reference data, where changes to the Specification attribute are being tracked.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Surrogate Key</th>
<th>Specification</th>
<th>Effective Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>Blue pen</td>
<td>2005/11/01</td>
<td>2005/12/22</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>Black pen</td>
<td>2005/12/23</td>
<td>2006/02/09</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>Red pen</td>
<td>2006/02/10</td>
<td></td>
</tr>
</tbody>
</table>
You can see that the specification for product P1 has changed since it was first defined and last changed on 2005/02/10. The range of dates for the Product reference data is 2005/11/01 to the current date.

Suppose all sales figures for the period October 2005 to March 2006 are delivered in March 2006. Because the product reference data has changed during this time period, the sales figures contain late arriving facts.

The timestamp for record 2 is 2006/01/31, which falls within the range of dates specified for the Product reference data. The fact record is delivered to the data mart with surrogate key 2.

The timestamp for record 4 is null. If you choose to use the current reference record, the fact record is delivered with surrogate key 3.

The timestamp for record 1 is 2005/10/31, which is outside the effective date range for the Product reference data. If you

- use the current reference record, the fact record with surrogate key 3 is delivered
- use the closest reference record, the fact record with surrogate key 1 is delivered

### Merge Data for a Measure or an Attribute

Merging is useful where more than one input record can have the same set of dimension values. It is also useful for merging disparate input data so that similar key sets of figures can be compared, for example, forecast and actual sales figures can be compared.

By default, IBM® Cognos® Data Manager assigns a merge method of SUM to measures and FIRST NON-NULL to attributes. However, you can change this by choosing from these merge methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>Adds together the duplicate detail values.</td>
</tr>
<tr>
<td>MAX</td>
<td>Takes the detail value with the maximum value.</td>
</tr>
<tr>
<td>MIN</td>
<td>Takes the detail value with the minimum value.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Counts the detail value members.</td>
</tr>
<tr>
<td>COUNT DISTINCT</td>
<td>Assigns the count of the number of unique detail values to the summary data value.</td>
</tr>
<tr>
<td>AVG</td>
<td>Averages the detail values.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Takes the value from the first detail row.</td>
</tr>
<tr>
<td>FIRST NON-NULL</td>
<td>Takes the value from the first non-null detail row.</td>
</tr>
<tr>
<td>LAST</td>
<td>Takes the value from the last detail row.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LAST NON-NULL</td>
<td>Takes the value from the last non-null detail row.</td>
</tr>
<tr>
<td>ANY</td>
<td>Sets the summary value to one if there are any detail values. Otherwise, the summary value is set to zero.</td>
</tr>
</tbody>
</table>

**Steps**

1. Click the measure or attribute element for which you want to perform merging.
2. From the Edit menu, click Properties.
3. Click the Merge tab.
4. In the Merge behavior box, click the required merge method.
   - **Note:** If the data is coming from multiple data sources, a merge operation is performed first, and then the transformation is specified in the fact build.
5. Click OK.
6. Click the fact build that contains the measure or attribute element for which you want to perform merging.
7. From the Edit menu, click Properties.
8. Click the Input tab.
9. Click Merge records with duplicate keys.
10. Click OK.
   - The Visualization pane is refreshed to show graphically that duplicate records are to be merged.

**Map the DataStream to the Transformation Model**

To enable the flow of data from a data source to the transformation model, you must

- map the acquired data source to the DataStream
- map the DataStream to existing transformation model elements or map the DataStream and create transformation model elements

The following example illustrates this data flow.
Map the DataStream to Existing Transformation Model Elements

After you add transformation model elements to the fact build, you must map DataStream items to them.

Tip: You can create attribute, measure, and dimension elements while performing mapping. For information, see "Map the DataStream and Create Transformation Model Elements" (p. 203).

Steps
1. Click the appropriate transformation model.
2. From the Edit menu, click Mapping.
   The Transformation Model Mapping window appears.
3. In the Transformation Model pane, click the transformation model element to map.
   If you are mapping a dimension element, expand the element and reference structure, and then click the required attribute ID.
4. In the DataStream Item column, click the DataStream item to map.
5. Click Map.
   Tip: You can also perform mapping by dragging the transformation model element to the relevant position in the Maps To column.
6. Repeat steps 3 to 5 to complete the required mapping.
7. Click OK.

Tips
- To clear a mapping, click the mapped item in the Maps To column, and then click Clear.
- To clear all the mappings, click Clear All.
- To add a transformation model element, click Add, and then click the type of element to create.
- To edit the name of a transformation model element, click the element in the Transformation Model pane, and then click Edit.
- To delete a transformation model element, click the element in the Transformation Model pane, and then click Delete.
Map the DataStream and Create Transformation Model Elements

Instead of creating transformation model elements and then mapping the DataStream items to the transformation model separately, you can perform both tasks together.

You can create attribute, dimension, and measure elements using this method, but not derivations or derived dimensions.

**Steps**

1. Click the appropriate transformation model.

2. From the **Edit** menu, click **Mapping**.
   
The **Transformation Model Mapping** window appears.

3. In the **DataStream Item** column, click the DataStream item that you want to map to the transformation model.

4. Drag the selected item to the white space in the **Transformation Model** column.
   
   **Note:** Be careful not to drag the item to any transformation model elements because this maps the DataStream item to the selected transformation model element.
   
   A menu appears prompting you to choose the element type to create.

5. Click the element type to create.
   
   **Note:** If a matching transformation model element already exists, a new element is not created, and the DataStream item is mapped to the existing one.
   
   The transformation model element is created and the selected item is mapped to the element.
   
   If you choose to create a dimension element, the element is not mapped. You must associate it with a reference item first. For information, see "Steps to Associate a Dimension Element With a Reference Item During the Mapping Process" (p. 172).

6. Repeat steps 3 to 5 to complete the required mapping.
   
   • To create multiple transformation model elements of the same type, hold down the Ctrl key and click each DataStream item, and then repeat steps 4 and 5.
   
   • To add a transformation model element, click **Add**, and then select the type of element to create.
   
   • To edit the name of a transformation model element, click the element in the **Transformation Model** pane, and then click **Edit**.
   
   • To delete a transformation model element, click the element in the **Transformation Model** pane, and then click **Delete**.

7. Click **OK**.
   
The **Add New Elements** window appears.

8. For each listed element, select the **Add to Fact Deliveries** check box to add the element to each fact delivery in the fact build.
9. Click OK.

**Map the DataStream to Transformation Model Elements Automatically**

You can map all unmapped DataStream items in a single step.

If transformation model elements exist with matching names, the DataStream items are automatically mapped to them. For example, if you have a DataStream item named OrderDate, and a transformation model element named OrderDate, they are automatically mapped.

If no matching transformation model elements exist, they are automatically created during the mapping process.

**Steps**

1. Click the appropriate transformation model.

2. From the Edit menu, click Mapping.

   The Transformation Model Mapping window appears.

3. Click Auto Map.

   If elements need to be created, a message appears prompting you to choose the element type to create.

   **Note:** All new elements are created with the same type.

   All the unmapped DataStream items are mapped.

   If you choose to create dimension elements, IBM Cognos Data Manager does not map the element. You must associate it with a reference item first. For information, see "Steps to Associate a Dimension Element With a Reference Item During the Mapping Process" (p. 172).

4. Click OK.

   If any new elements have been created, the Add New Elements window appears.

5. For each listed element, select the Add to Fact Deliveries check box to add the element to each fact delivery in the fact build.

6. Click OK.

**Add an Element to the Deliveries**

You can automatically add transformation model elements to the deliveries for the fact build.

**Steps**

1. Either add a new element to a fact build, or click an element created previously and, from the Actions menu, click Add to Delivery Modules.

   The Deliver <element_name> dialog box appears.

2. To add the element to each delivery module in the fact build, click Yes.
Change an Element Type

You can change the element type of any transformation model element.

Steps
1. Click the transformation model element to change to a different type.
2. From the Actions menu, click Convert, followed by the relevant option.

Reposition Elements

The order in which the transformation model elements are shown in the Tree pane is reflected in the fact columns identified in the fact deliveries. This affects the column order in the target table if it is created as part of the fact build, but does not affect the ability of IBM® Cognos® Data Manager to deliver data to the target table.

Another consideration when positioning elements is that when you create an expression for a derived dimension, in addition to using any measure, attribute, or dimension element in the fact build, you can also use any derived dimension that is before it in the fact build.

Steps to Reposition Elements From the Tree Pane
1. In the Tree pane, click the transformation model element to reposition.
2. From the Actions menu, click either Move Up or Move Down.
3. Click OK.
   Tip: You can also reposition an element by clicking the element in the Tree pane and dragging it to the required position.

Steps to Reposition Elements From the Transformation Model Mapping Window
1. Click the appropriate transformation model.
2. From the Edit menu, click Mapping.
   The Transformation Model Mapping window appears.
3. In the Transformation Model pane, click the transformation model element to reposition.
4. Click either Move Up or Move Down.
5. Click OK.

Delete an Element

You can delete a transformation model element that is no longer required.

Steps
1. Click the required element.
2. From the Edit menu, click Delete.

3. In the message box, click Yes.
Chapter 16: Delivering Fact Data Using a Fact Build

In IBM® Cognos® Data Manager, you use delivery modules to deliver transformed data to the specified data repositories. Data Manager organizes the delivery modules into these groups:

- fact delivery modules that deliver data produced by a fact build
- dimension delivery modules that deliver data describing a single dimension

Each fact build can have as many deliveries as you require.

Fact Delivery Modules

Fact deliveries deliver the fact data that a fact build produces.

You can deliver

- data to different tables within the same or different target databases, and to different OLAP targets.
- specific transformation model elements to multiple tables in the same or different databases by subscribing to elements.
- specific data rows by applying an output filter.
- specific dimensions and hierarchy levels by applying one or more level filters.
- columns that are only occasionally used in queries into a separate table by partitioning the data.

A number of fact delivery modules are available. These include delivery modules for relational tables, native bulk loaders for relational databases, OLAP targets, and text files. Each of these modules has different configuration options and is responsible for converting fact data into an appropriate format for the target.

The fact delivery modules fall into two groups, those that deliver the data to tables, and those that do not.

Table Delivery Modules

The following fact delivery modules deliver data to tables:

- DB2® LOAD
- Informix® LOAD
- Microsoft® SQL Server BCP Command
- Netezza NZLoad
- ORACLE SQL *Loader
Red Brick® Loader (TMU)
- Relational Table
- Microsoft® SQL Server Bulk Copy via API
- Sybase ASE
- Sybase IQ
- Teradata Fastload
- Teradata Multiloading
- Teradata TPump

**Non-table Delivery Modules**

The following fact delivery modules do not deliver data to tables:

- Essbase Direct Feed
- Essbase File Feed
- Text File
- TM1® Turbo Integrator

Each delivery module has a set of properties. Module properties apply to the delivery as a whole, for example, the table name of a target relational table. Element properties apply to a delivered element. Not all delivery elements have the same properties. The relevance of each property depends on the element type with which it is associated, that is, whether it is a dimension, attribute, measure, or derivation.

**Add a Fact Delivery**

You can add fact deliveries to a fact build manually or while creating a fact build using the Fact Build wizard.

For information about adding a fact delivery using the Fact Build wizard, see "The Fact Build Wizard" (p. 133).

**Steps**

1. In the appropriate fact build, expand **Delivery Modules** and click **Fact Delivery**.
2. From the **Insert** menu, click **Build**, and then click **Fact Delivery**.
3. From the **Fact Delivery** menu, click the delivery module that you want to add.
   - The **Properties** window for the selected delivery module appears.
4. Click the **General** tab.
   - **Note:** The set of tabs in this window varies according to the fact delivery module that you select.
5. Type a name for the delivery and, if required, a business name and description.

6. If you do not want to deliver the same data rows to any deliveries that are after this one in the top-down order of the delivery tree, select the Exclusive check box.

7. If you want to exclude the delivery when the build is executed, clear the Enabled check box. You can now temporarily disable the delivery without having to delete it, and later recreate it.

8. Click the Module Properties tab and enter values as required.
   For information about the properties for each delivery module, see "Fact Delivery Module Properties" (p. 455).

9. Click OK.

Define the Properties for a Table Delivery

If you select a delivery module that delivers data to tables, you can
● specify the target database and target tables
● specify how indexing is to be performed
● subscribe to and define the properties for transformation model elements
● change column names
● deliver columns across more than one table
● automatically add any columns that are missing from the target table
● specify how to handle failed data in a table delivery
● filter the delivery data
● partition the data so that columns that are only occasionally used in queries are delivered into separate tables

Steps
1. Click the fact delivery for which you want to set the table properties.

2. From the Edit menu, click Properties.

3. Click the Table Properties tab.

4. In the Connection box, click the connection that corresponds to the database in which the table resides.

5. In the Table name box, enter the name of the target table. This value is mandatory.
   If you enter the name of a table that does not exist, IBM Cognos Data Manager creates it.
   Tip: You can also click the Browse for Table button and choose the required table from the list in the Select Table dialog box.
6. Select the check boxes to the left of the transformation model elements to specify the elements to which the delivery subscribes, that is, which elements are to be delivered. For more information, see "Deliver to Multiple Tables by Subscribing to Elements" (p. 212).

7. To change the name of a column, click the column name, and type the new name. For more information, see "Change the Column Names of Delivered Data" (p. 215).

8. Select the check boxes in the Key column to indicate which columns form part of the primary key of the table.

   If there are dimension elements included in the delivery table, by default, Data Manager selects them all as the key columns if no other columns are currently selected. To use specific dimension elements only, select the relevant check boxes.

   If there are no dimension elements in the delivery table, and you do not specify your own key columns, Data Manager cannot determine which columns already exist. They are therefore are treated as columns to be inserted.

   For information about key columns for the Relational Table delivery module, see "Primary Key Settings for Update/Insert" (p. 477) and "Primary Key Settings for Update" (p. 484).

9. Select the check boxes in the Update column to indicate which columns Data Manager should update in the target table.

   By default, when Data Manager performs an update, it updates all columns in the target table, with the exception of any columns identified as key columns.

   ● You cannot select a column that is identified as the key. If you need to update it, you must clear the Key column check box first.

   ● Performing an update only applies if you use the Relational Table or the Red Brick® Loader (TMU) delivery modules.

10. To create an index for all the dimension columns in the table (a composite index), select the Index check box.

    You can define the properties for the index by clicking Index Properties.

    For more information, see "Define the Index Properties" (p. 211).

11. To create an index for an element (as opposed to creating a composite index for the whole delivery), select the relevant check box in the Index column.

    You can define the properties for the index by clicking the Browse button in the Index Properties column.

    For more information, see "Define the Index Properties" (p. 211).

12. If you want Data Manager to automatically add missing columns to the target table, select the Automatically add columns to table check box.

    For more information, see "Add Missing Columns to the Target Table" (p. 215).

13. To define the properties for an element, click the element, and then click the button in the Element Properties column.
For more information, see "Define the Properties for a Subscribed Element" (p. 214).

14. To add control attributes to the table, select the relevant check boxes as required:
   - Create Date
   - Last Update Date
   - Record Identity

For more information, see "Add Control Attributes to a Table Delivery" (p. 212).

15. Click OK.

Define the Index Properties

You can define the index properties for all the dimension columns in a table (a composite index), or for a single transformation model element (an element index).

Steps

1. Click the fact delivery for which you want to set the table properties.

2. From the Edit menu, click Properties.

3. Click the Table Properties tab.

4. Define the properties for a composite index, an element index, or both.
   - To create an index for all the dimension columns in the table (a composite index), select the Index check box, and then click Index Properties.
     - To create an index for a single element, as opposed to creating a composite index for the whole delivery, select the relevant check box in the Index column, and then click the Browse button in the Index Properties column.

5. In the Name box, type a name for the index.

6. In the Type box, click the type of index to create:
Click **Unique** for an index that constrains the dimension data so that each value in the indexed column is unique.

Click **Bitmap** for an index that is of a compressed format, which some database management systems implement. Using an index of this type can enhance search performance.

For more information, see the manufacturer's documentation for the type of database to which you are delivering.

If the **Type** box is empty, the delivery creates the default index type for the target DBMS.

7. In the **Options** box, type any additional options for the index. The contents of this box are appended to the CREATE INDEX statement.

For information about valid options, see the documentation for the target DBMS.

8. To continue generating the index if errors are encountered, select the **Suppress errors** check box.

9. To drop the indexes on the target table before loading the data, and recreate them when the load is complete, select the **Recreate index** check box.

When a large amount of data is to be loaded, recreating the indexes after loading greatly improves the performance of the load because the indexes do not require maintenance during loading.

**Note:** If a load requires some data to be updated rather than just inserted, do not drop the indexes because they are required to support the update operation. Removing indexes that support updates results in very slow performance. If you require specific index management, consider using an SQL JobStream node.

10. Click **OK**.

## Add Control Attributes to a Table Delivery

You can add the following special control attributes to a fact delivery table:

- **Create Date** is the date on which a new row was inserted into the target table
- **Last Update Date** is the date on which an existing row was updated

For more information, see "Control Attributes" (p. 480).

- **Record Identity** is a unique number assigned to each row that indicates that the fact delivery data is partitioned across more than one table

For more information, see "Partition Columns Horizontally Across a Table" (p. 221).

## Deliver to Multiple Tables by Subscribing to Elements

For each delivery, you can choose the transformation model elements to which each delivery subscribes by subscribing to the elements to deliver. For example, you could deliver to two tables as follows:
• Delivery1 for period_no, state_cd, product_cd, units_sold, units_forecast
• Delivery2 for period_no, state_cd, product_cd, revenue, revenue_forecast

This separates the data so that sales data and revenue data are delivered to different targets.

How you do this depends on whether the selected delivery module delivers data to a table. For information, see "Fact Delivery Modules" (p. 207).

Steps for Table Deliveries
1. Click the required fact delivery.

2. From the Edit menu, click Properties.

3. Click the Table Properties tab.

To the left of each transformation model element name is a check box. By default, these check boxes are all selected so each element is delivered.

4. Clear the check box for any element that you do not want to deliver.

5. Move the elements to the required position.

The order in which the subscribed elements appear in this window determines their order in the target table.

6. Click OK.

Steps for Non-Table Deliveries
1. Click the required fact delivery.

2. From the Edit menu, click Properties.

3. Click the Element Properties tab.

The Available Elements pane shows all the elements in the transformation model. The Subscribed Elements pane shows the list of transformation model elements to which the delivery subscribes. This includes all the elements by default. Only values from the subscribed elements list are delivered.

4. For each element that you do not want to deliver, right-click the element in either pane, and click Unsubscribe.

The element disappears from the Subscribed Elements pane.
5. Move the elements to the required position.
   The order in which the subscribed elements appear in this window determines their order in
   the target table.

6. Click OK.

Define the Properties for a Subscribed Element

You can, for most delivery modules, define properties at the element level and for the delivery
module as a whole. How you do this depends on whether the selected delivery module delivers data
to a table.

Steps for Table Deliveries

1. Click the fact delivery for which you want to set the transformation model element properties.

2. From the Edit menu, click Properties.

3. Click the Table Properties tab.

4. Click the transformation model element for which you want to set properties, and then click
   the Browse button in the Element Properties column.

   The Delivery Element Properties dialog box appears.

5. Enter values as required.

   For information about the available properties, see "Fact Delivery Module Properties" (p. 455).

6. Click OK.

Steps for Non-Table Deliveries

1. Click the fact delivery for which you want to set the element properties.

2. From the Edit menu, click Properties.

3. Click the Element Properties tab.

4. In the Subscribed Elements pane, right-click the transformation model element for which you
   want to set properties, and click Properties.

   The Delivery Element Properties dialog box appears.

5. Enter values as required.

   For information about the available properties, see "Fact Delivery Module Properties" (p. 455).

6. Click OK.

   The Subscribed Elements pane is updated to show that you added element information.

Example

In this example, values have been entered in the Field delimiter and Date format boxes.
Change the Column Names of Delivered Data

By default, element names are used for column names when data is delivered to a table. However, you can specify different column names.

Note: Changing a column name is only applicable to table delivery modules.

Steps
1. Click the required fact delivery.
2. From the Edit menu, click Properties.
   The Table Delivery Properties window appears.
3. Click the Table Properties tab.
4. In the Column Name column, click the name to change, and then type the new name.
5. Click OK.

Add Missing Columns to the Target Table

When IBM® Cognos® Data Manager delivers data, all the columns in the source table must exist in the target table. Data Manager determines this by checking the tables during execution.

If they are different, and you are

- executing from the Designer, a message appears informing you of the problem and you can drop the tables.

  For information on checking and dropping the target table, see "Check the Target Table Prior to Executing a Build" (p. 275).

- executing from the command line, no warning can be given and so the build fails.

You can specify that Data Manager automatically add any missing columns to the target table. The build will then not fail simply because the columns in the table are incompatible. When you select this option, Data Manager does not automatically check the table during execution; you must manually check the table.

Tip: To stop Data Manager automatically checking the tables during execution, from the Tools menu, click Options, and then click Skip table compatibility checks on builds. If you select this option and the tables are incompatible, the build fails.

Steps
1. Click the appropriate fact build delivery.
2. From the Edit menu, click Properties.
3. Either
   - In the Table Delivery Properties window, click the Table Properties tab, and then click the Automatically add columns to table check box.
   - In the Dimension Table Properties window, click the Table tab, and then click the Automatically add columns to table check box.

4. Click OK.

Handle Failed Rows in a Relational Table Delivery

By default, when delivering fact data to a relational table, if IBM® Cognos® Data Manager encounters an error in a row of fact data, it stops the delivery at that row and reports an exception error. No further rows are delivered and no detailed error reporting occurs for a failed row.

For a specific set of errors, if a row fails, you can configure Data Manager to report the error reason, together with the failed row data, without terminating the table delivery. These errors can include:

- unique constraint errors, such as duplicate primary keys
- integrity constraint errors, such as a failing foreign key
- bind errors
- non-unique key errors
- null key errors

You can select the way in which Data Manager handles rows that fail with these errors:

- **Don’t write failed rows** (default)
  Data Manager does not provide detailed error reporting for failed rows, and terminates the table delivery.

- **Write failed row to file**
  Data Manager writes row errors to a simple tab delimited text file and continues with the table delivery. The file is recreated every time the build is executed.

- **Write failed row to table**
  Data Manager writes row errors to a table and continues with the table delivery. The error table can reside in any writable Data Manager connection. Data Manager recreates this table each time the build is executed.

**Note**: If you are delivering fact data using a DBMS that supports array insert, such as Oracle, Data Manager is unable to report individual row errors. You can overcome this by setting the variable named DS_UDA_BULKWRITE_ROWS to 1. For information, see "DS_UDA_BULKWRITE_ROWS" (p. 328).

**Steps to Write Failed Records to File**

1. Click the appropriate fact build delivery.
2. From the Edit menu, click Properties.

3. Click the Failure Handling tab.

4. In the Failure Handling Method box, click Write failed rows to file.

5. In the File name box, enter the full directory path and name for error file to which you want to save failed rows.

6. If required, in the Failure limit box, specify the maximum number of errors to report before the table delivery should terminate.
   By default, the limit is set to zero, indicating that no failure limit is set.

7. Click OK.

**Steps to Write Failed Records to a Relational Table**

1. Click the appropriate fact build delivery.

2. From the Edit menu, click Properties.

3. Click the Failure Handling tab.

4. In the Failure Handling Method box, click Write failed rows to table.

5. In the Connection box, click the connection that contains the table where you want the details saved.
   Tip: To create a connection, click New. For more information, see "Create a Database Connection" (p. 63).

6. In the Table name box, enter the name of the table to which you want to save failed rows.

7. If required, in the Failure limit box, specify the maximum number of errors to report before the table delivery should terminate.
   By default, the limit is set to zero, indicating that no failure limit is set.

8. Click OK.

**Set Up Delivery Filters**

You can restrict the fact data to be delivered using a delivery filter. A delivery filter can contain
- one or more level filters
  Level filters allow you to restrict a delivery to specific dimensions and hierarchy levels.
- a single output filter
  Output filters allow you to restrict a delivery to specific data rows.

Each delivery filter you set up within a fact build is saved as a generic filter, and you can apply it to any fact delivery within the fact build.
Create a Delivery Filter

Before you can create a level filter or an output filter, you must create a delivery filter. You can then add the level and output filters that you require to the delivery filter.

You can

- create and apply a delivery filter within a specific fact delivery
- create a delivery filter that you can then apply to any fact deliveries within the selected fact build

Steps to Create a Delivery Filter

1. In the Fact Delivery folder, click Delivery Filters.

2. From the Insert menu, click Build, and then click Delivery Filter.

3. Type a name for the delivery filter and, if you require, a business name and description.

   You can now add level and output filters to the delivery filter.

   For more information, see "Use a Level Filter to Deliver Specific Dimensions and Hierarchy Levels" (p. 219) and "Use an Output Filter to Deliver Specific Data Rows" (p. 220).

4. When you have finished adding level and output filters to the delivery filter, click OK.

   The delivery filter is added to the list of delivery filters available in the fact build. To use a delivery filter, you must apply it to one or more fact deliveries. For more information, see "Apply a Delivery Filter to a Fact Delivery" (p. 221).

Steps to Create a Delivery Filter Within a Fact Delivery

1. Click the required fact delivery.

2. From the Edit menu, click Properties.

3. Click the Filters tab.

   Any defined delivery filters are listed in the Delivery filter box.

   **Tip:** To edit a delivery filter, click the required filter, and then click Edit.

4. Click New.

5. Type a name for the delivery filter and, if you require, a business name and description.

   You can now add level and output filters to the delivery filter.

   For more information, see "Use a Level Filter to Deliver Specific Dimensions and Hierarchy Levels" (p. 219) and "Use an Output Filter to Deliver Specific Data Rows" (p. 220).

6. When you have finished adding level and output filters to the delivery filter, click OK to close the Delivery Filter Properties window.

7. Click OK to close the Table Delivery Properties or the Properties window.

   The delivery filter is automatically applied to the fact delivery. It is also added to the list of delivery filters available in the fact build.
Use a Level Filter to Deliver Specific Dimensions and Hierarchy Levels

You can restrict a delivery to specific dimensions and hierarchy levels by using a level filter.

You can apply as many level filters to a fact delivery as you require. Create one level filter for each combination of dimension/hierarchy levels to deliver. Express these filters in the following format where element1, element2, and so on, are dimension elements, and level1, level2, and so on are hierarchy levels within the corresponding dimension.

\[
\text{element1.level1} \ast \text{element2.level2}[\ast \text{element3.level3} \ldots]
\]

The following is the syntax for a level filter where \(<\text{element}>\) is the name of a dimension element and \(<\text{level}>\) is the name of a level from the corresponding dimension. An asterisk (\(\ast\)) specifies the intersection of the levels it joins. You can specify more than one level of the same hierarchy by separating the level names with commas:

\[
<\text{element}>.<\text{level}>[<\text{level}>[\ast<\text{element}>.<\text{level}>[<\text{level}>[\ldots]]]
\]

For example, the following filter expression accepts data that lies at the intersection of the Product-Type level of the Product hierarchy and either the Year level or Quarter level of the Dates hierarchy.

\[
\text{ProductNumber.ProductType} \ast \text{OrderDate.Year,Quarter}
\]

For example, you could partition data into Period, Quarter, and Year hierarchical levels. To do this, create three deliveries, each having a level filter, and set the filters to period_no.Period, period_no.Quarter, and period_no.Year, respectively.

Steps
1. Open the Delivery Filter Properties window.
2. Click the Level Filters tab.
   Any defined level filters are listed.
   Tip: To edit a level filter, click the level filter, and then click Edit.
3. Click Add.
   The Level Filter Properties window appears.
4. Type an expression for the filter, or right-click, click Dimension Levels, and then click the levels to insert.
5. Click OK.

The level filter is added to the Level Filter tab in the Delivery Filter Properties window.

Tips
- To show the full text of each level filter listed, click Show All Expression Text.
- To delete a level filter, click the level filter, and then click Delete.

Use an Output Filter to Deliver Specific Data Rows

You can restrict a delivery to specific data rows using an output filter. Each delivery filter may have no more than one output filter.

An output filter is an expression that results in TRUE or FALSE when applied to each output row. The delivery only delivers those data rows for which the expression evaluates to TRUE.

Typically, you would use an output filter to vertically partition data other than by hierarchical levels. For example, you could partition data by sales. Two deliveries, one having the output filter units_sold > 0, and the other having the output filter units_sold = 0, would partition the data according to whether there were sales of that product.

Steps
1. Open the Delivery Filter Properties window.
2. Click the Output Filter tab.
3. In the right pane, enter an expression for the filter.

You can type the expression, and you can use the tree structure in the left pane to select the items from the following folders:
- Elements to select elements defined within the transformation model
- Reference Attributes to select a dimension and member combination
- **Operators** to select operators
- **Functions** to select predefined and user-defined functions
- **Control** to select controls
- **Variables** to select variables and substitution variables

For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305). For information about variables and substitution variables, see "Variables" (p. 319).

You can select an item from a folder by dragging it to the right pane, or by double-clicking it.

4. If required, test the expression. For information, see "Testing Expressions and Scripts" (p. 341).

5. Click OK.

### Apply a Delivery Filter to a Fact Delivery

You can apply a delivery filter to any fact delivery in the fact build.

**Note:** Any delivery filters that you create in a specific fact delivery are automatically saved as delivery filters and you can use them in other fact deliveries within the fact build.

**Steps**

1. Click the required fact delivery.

2. From the **Edit** menu, click **Properties**.

3. Click the **Filters** tab.

4. From the **Delivery filter** box, click the delivery filter that you want to apply.

5. Click **OK**.

### Delete a Delivery Filter

If you delete a delivery filter, all level and output filters defined within it are also deleted.

**Steps**

1. In the **Fact Delivery** folder, click **Delivery Filters**.

2. From the **Edit** menu, click **Delete**.

3. In the message box, click **Yes**.

### Partition Columns Horizontally Across a Table

If your fact data contains columns that are only occasionally used in queries, such as description columns, you can partition the data so that the columns are stored in a separate, but linked, table.
IBM® Cognos® Data Manager selects the Record Identity control attribute to indicate that the table is partitioned.

<table>
<thead>
<tr>
<th>Element</th>
<th>Column Name</th>
<th>Key</th>
<th>Update</th>
<th>Index</th>
<th>Index Properties</th>
<th>Element Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_DATE</td>
<td>ORDER_DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT_NUMBER</td>
<td>PRODUCT_NUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETAILER_SITE_CODE</td>
<td>RETAILER_SITE_CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALES_STAFF_CODE</td>
<td>SALES_STAFF_CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUANTITY</td>
<td>QUANTITY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_COST</td>
<td>UNIT_COST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_PRICE</td>
<td>UNIT_PRICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_SALE_PRICE</td>
<td>UNIT_SALE_PRICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Manager distinguishes between the primary and secondary tables using the Record_Identity column. If the Record_Identity column is not selected as the primary key, Data Manager knows that this is the primary partitioned table; if the Record_Identity column is selected as the primary key, Data Manager knows that this is a secondary partitioned table.

The relationship between the primary and secondary table partitions is determined by their position in the Tree pane. Data Manager assumes that tables with secondary partitioning are associated with the table directly above that has primary partitioning. For example

If you insert another delivery between the primary and secondary table deliveries, the table partitioning fails. For example

You can see how the partitioning will work from the Fact Delivery tab in the Visualization pane.

The following example shows a fact build that is to deliver to two tables, the primary partitioned table named F_Additional, and the secondary partitioned table named F_Additional_des. Data Manager allows you to distinguish between the primary and secondary table using a red number one to indicate the primary table, and a red number two to indicate the secondary table or tables. The linked Record_Identity columns show the link between the two tables.
If you apply a delivery filter to a table with primary partitioning, it also applies to any secondary tables. Secondary tables can also have delivery filters applied to them, but these do not affect the primary table. For information, see "Set Up Delivery Filters" (p. 217).

If you want to export metadata for a fact delivery table that has been partitioned, the partitioning is shown in the Metadata Star Properties window. For information, see "Create a Metadata Collection" (p. 298).

Partitioning data can result in improved query performance.

**Steps to Create a Table With Primary Partitioning**

1. Click the fact delivery for which you want to partition the data.

2. From the **Edit** menu, click **Properties**.

3. Click the **Table Properties** tab.

4. From the **Partition** box, click **Primary partition**.

The Record Identity control element is selected to indicate that this table is partitioned.

<table>
<thead>
<tr>
<th>Element</th>
<th>Column Name</th>
<th>Key</th>
<th>Update</th>
<th>Index</th>
<th>Index Properties</th>
<th>Element Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER_DATE</td>
<td>ORDER_DATE</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT_NUMBER</td>
<td>PRODUCT_NUMBER</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETAILER_SITE_CODE</td>
<td>RETAILER_SITE_CODE</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALES_STAFF_CODE</td>
<td>SALES_STAFF_CODE</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUANTITY</td>
<td>QUANTITY</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_COST</td>
<td>UNIT_COST</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_PRICE</td>
<td>UNIT_PRICE</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_SALE_PRICE</td>
<td>UNIT_SALE_PRICE</td>
<td>☑</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Create Date)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Last Update Date)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>![Record Identity]</td>
<td>Record_Identity</td>
<td>☑</td>
<td></td>
<td>☑</td>
<td>☑</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** In a primary table, you cannot select the Record_Identity column as the primary key.

To the left of each transformation model element name is a check box. By default, these check boxes are all selected, therefore each element is included in the table.

5. Clear the check box for any element to exclude from the primary partitioned table.

6. If required, move the selected elements to the required position.

The order in which the selected elements are shown in the target table is determined by their order in this window.
7. Click OK.

**Steps to Create a Table With Secondary Partitioning**

1. Click the fact delivery for which you want to partition the data.
2. From the Edit menu, click Properties.
3. Click the Table Properties tab.
4. From the Partition box, click Secondary partition.

The Record Identity control element indicates that this table is partitioned. Data Manager automatically selects the Record_Identity column as the primary key, and any columns that were previously selected as primary keys are cleared. Selecting the Record_Identity column as the primary key is mandatory in a secondary table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Column Name</th>
<th>Key</th>
<th>Update</th>
<th>Index</th>
<th>Index Properties</th>
<th>Element Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT_DATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT_NUMBER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETAILER_SITE_CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SALES_STAFF_CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUANTITY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_COST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_PRICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNIT_SALE_PRICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT_DESCRIPTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Create Data)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Last Update Date)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To the left of each transformation model element name is a check box. By default, these check boxes are all selected; each element is therefore included in the table.

5. Clear the check box of any element that you do not want to include in the delivery.

**Note:** You can include a column in more than one table.

6. If required, move the selected elements to the required position.

The order in which the selected elements are shown in the target table is determined by their order in this window.

7. Click OK.

**Disable and Enable a Fact Delivery**

At times, you may want to temporarily disable a delivery. Although you can delete the delivery, and later recreate it, a better option is to disable the delivery. You can then enable it when it is required.

For more information on deleting a fact delivery, see "Delete a Fact Delivery" (p. 225).

**Note:** If you disable a table that has a primary partition, its secondary tables are automatically disabled.

**Steps**

1. Click the required delivery.
2. From the Edit menu, click Properties.

3. Click the General tab.

4. Either clear or select the Enabled check box.

5. Click OK.

   **Tip:** You can also disable a delivery by clicking the delivery and then, from the Edit menu, clicking Enabled.

---

**Change the Delivery Module for a Fact Delivery**

If required, you can change the delivery module that delivers the fact data.

Each delivery module has a different set of properties. After changing the delivery module type, you must specify the module properties and element properties for the new delivery type. For information, see "Fact Delivery Module Properties" (p. 455).

**Steps**

1. Click the required fact delivery.

2. From the Actions menu, click Change Delivery Module.

3. From the Change to list, click the new delivery module.

4. Click OK.

---

**Delete a Fact Delivery**

You can delete a fact delivery that is no longer required.

**Steps**

1. Click the required delivery.

2. From the Edit menu, click Delete.

3. In the message box, click Yes.

   **Tip:** You may want to temporarily disable a delivery rather than delete it, and then perhaps later have to create it. For information, see "Disable and Enable a Dimension Delivery" (p. 229).
Chapter 16: Delivering Fact Data Using a Fact Build
Chapter 17: Delivering Dimension Data Using a Fact Build

In IBM® Cognos® Data Manager, you use delivery modules to deliver transformed data to the specified data repositories. Data Manager organizes the delivery modules into these groups:

- fact delivery modules that deliver data produced by a fact build
- dimension delivery modules that deliver data describing a single dimension

Each fact build can have as many deliveries as you require.

You can manually add a delivery module to a fact build, or you can use the Fact Build wizard. If you want to deliver partitioned data to different tables within different databases, you must do it manually.

Dimension Delivery Modules

Dimension deliveries deliver data that describes a single dimension. You can deliver the data to a single table (a star schema) or multiple tables (a snowflake schema). You can also deliver more than one dimension delivery per dimension (for example, two star schemas).

Although fact builds can deliver dimension data, they are not as flexible at managing dimensions as dimension builds. For information about when to use a dimension delivery in a fact build, and when to use a dimension build, see "Separate Reference Data and Fact Data" (p. 113).

Note: If you want to add a dimension delivery to a fact build, the fact build must have reference dimensions defined for it.

Add a Dimension Delivery

You can add dimension deliveries to a fact build manually or while creating a fact build using the Fact Build wizard.

For information about adding a dimension delivery using the Fact Build wizard, see "The Fact Build Wizard" (p. 133).

Steps

1. In the appropriate fact build, expand Delivery Modules and click Dimension Delivery.
2. Click the dimension for which you want to add a delivery.
3. From the Insert menu, click Build, and then click Dimension Delivery.
4. From the Dimension Delivery menu, click Relational Table.
   The Dimension Delivery Properties window for the selected dimension appears.
5. Click the General tab.

6. In the Name box, type a name for the delivery. If required, type a business name and description.

7. If you want to exclude the delivery when the build is executed, clear the Enabled check box.
   You can now temporarily disable the delivery without having to delete it and later recreate it.

8. Click the Output tab.

9. In the Deliver into database box, click the target database.

10. In the ID of top parent box, type the value of the parent attribute of the top-level members of the hierarchy.

11. If you want IBM® Cognos® Data Manager to truncate the table before delivery, select the Full refresh check box.

12. If the reference member has more than one parent and you want to deliver only the first parent, select the Ignore multiple parents check box.

13. Click OK.

   A dimension delivery is added to the selected dimension. You can now add tables to the dimension delivery.

---

**Add a Table to a Dimension Delivery**

Each dimension delivery delivers dimension data to one or more tables. There is no limit to the number of tables that you can add to a dimension delivery.

**Steps**

1. Click the dimension delivery to which you want to add a table.

2. From the Insert menu, click Build, and then click Table.

3. Click the Table tab.

4. Enter the name of the dimension table:
   - To deliver to a new table, in the Table name box, type a name for the table.
   - To deliver to an existing table, click the Browse for table button. The Select Table dialog box appears. Click the required table, and click OK.

5. In the Commit interval box, type (in terms of the number of rows updated/inserted) the interval between commits of data in the target table.

   By default, IBM® Cognos® Data Manager does not issue a COMMIT instruction until it has delivered all the data rows. Data Manager executes the entire delivery as a single transaction. You must decide whether this represents an unacceptably large or long transaction.

6. If you do not want rows for higher level members with no children to be delivered, select the Exclude Partially Populated Rows check box.
If this check box is clear, rows are delivered for higher level members with no children, and the lower level columns are set to null.

**Note:** You must clear this check box if you want to deliver an unbalanced hierarchy. For more information on unbalanced hierarchies, see "Hierarchy Types" (p. 121).

7. If you want to exclude the table when the build is executed, clear the **Enabled** check box.

8. If you want Data Manager to automatically add any missing columns to the target table, select the **Automatically add columns to table** check box.

   For more information, see "Add Missing Columns to the Target Table" (p. 215).

9. Click the **Columns** tab.

10. In the **Use template** box, click the template that defines the behavior of the dimension table.

    **Tip:** Alternatively, create a template by clicking **New**, or edit a template by clicking **Edit**. For more information about templates, see "Templates" (p. 77).

    The attributes of the template are shown.

    You can now specify the source of the dimension table columns from the column tokens and reference structure level attributes.

11. In the **Available Sources** pane, click the column token or level attribute and then in the left pane, click the column.

    For information about column tokens, see "Column Tokens" (p. 237).

12. Click **Map**.

    **Tip:** You can also drag the column token or level attribute to the relevant position in the **Sourced From** column.

13. To create an index for a column, select the relevant check box in the **Index** column. You can define the properties for the index by clicking the **Browse** button in the **Index Properties** column.

    For more information, see "Define the Index Properties" (p. 248).

14. Create any new columns that you require.

    New columns are added to the bottom of the list. You can reposition them as required.

15. Click **OK**.

---

**Disable and Enable a Dimension Delivery**

At times, you may want to temporarily disable a delivery. Although you can delete the delivery, and later recreate it, a better option is to disable the delivery. You can then enable it when it is required.
For more information, see "Delete a Dimension Delivery" (p. 230).

**Steps**

1. Click the required delivery.
2. From the Edit menu, click Properties.
3. Click the General tab.
4. Either clear or select the Enabled check box.
5. Click OK.

   **Tip:** You can also disable a delivery by clicking the delivery and then, from the Edit menu, clicking Enabled.

---

**Delete a Dimension Delivery**

You can delete a dimension delivery that is no longer required.

**Steps**

1. Click the required delivery.
2. From the Edit menu, click Delete.
3. In the message box, click Yes.

   **Tip:** You may want to temporarily disable a delivery rather than delete it, and then perhaps later have to create it. For more information, see "Disable and Enable a Dimension Delivery" (p. 229).
Chapter 18: Dimension Builds

A dimension build delivers data that describes a single business dimension, such as Product or Customer. Each dimension build

- acquires dimension data from the reference structure you specify
- delivers the dimension data to the target data mart, using a template to define the behavior of each column

You can choose to deliver the dimension data to a single table (star schema), to one table for each hierarchy level (snowflake schema), or to a table partitioned by custom criteria.

A dimension build can deliver dimension data for many fact builds. Fact builds can also be used to deliver dimension data, but this may mean several fact builds delivering the same data, resulting in a duplication of effort.

Use a dimension build when you

- have a number of target fact tables that share the same dimension tables
- want to deliver all the dimension data, including that for which there is no related fact data
- want to prepare the dimension tables prior to delivering the fact data (for example, where the fact data is temporarily unavailable)
- want to deliver dimension data with attribute changes tracked
- want to build a conformed data mart

For information about tracking attribute changes, see "Track Attribute Changes" (p. 238).

You can also use more advanced dimension build techniques to manage data containing changes to dimensional structures, and attributes containing historical changes. These advanced techniques could be used in the following situations:

- Your source dimension data contains specific date columns and you want to use these dates when setting the effective start and end dates for sets of dimension members that share a common business key.

  For more information on using specific date columns to set effective dates, see "Dimensional History" (p. 241).

  An example of this would be an Employee dimension that must use employment commencement dates from the source data to control dimensional processing.

- Your source dimension data contains more than one update for a dimension member. Each dimension update must be processed in timestamp order so that when the data is delivered to the dimension table, it accurately reflects all the changes that have occurred to a dimension member.

  For more information about processing dimension data in timestamp order, see "Dimensional History" (p. 241).
Your source fact data contains late arriving fact records and you want to ensure that each record can always be associated with the correct dimension record.
For more information, see "Process Late Arriving Facts" (p. 197).

View Dimension Builds

You can view dimension builds in the Tree pane or the Visualization pane.

View Dimension Builds in the Tree Pane

Each dimension build in the Builds and JobStreams folder is represented in the Tree pane as follows.

Tip: You can view just the dimension builds contained in a catalog by clicking the Dimension Builds tab at the bottom of the Tree pane.

View Dimension Builds in the Visualization Pane

When you click a dimension build in the Tree pane, a visual representation of the whole dimension build appears in the Visualization pane.

For a description of each of the symbols used in the Visualization pane, see "Buttons and Icons" (p. 549).

Use the Dimension Build Wizard

You can use the Dimension Build wizard to help you to create a dimension build that delivers the schema that you specify. For information, see "Schema Types" (p. 233).

Note: You can also create custom schema types by changing a dimension build to suit your requirements.

The Dimension Build wizard automatically creates the necessary templates that define the behavior of each column of the delivered tables.

When you use the Dimension Build wizard you define the

- schema details
- schema naming conventions
- features of the dimension build
• properties of the dimension build
• attributes for the dimension tables
• change tracking behavior

When you create a dimension build using the Dimension Build wizard, you can amend it, and add to it, as you require.

Steps
1. From the Tools menu, click Dimension Build Wizard.
2. Follow the instructions.

Schema Types

When you use the Dimension Build wizard, you must specify the schema to use to deliver dimension data. You can choose from these schema types:

• star
• optimal star
• snowflake
• optimal snowflake
• parent-child

Star Schema

A star schema represents the dimension in a single table. Each level of the delivered hierarchy is represented by one or more columns of this table. Therefore, this schema is a fully denormalized representation of the dimension.

If the table includes a surrogate key, this is assumed to be the primary key. Otherwise the ID of the lowest level is assumed to be the primary key.

If you want to use the Dimension Build wizard to create a dimension build that supports tracking of attribute changes, you must use this schema type.

Optimal Star Schema

An optimal star schema is a variant of the star schema. It has a main table that contains the keys of all the levels of the delivered hierarchy. However, it contains the description of only the lowest level. To save storage space, the descriptions of the higher levels reside in supplementary tables.
Snowflake Schema

A snowflake schema is more complex than a star schema. It represents the delivered hierarchy with one table for each hierarchy level. Each table has a foreign key to the level above. The primary key of each table is the member ID of the corresponding hierarchy level. This schema is a normalized representation of the dimension.

Optimal Snowflake Schema

An optimal snowflake schema is similar to a snowflake schema. However, the table that corresponds to the lowest level contains foreign keys to all of the levels above. This can improve performance because the analysis software does not need to follow the chain of foreign keys to identify a higher-level member.

Parent-Child Schema

A parent-child schema represents the delivered dimension in a single table. Each row of this table contains the ID of the member, together with the ID of its parent (where a parent exists). Although not required, a parent-child schema may identify the hierarchy level of each member. This schema is a recursive representation of the dimension.
A variant of this schema uses one table for all dimensions and has an additional column to identify the dimension to which each row relates.

We do not recommend the use of parent-child schemas with IBM® Cognos® software.

Create a Dimension Build Manually

To create a dimension build manually, you must

❑ add a dimension build
❑ add the required dimension tables

Add a Dimension Build

This section describes how to add a dimension build.

Steps

1. Click the Builds and JobStreams folder.
2. From the Insert menu, click Build, and then click Dimension Build.
3. Click the General tab.
4. In the Name box, type a unique name for the new dimension build. If you require, type a business name and description.
5. Click the Dimension tab.
6. In the Dimension to be delivered box, click the reference dimension that contains the reference structure to deliver.
7. In the Hierarchy/Lookup to be delivered box, click the reference structure that you want the dimension build to deliver.
8. In the Deliver into database box, click the target data mart for the dimension build.
   **Tip:** Click New to create a connection. For information, see Create a Database Connection.
9. If you want to deliver the reference structure as a parent-child schema, in the ID of top parent box, type the ID of the parent of the top member of the hierarchy.
   **Note:** This value is normally null because the top member has no parent.
10. To prevent delivery of unused, automatically generated, foster parents, select the Remove Unused Foster Parents check box.
    For information, see "Foster Parents" (p. 103).
11. If the dimension build should truncate the dimension tables each time it is executed, select the Full Refresh check box.
12. If the dimension build should only deliver the first-encountered parent of each member, select the **Ignore Multiple Parents** check box.

13. Click **OK**.

You have now entered the minimum information necessary to create a dimension build, and are ready to **add a table to the dimension build**.

## Add a Table to a Dimension Build

Each dimension build delivers dimension data to one or more tables. There is no limit to the number of dimension tables that a dimension build can deliver. However, they must all reside in the same database.

### Steps

1. Click the dimension build to which you want to add a table.

2. From the **Insert** menu, click **Build**, and then click **Table**.
   
   The **Dimension Table Properties** window appears.

3. Click the **Table** tab.

4. Enter the name of the dimension table:
   
   - To create a dimension table, in the **Table name** box, type a name for the table.
   
   - To deliver the dimension data to an existing table, click the **Browse for table button**.
     
     The **Select Table** dialog box appears. Click the required table, and click **OK**.

5. In the **Commit interval** box, type (in terms of the number of rows updated/inserted) the interval between commits of data in the target table.

   By default, a COMMIT instruction is not issued until all the data rows have been delivered. The entire delivery is executed as a single transaction. You must decide whether this represents an unacceptably large or long transaction.

6. If you do not want rows for higher level members with no children to be delivered, select the **Exclude Partially Populated Rows** check box.

   If this check box is cleared, then rows are delivered for higher level members with no children, and the lower level columns are set to null.

   **Note:** You must ensure this check box is cleared if you want to deliver an unbalanced hierarchy. For more information on unbalanced hierarchies, see "Hierarchy Types" (p. 121).

7. If you want to exclude the table when the build is executed, clear the **Enabled** check box.

8. If you want IBM® Cognos® Data Manager to automatically add any missing columns to the target table, select the **Automatically add columns to table** check box.

   For more information, see "Add Missing Columns to the Target Table" (p. 215).

9. Click the **Columns** tab.
10. In the **Use template** box, click the template that defines the behavior of the dimension table.

   You can create a template, or use any suitable existing template in the selected reference dimension.

   **Tip:** To create a template, click **New**, or to edit a template, click **Edit**.

   For more information, see "Create a Template" (p. 79).

   You are now ready to select the source of the dimension table columns from the column tokens and reference structure level attributes.

11. In the **Available Sources** pane, click the column token or level attribute and then, in the left pane, click the column.

   For more information, see "Column Tokens" (p. 237).

12. Click **Map**.

   **Tip:** You can also drag the level attribute or column token to the relevant position in the **Sourced From** column.

13. If you want to deliver data with the attribute changes tracked, select the **Track changes (Slowly Changing Dimension)** check box. In the **Track** column that appears, specify the columns to track and the tracking method to use, by clicking the appropriate tracking method (either 1, 2, or 0) from the relevant drop down list.

   **Note:** Only certain columns can be tracked. If a column cannot be tracked, the track value for the column is blank.

   For more information, see "Track Attribute Changes" (p. 238).

14. If you want to create an index for a column, select the relevant check box in the **Index** column.

   You can then define the properties for the index by clicking the **Browse** button in the **Index Properties** column.

   For more information, see "Define the Index Properties" (p. 248).

15. Click **OK**.

---

**Column Tokens**

When you add a table to a dimension build or a dimension delivery, you can select the source of the columns in the table from column tokens. Column tokens represent level attributes in a reference structure that have certain properties, irrespective of the level. For example, the level attribute that provides the ID for each row of data is represented by the $ID column token.

**Note:** You can specify a column token as the source of more than one dimension table column. This allows you to create as many custom dimension table deliveries as you require.

The following table describes each column token.

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ID</td>
<td>The ID of the member.</td>
</tr>
</tbody>
</table>
### Token Description

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CAPTION</td>
<td>The business name of the member.</td>
</tr>
<tr>
<td>$PARENT</td>
<td>The ID of the parent of the member.</td>
</tr>
<tr>
<td>&lt;level&gt;.$SURROGATE</td>
<td>The surrogate ID of the ancestor at the specified level. For more information, see &quot;Surrogate Keys&quot; (p. 83).</td>
</tr>
<tr>
<td>$SURROGATE</td>
<td>The surrogate ID of the member.</td>
</tr>
<tr>
<td>$PARENT_SURROGATE</td>
<td>The surrogate ID of the parent of the member.</td>
</tr>
<tr>
<td>$LEVEL_NAME</td>
<td>The name of the level in which the member resides.</td>
</tr>
<tr>
<td>$LEVEL.Caption</td>
<td>The business name of the level in which the member resides.</td>
</tr>
<tr>
<td>$LEVEL_NO</td>
<td>The ordinal number of the level in which the member resides. The topmost level has the ordinal number 1.</td>
</tr>
<tr>
<td>$SEQ_IN_LEVEL</td>
<td>The ordinal number of the member in a flat ordering sequence. The sequence numbers each level from left to right with the first number of each level continuing from the level above. For example</td>
</tr>
<tr>
<td>$SEQ_IN_ANCESTOR</td>
<td>The ordinal number of the member with respect to its ultimate ancestor. Each member of the top level has the number 1. Under each top-level member, the numbering restarts at 1. For example</td>
</tr>
</tbody>
</table>

### Track Attribute Changes

Tracking attribute changes is a technique for managing historical data. It allows you to maintain dimensions for which non-key attributes can change over time without corresponding changes in the business key. For example, employees may change their department without changing their employee number, or the specification for a product may change without changing the product code.

Each column that you include in a dimension table corresponds to a dimension attribute. These columns can either overwrite attribute changes or track attribute changes. When you add a column it is automatically set to overwrite attribute changes (type 1).

Tracking cannot be performed on the
• primary business key
• maintained surrogate key
• effective start or end dates
• create last update date
• current indicator

You set up attribute change tracking from the Columns tab in the Dimension Table Properties window.

Overwrite Attribute Changes (Type 1 Changes)

You overwrite attribute changes when historical values are not required. This is known as a type 1 change and is the default.

For example, a customer’s address may change but there is no business requirement to track previous addresses. All customer records containing the address are updated with the new address.

If IBM® Cognos® Data Manager encounters an attribute that requires a type 1 change, it updates the existing value with the new value, and sets the update date in all applicable records.

For information about how Data Manager sets the update date, see "Set the Date Source for Timestamping Data" (p. 137).

Track Attribute Changes (Type 2 Changes)

You track attribute changes when you want to keep a record of historical values. This is known as a type 2 change.

For example, when an employee is moved from branch to branch, all old transactions should remain in the old branch, and only new transactions should relate to the new branch.

If IBM® Cognos® Data Manager encounters an attribute that requires a type 2 change, it

• creates a new dimension data record with
  • the new attribute values
Chapter 18: Dimension Builds

- a new surrogate key
- the effective start date and current indicator
- current values of unchanged attributes
- updates the previous record by setting the effective end date and current indicator with the appropriate previous record behavior

For information about setting effective dates, see "Specify Effective Date Options" (p. 247).

**Note:** Where type 1 and type 2 changes have occurred for the same record, the type 1 updates that are not marked as type 2 are applied to all records for the business key at the level for which the type 1 attribute is declared, regardless of whether the record is current or not. For an example of this scenario, see "Example 2: Dimension Table With Attribute Change Tracking Behavior" (p. 251).

**Corrective Changes (Type 0 Changes)**

Type 0 changes can only occur where

- type 2 changes are set to occur
- the effective dates are timestamped according to the data

If there are multiple changes to an attribute on the same date, rather than maintaining a history of each change, IBM® Cognos® Data Manager overwrites the current record for each change.

For example, suppose a dimension table contains the following Product reference data, where the Prod_Desc attribute has been marked for a type 0 update, and the Prod_Type attribute marked as a type 2 update.

<table>
<thead>
<tr>
<th>Surr_key</th>
<th>Prod_Code</th>
<th>Prod_Type</th>
<th>Prod_Desc</th>
<th>Eff_Date</th>
<th>End_Date</th>
<th>Curr Ind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1</td>
<td>002</td>
<td>Blue pen</td>
<td>2005/02/10</td>
<td>2006/12/15</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>P2</td>
<td>001</td>
<td>Green pen</td>
<td>2005/03/15</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>P1</td>
<td>001</td>
<td>Blue pen</td>
<td>2006/12/15</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

A single dimension build is then used to load the following source data, to the dimension table shown above.

<table>
<thead>
<tr>
<th>Prod_Code</th>
<th>Prod_Type</th>
<th>Prod_Desc</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>001</td>
<td>Orange pen</td>
<td>2006/12/22</td>
</tr>
</tbody>
</table>

A type 0 change is applied to the Prod_Desc attribute for record 3, by overwriting the previous value with the value Orange pen.
Dimensional History

By default, when processing dimensional dates, IBM® Cognos® Data Manager uses the system date of the computer on which builds are executed to timestamp data. However, the source data may already contain dates that should be used instead of the system date. Additionally, the source data may contain more than one occurrence of a business key, and a date column that describes a series of dimensional updates for a common business key. These characteristics are common in employee dimensions or where more than one update occurs between updates being processed. This is known as dimensional history.

When delivering dimension data, Data Manager normally delivers only the most current record for a specific dimension member, ignoring any historical records. If your dimension table includes an attribute for which you are tracking changes, and there has been more than one change to a dimension member since you previously delivered dimension data, the result may be that there are gaps in the dimensional history.

For example, suppose the following Product reference data has already been delivered to a dimension table.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Surrogate Key</th>
<th>Specification</th>
<th>Effective Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>Blue pen</td>
<td>2000/11/01</td>
<td>2003/12/22</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>Black pen</td>
<td>2003/12/23</td>
<td>2004/02/09</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>Red pen</td>
<td>2004/02/10</td>
<td></td>
</tr>
</tbody>
</table>

You can see that changes to the Specification attribute are being tracked. The current dimension record for product P1 is record 4.

If you load the following dimension data, using a single dimension build, to the dimension table shown above, only the most current record (dated 2004/10/10) is delivered.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Specification</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Orange pen</td>
<td>2004/02/14</td>
</tr>
<tr>
<td>P1</td>
<td>Mauve pen</td>
<td>2004/04/27</td>
</tr>
</tbody>
</table>
The records dated 2004/02/14 and 2004/04/27 are ignored. Since the dimension data includes three changes to the specification of product P1, and only one is delivered, the dimension table now contains an incomplete history of changes to product P1.

By delivering the dimensional history, you can include a full and accurate history of changes to a dimension member since the previous delivery.

**Notes**

- You cannot deliver dimensional history that occurs prior to the effective date of the current dimension record in the dimension table. For more information, see "Reject Late Arriving Dimension Details" (p. 244).

- Delivering dimensional history is necessary if you want to process source data that contains late arriving facts. For more information, see "Process Late Arriving Facts" (p. 197).

Before you can deliver dimensional data, you must set up the dimensional history. For information, see "Set Up Dimensional History" (p. 242)

---

**Set Up Dimensional History**

To enable dimensional history in IBM® Cognos® Data Manager, and use a date column from the data source as the source for dimensional processing, you must ensure that

- the template that is used to define the attributes being read from the data source includes a column with effective start date behavior
  
  For more information, see "Steps to Set the Column Behavior in the Source Template" (p. 242).

- the template that is used to deliver the dimension build includes the same column, again with effective start date behavior
  
  For more information, see "Steps to Set the Column Behavior in the Delivery Template" (p. 243).

- the dimension build includes a dimension table column with effective start date behavior that is sourced from a reference structure attribute containing date values
  
  For more information, see "Add a Table to a Dimension Build" (p. 236).

- the dimension table must be set to track changes
  
  For more information, see "Track Attribute Changes" (p. 238).

---

**Steps to Set the Column Behavior in the Source Template**

1. In the Templates folder for the appropriate dimension, click the template that is used to define the attributes being read from the data source.

2. From the Edit menu, click Properties.

   The Template Properties window appears.
3. Click the Attributes tab.

4. In the Behavior column, select Effective Start Date for the column that is used as the date attribute to define dimensional history.

![Image of Attributes tab in template properties window]

5. Click OK.

**Steps to Set the Column Behavior in the Delivery Template**

1. Click the delivery template that is used to deliver the dimension table in the dimension build.

2. From the Edit menu, click Properties.

   The Template Properties window appears.

3. Click the Attributes tab.

4. In the Behavior column, select Effective Start Date for the column that was also marked as an effective start date in the source.

![Image of Attributes tab in template properties window]

When the attribute is defined as the effective start date, all dimension date processing for effective start and end dates uses these dates.

**Note:** It is important to ensure that only one attribute is marked as Effective Start Date.
5. Click OK.

**Reject Late Arriving Dimension Details**

If a dimension table includes an attribute for which you are tracking changes, when IBM® Cognos® Data Manager encounters a change in that attribute for a specific dimension member, it normally only checks the current dimension record for that member when updating the dimension table.

If the dimension data for a dimension member includes additional historical attribute changes which occurred prior to the effective start date for the current dimension record, they cannot be included in the dimension table and are rejected.

If the dimension data for a dimension member includes updated attributes and the effective start dates in the source data and current record are the same, the attributes are overwritten. This happens even if history is being tracked for one or more attributes. This occurs because the effective start dates are the same, and the record is considered to be a correction.

For example, suppose the following Product reference data has already been delivered to a dimension table.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Surrogate Key</th>
<th>Specification</th>
<th>Effective Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>Blue pen</td>
<td>2000/11/01</td>
<td>2003/12/22</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>Black pen</td>
<td>2003/12/23</td>
<td>2004/02/09</td>
</tr>
<tr>
<td>P1</td>
<td>4</td>
<td>Red pen</td>
<td>2004/02/10</td>
<td></td>
</tr>
</tbody>
</table>

You can see that changes to the Specification attribute are being tracked. The current dimension record for product P1 is record 4.

If the specification of product P1 changes on 2004/06/07, record 4 is checked, and the dimension table is updated as required.

Suppose the following dimension data is delivered, using a single dimension build, to the dimension table shown above.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Specification</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Orange pen</td>
<td>2001/05/14</td>
</tr>
<tr>
<td>P1</td>
<td>Mauve pen</td>
<td>2003/04/27</td>
</tr>
<tr>
<td>P1</td>
<td>Green pen</td>
<td>2004/10/10</td>
</tr>
</tbody>
</table>

The dimension data includes three changes to the specification of product P1, two of those changes occurring prior to the effective date of the current dimension record in the dimension table (record 4) for product P1. In this case, the dimension changes dated 2001/05/14 and 2003/04/27 are rejected, and only the dimension table is updated by using changes for 2004/10/10.
Save Rejected Late Arriving Dimension Details

When IBM® Cognos® Data Manager rejects records during build execution, the rejected data can be saved to

- file
  Data Manager creates a simple tab delimited text file. The first time a build that creates reject records is executed, Data Manager creates a text file. If the build has been executed previously, the existing data in the text file is updated.
  By default, the text file is named <build_name>.rej and is stored in the Data Manager Data directory.
  Saving reject records to file is the default.

- table
  This table can reside in any writeable Data Manager connection. The default is the same DBMS as the delivered fact build.

If you do not want to save rejected records, click Don’t write reject records.

Steps to Write Details to a File

1. Click the dimension table for which you want to specify dimension history options.

2. From the Edit menu, click Properties.

3. Click the Dimension History Options tab.
   Note: This tab is only available if the dimensional history has been set up. For information, see "Set Up Dimensional History" (p. 242).

4. In the Late Arriving Dimension Details box, click Write details to file.

5. If you want to change the default name and location of the reject file, in the File name box, enter the full directory path and file name.

6. Click OK.

Steps to Save Details to a Relational Table

1. Click the dimension table for which you want to specify dimension history options.

2. From the Edit menu, click Properties.

3. Click the Dimension History Options tab.
   Note: This tab is only available if the dimensional history has been set up. For information, see "Set Up Dimensional History" (p. 242).

4. In the Late Arriving Dimension Details box, click Write details to table.

5. In the Connection box, click the connection that contains the table where the details are to be saved.
Tip: To create a connection, click New. For information on creating a connection, see "Create a Database Connection" (p. 63).

6. In the **Table name** box, enter the name of the table to which you want to save the details.
   
   **Tip:** You can also click the **Browse for table** button and then click the required table from the list in the **Select Table** dialog box.

7. Click **OK**.

**Steps to Save Details Ready for Use in IBM® Cognos® Data Manager SQLTXT Designer**

1. Click the dimension table for which you want to specify dimension history options.

2. From the **Edit** menu, click **Properties**.

**Specify the Source of the Effective Start Date in Initial Records**

By default, when you deliver dimension data for a dimension member, the effective start date in the initial record is set according to the template. IBM® Cognos® Data Manager uses source dates for the effective start and end dates when tracking dimensional history updates.

However, it is possible that the effective start date in the initial record for each dimension member may differ, as illustrated below.

<table>
<thead>
<tr>
<th>Product Code</th>
<th>Surrogate Key</th>
<th>Specification</th>
<th>Effective Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>Blue pen</td>
<td>2000/11/01</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>White paper</td>
<td>2003/11/01</td>
<td></td>
</tr>
</tbody>
</table>

It may be important to override the initial start date in the template settings for the first record in a dimensional history load. This could occur, for example, if you want the start date for the initial record to be set to a date that defines the beginning of all records. The initial start date could also be set to a null date.

You can define the source for the first effective start date in a sequence using the **Effective Start Date in initial records options** on the Dimension History Options tab. The source can be

- a source attribute
  
  Data Manager sets the initial start date according to the date stored in the source column.

- a reference template
  
  This allows you to specify how you want the start date to be set.

For more information on setting the effective start date in a template, see "Specify Effective Date Options" (p. 247).

**Steps**

1. Click the dimension table for which you want to specify dimension history options.
2. From the **Edit** menu, click **Properties**.

3. Click the **Dimension History Options** tab.

   **Note:** This tab is only available if the dimensional history has been set up. For information, see "Set Up Dimensional History" (p. 242).

4. In the **Effective Start Date in initial records** box, you specify the source to use for the effective start date in the initial dimension record. Click
   - **From source attribute** to set the effective start date to the date specified for each dimension member in the dimension data. (default)
   - **According to reference template** to use a template to set the effective date for all dimension members.

5. Click **OK**.

**Specify Effective Date Options**

You can specify the way in which the effective start date and effective end date are set in the template referenced by a dimension table. This controls the way in which the date sequence of dimensional changes occurs.

For example, sometimes the grain of dimensional updates requires that changes occur throughout a day, and sometimes only whole days are required.

**Note:** Changing the template affects effective date options for all dimension tables referencing that template. For more information, see "Templates" (p. 77).

**Steps**

1. Click the dimension table for which you want to specify effective date options.

2. From the **Edit** menu, click **Properties**.

3. Click the **Columns** tab, and then click **Edit**.

   The **Template Properties** window appears.

4. Click the **Effective Date Options** tab.

5. To specify the format to use for effective dates, in the **Effective Date granularity** box click
   - **Timestamp (date and time)** if dates should include the date and time. (default)
   - **Date only** if dates should not include the time.

6. To specify how the effective start date in the initial record for a specific dimension member is set, in the **Effective Start Date in initial records** box, click
   - **Use data timestamp value** to specify that the date should correspond to the timestamp set by the dimension build.
   - **Variable** to use a variable to set the date, and then in the adjacent box, type the variable name.
Note: The variable must be defined on the Variables tab. For information, see "Variables" (p. 319).

- **Date** (yyyy-mm-dd hh:mm:ss) to set an explicit date and time, and then in the adjacent box, type the date and time.

  Note: When **Date only** is selected in the Effective Date granularity box, you cannot include the time.

- **Null** to specify that no date should be set.

7. You specify how the effective end date should be set when a change in the dimension record is detected in the Effective End Date in current records box. For information, see step 6.

8. To specify how the effective end date for the previous row should be set when a new dimension data row is created, in the Set previous record Effective End Date to box, click

   - **New Effective Start Date** to set the date to be the same as the effective start date of the new dimension data row.

   - **New Effective Start Date - 1 day or second** to set the date to the effective start date of the new dimension data row minus one day or one second. (default)

     Note: When **Date only** is selected in the Effective Date granularity box, the New Effective Start Date option is set minus 1 day. If **Timestamp (date and time)** is selected, the option is set to minus 1 second.

9. Click **OK**.

### Define the Index Properties

When you create an index for a dimension build, you define its properties in the Index Properties dialog box.

**Steps**

1. Click the table for which you want to define the index.

2. From the Edit menu, click **Properties**.

3. Click the Columns tab.

4. In the Index column, select the check box for the column to index, and then click the Browse button that appears.

5. In the Name box, type a name for the index.

6. In the Type box, click the index type to create:

   - **Unique** for an index that constrains the dimension data so that each value in the indexed column is unique.

   - **Bitmap** for an index that is of a compressed format which some database management systems implement. Using an index of this type can enhance search performance.
Note: If the Type box is empty, the default index type for the target DBMS is created. For more information, see the manufacturer’s documentation for the type of database to which you are delivering.

7. In the Options box, type any additional options for the index. The contents of this box are appended to the CREATE INDEX statement. For information about valid options, see the documentation for the target DBMS.

8. If you want the index to continue generating when errors are encountered, select the Suppress errors check box.

9. If you want the index deleted and then recreated, select the Recreate index check box. This method is quicker than updating the existing index.

10. Click OK.

Recommended Indexes for Dimension Tables

IBM® Cognos® Data Manager manages dimension tables by issuing SQL requests to insert and update dimension table records. The speed of processing dimension table updates can be affected by the indexes that are applied to the table in the database. The surrogate key of the lowest level of any hierarchy defined in the dimension is typically the primary key of the dimension table. This column should be set as the primary key for the table in the DBMS. Updates to the dimension table are typically constrained with an SQL WHERE clause, using the business keys of the levels on any hierarchy defined in the dimension. It is therefore recommended that non-unique indexes are implemented on columns that are marked as business keys in the delivery template.

Set the Date Source for Timestamping Data

When you execute a dimension build, records that contain attributes with create date or update date behavior are timestamped. For more information about these attributes, see "Behavior Types in Templates" (p. 80).

By default, the data is timestamped using the current system date and time of the computer on which the build is executed. If you require, you can timestamp the data using a different data source. For example, if your target database server is situated on a different computer to the one on which IBM® Cognos® Data Manager is running, it may be important that you timestamp data from the database server.

Note: You can use this setting to control the date source for all date attributes, unless the effective start date attribute is sourced from a column in the source template. Where the effective start date attribute is mapped from the source, this setting controls only the create date and update date attributes. For more information, see "Specify Effective Date Options" (p. 247).

Steps

1. Click the required dimension build.

2. From the Edit menu, click Properties.
3. Click the General tab.

4. In the Data timestamp box, choose the date source to be used for timestamping data:
   - Click Client system date/time to set the date to the current system date and time on the computer on which the build is executed. (default)
   - Click Variable to use a variable to set the date and, in the adjacent box, type the variable name.
     Note: The variable must be defined on the Variables tab. For information, see "Variables" (p. 319).
   - Click Date (yyyy-mm-dd hh:mm:ss) to set an explicit date and time and, in the adjacent box, type the date and time.

5. Click OK.

Delete a Dimension Build

You can delete a dimension build that is no longer required.

Steps
1. Click the required dimension build.
2. From the Edit menu, click Delete.
3. In the message box, click Yes.
   If deleting the dimension build results in a template being unused, the Delete Templates dialog box appears.
4. Click Yes to delete the templates listed, or No to delete the dimension build, but not the template.

Examples of how IBM Cognos Data Manager Maintains Dimension Tables

The examples that follow are based on a simple star schema, and describe how dimension tables are maintained in IBM® Cognos® Data Manager.

The basic requirement of a star table is that there
- is a column for a business key
- may be columns that are used to maintain the table
- may be further columns representing various attributes of the key. These attributes may or may not represent possible aggregations of the key.

Example 1: Simple Dimension Table With No Tracking Behavior

In a simple dimension table there
- must be a business key
- may be a surrogate key (although recommended, it is not mandatory)
- may be further attributes of the business key

The Product dimension build in the DS_Tutorial catalog is an example of a simple dimension table. The following are possible scenarios when adding records to a simple dimension table.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>The key does not exist.</td>
<td>A record is created in the dimension table, and if required, a new surrogate is generated.</td>
</tr>
<tr>
<td>The key does exist and one or more attributes have changed.</td>
<td>The attributes are updated.</td>
</tr>
<tr>
<td>The key does exist and no attributes have changed.</td>
<td>The record is ignored.</td>
</tr>
</tbody>
</table>

When a simple dimension table is described using a template, this information is recorded.

<table>
<thead>
<tr>
<th>Column</th>
<th>Behavior</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Business key</td>
<td>Identifies the dimension record, but it is not necessarily unique within the dimension table.</td>
</tr>
<tr>
<td>Surrogate key</td>
<td>Surrogate key</td>
<td>A surrogate value generated in response to a new dimension record. It is linked to a business key.</td>
</tr>
<tr>
<td>Attribute 1, 2, 3, and so on</td>
<td>Normal No tracking behavior defined in dimension table</td>
<td>Attributes that will be overwritten when they change. In other words, type 1 changes are performed on these attributes.</td>
</tr>
</tbody>
</table>

**Example 2: Dimension Table With Attribute Change Tracking Behavior**

A dimension table that tracks attribute changes builds on the action of the simple dimension table. The StaffSCD dimension build in the DS_Advanced catalog is an example of a dimension table that tracks attribute changes.

The following are possible scenarios when adding records to a dimension table that tracks attributes changes.
### Scenario | Response
--- | ---
The key does not exist. | A record is created in the dimension table, and a new surrogate is generated.
The key does exist and one or more type 1 attributes have changed. | The attributes are updated for any record in the dimension table that has the same business key as the level for which the attribute has changed. The update date column is updated for all affected records.
The key does exist and one or more type 2 attributes have changed. | A record is created in the dimension table, and a new surrogate is generated.
The end date and current indicator are updated for the previous record, according to the options defined in the delivery template.
The effective date and current indicator are set for the new record, according to the rules defined in the delivery template.
The key does exist and type 1 and type 2 attributes have changed. | A record is created in the dimension table, and a new surrogate is generated.
The new record contains the current view of the dimension with all current attributes.
All type 1 attributes in non-current records are updated to the current type 1 value, if the records have the same business key that is associated with the updated type 1 attribute.
All previous records retain the past values for attributes marked as type 2. Date attributes and current indicator flags are updated according to the rules defined in the delivery template.
The key does exist and no attributes have changed. | The record is ignored.

The objective of such a dimension is to track the changing values of attributes over time. In other words, rather than just overwriting changed attributes, the history within a single table is tracked. There are some additional table columns that make the process of tracking the changing values simpler. These table columns are all concerned with tracking the behavior of the table rows.
### Table: Dimension Column Behaviors

<table>
<thead>
<tr>
<th>Column</th>
<th>Behavior</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Business key</td>
<td>Identifies the dimension record, but it is not necessarily unique within the dimension table.</td>
</tr>
<tr>
<td>Surrogate key</td>
<td>Surrogate key</td>
<td>A surrogate value generated in response to a new dimension record. It is linked to a business key.</td>
</tr>
<tr>
<td>Attribute 1, 2, 3, and so on</td>
<td>Normal</td>
<td>Attributes that will be overwritten when they change. In other words, type 1 changes are performed on these attributes.</td>
</tr>
<tr>
<td>Attribute 4, 5, 6, and so on</td>
<td>Normal</td>
<td>Attributes for which a dimension record will be created when they change. In other words, type 2 changes are performed on these attributes.</td>
</tr>
<tr>
<td>eff_date</td>
<td>Effective start date</td>
<td>The date the record became effective. If the same business key was already in the table, all previous records become ineffective (see end_date and curr_ind).</td>
</tr>
<tr>
<td>end_date</td>
<td>Effective end date</td>
<td>The date the record became ineffective due to the generation of a new dimension record for the same business key. For a current record, this value is set according to the template properties.</td>
</tr>
<tr>
<td>curr_ind</td>
<td>Current indicator</td>
<td>A pair of values used to indicate current and past records.</td>
</tr>
<tr>
<td>cre_date</td>
<td>Create date</td>
<td>The date the record was created. This date is always the same as udt_date.</td>
</tr>
<tr>
<td>udt_date</td>
<td>Last update date</td>
<td>The last date when the record changed. This could be because an attribute was updated with a type 1 change.</td>
</tr>
</tbody>
</table>

In this case the template has considerable knowledge of the meaning of columns in the dimension table. These column behaviors are used both to maintain the dimension table, and to access the table data. In maintaining a dimension table it is the current dimension values that must be compared against new records and potentially be updated. This means that fast access to current values is important.
These dimension values are checked in the following order when trying to locate a current dimension record.

1. The current indicator.
   
   This is the simplest and fastest way to locate current dimension records.

2. The effective start date.

   If the current indicator is not stored in dimension records, IBM® Cognos® Data Manager searches for the record with the latest effective start date. Access to current values is slower because a sort must be performed to find the latest record.

In summary, when delivering dimension data, Data Manager searches the dimension table for the business key value of the row to be delivered. If this search shows that the business key value exists in the dimension table, Data Manager determines whether any attributes have changed value, and updates the existing record, or creates a new record accordingly.
You can automate tasks related to managing the build process by creating a JobStream to contain a series of steps. For example, a JobStream can

- coordinate fact and shared dimension builds
- prefix dimension builds with staging builds
- perform pre and post processing SQL
- set different arrival rates of source data
- distribute tasks over multiple processors
- customize application logging
- provide build status

A JobStream contains a node for each step in a process, which can be a fact build, dimension build, SQL statement, condition, procedure, alert, email, or another JobStream. Each JobStream can use any number of these nodes.

You link the nodes together to create a flow to instruct IBM® Cognos® Data Manager of the order in which each node is to be processed. Each JobStream can have a single flow or multiple flows. Data Manager can execute a JobStream in sequence or in parallel.

You can execute any JobStream from the command line, or integrate it into an application or scheduling architecture. For more information, see "Execute a JobStream on your Local Computer" (p. 276).
Fact Build Node
You can include any fact builds from the current catalog in a JobStream.
For more information, see "Fact Builds" (p. 131).

Dimension Build Node
You can include any dimension builds from the current catalog in a Jobstream.
For more information, see "Dimension Builds" (p. 231).

SQL Node
An SQL node can contain a series of SQL statements that are processed when the SQL node is reached in the JobStream flow. An SQL node is useful for pre and post table processing. Data Manager automatically commits successful SQL statements, but rolls back if the statement fails.

Notes
- Each SQL node is executed in a database transaction. Some databases have restrictions on the type of statements that can be executed in a database transaction. For specific examples, see the vendor's documentation for your database.
- SELECT statements cannot be used in SQL nodes.

Procedure Node
A procedure node can contain one or more Data Manager functions or statements. A procedure node is useful for checking for input files and generating custom logging and auditing messages.

Condition Node
A condition node provides branching between nodes for conditional execution. Each condition node can have many nodes linking to it, but only two output links, True and False.

JobStream Node
A JobStream node is a JobStream within a JobStream. This allows larger jobs to be broken down into a series of smaller jobs. When a JobStream node is encountered, all the steps within the JobStream node are processed before processing of the next node in the sequence starts.

Alert Node
An alert node writes a user-defined audit record of the type ALERT, into the Data Manager audit tables. You can use these records to record specific events that occur during JobStream and build execution.
You can also use other tools to access the audit tables, for example, custom audit reports using IBM® Cognos® Business Intelligence.

Email Node
An email node sends event notifications to mail systems. For example, you can set up emails to provide notifications when a JobStream has completed or failed. You can also include attachments with emails.
The method used to send an email varies according to whether or not you are using IBM® Cognos® BI.

If you are not using IBM® Cognos® BI, emails are sent using MAPI. However, if IBM® Cognos® BI is being used, you can specify that emails are sent using the delivery service. You do this by declaring the DM_EMAIL_VIA_SERVICE environment variable.

For more information, see "DM_EMAIL_VIA_SERVICE" (p. 326).

Note: Email attachments are not supported on the UNIX® operating system.

**DataStage Node**

A DataStage node executes an IBM® DataStage® job using the Data Manager engine. Using a DataStage node in a Data Manager JobStream allows you to separate the processing of data as follows:

- extract and transform source data using DataStage®
- deliver transformed data into data marts using Data Manager

Before you can use DataStage nodes in a JobStream, you must install the Information Server Client.

**View JobStreams**

You can view JobStreams in the **Tree pane** or the **Visualization pane**.

**View JobStreams in the Tree Pane**

Each JobStream in the Builds and JobStreams folder is represented in the Tree pane as follows.

```
| Start | Report Loop | Extract Data | Increment Counter | Test Counter | Report End |
```

Note: If the associated build or JobStream name differs from the node name, it is shown in brackets.

Tip: You can view just the JobStreams contained in a catalog by clicking the JobStreams tab at the bottom of the Tree pane.

**View JobStreams in the Visualization Pane**

When you click a JobStream in the Tree pane, a visual representation of the whole JobStream is shown in the Visualization pane. Viewing a JobStream in this way makes it easier to understand the JobStream flow.
For information about each of the symbols used in the Visualization pane, see "Buttons and Icons" (p. 549).

**Note**: If the associated build or JobStream name differs from the node name, it is shown in brackets.

You can add, link, and reposition nodes, show the properties for a node, and convert a node to a JobStream from the Visualization pane.

By default, the nodes remain in exactly the position where you placed them. However, you can specify that IBM® Cognos® Data Manager arranges the nodes in what it considers to be the most logical way. You can also choose to align nodes to an invisible grid. When you save a catalog, the nodes remain in position, whether they were placed manually by you or automatically by Data Manager.

**Notes**

- If you include a JobStream in a catalog backup or component package, the current layout is stored as an attribute of the JobStream.

- If you are using source code control, you must first check out the JobStream to change the layout.

**Step to Use Automatic Layout to Position the Nodes**

- From the View menu, click Layout the diagram automatically.

**Steps to Use the Grid to Position the Nodes**

1. From the View menu, click Snap to Grid.
   
   Until you turn off this feature, all the nodes that you add are automatically aligned to the grid.

2. To align existing nodes to the grid, select the node or nodes, and from the View menu, click Align selected objects to the Grid.

   **Tip**: To select more than one node, hold down the Ctrl key and click each node. Alternatively, click within the Visualization pane close to the first node that you want to select, keep the mouse button depressed, and drag the pointer so that all the required nodes are selected, and then release the mouse button.

**Create a JobStream**

This section describes how to create a JobStream.

**Steps**

1. Click the Builds and JobStreams folder.

2. From the Insert menu, click JobStream, and then click JobStream.

3. Click the General tab.
4. In the Name box, type a name for the new JobStream and, if you require, a business name and description.

5. Click OK.

The JobStream appears in the JobStream tree and in the Visualization pane. Note that in the Visualization pane, at this stage the JobStream is represented as a start node.

**Define Logging and Audit Information for a JobStream**

When you execute a JobStream, progress information is returned to the log file and audit tables. You can specify the type of information that is recorded. For more information, see "Execution Log Files" (p. 282) and "The Audit Trail" (p. 287).

**Define JobStream Variables**

If any nodes in a JobStream use variables, you must define those variables.

**Steps**

1. Click the JobStream.

2. From the Edit menu, click Properties.

3. Click the Variables tab.

4. Define the variables as required.

   For information about defining variables, see "Variables" (p. 319).

   **Tip:** After creating a JobStream, if you want to change the name of a variable, you must update all references to that variable.

5. Click OK.

**Add a Fact Build, Dimension Build, or JobStream Node to a JobStream**

When you add a fact build, dimension build, or JobStream node, you associate the relevant build or JobStream with the node. It is possible for the business name that you assign to the node to differ from the actual name of the associated build or JobStream. In this instance, the node is shown in the Tree pane with the associated build or JobStream name in brackets.

**Steps**

1. Click the JobStream to which you want to add a fact build, dimension build, or JobStream node.

2. From the Insert menu, click JobStream, and then click either
You can also add a fact build, dimension build, or JobStream by clicking on the arrow to the right of the relevant toolbar button. The Item List appears listing all the available components for the selected node type. Double-click the required component.

If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be added, and then the Properties window appears.

3. Click the General tab.

4. If you require, type a business name and a description.

   Note: You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. In the Associated build or Associated JobStream box, click the browse button and click the build or JobStream to use.

   You can choose from all the fact builds, dimension builds, or JobStreams in the current catalog.

6. Select or clear the Exclude this node from processing check box.

   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

7. Click the Details tab.

8. In the Action on failure box, click the action to take should a problem be encountered when processing the node.

   For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

9. If required, in the Weight box, type a suitable value to indicate the total load required to process the node.

   For information, see "Specify the Load Required to Process a Fact Build or Dimension Build Node" (p. 269).

10. The Result variable box shows the name of the variable that is to receive the result of processing the node. By default, this is RESULT however, you can change this if you require. If you specify another variable, you must add the variable to the Variables tab of the JobStream Properties window.

   For information, see "Define JobStream Variables" (p. 259).

11. Click OK.
Add a Procedure or Condition Node to a JobStream

You can add procedure and condition nodes to a JobStream.

Steps

1. Click the JobStream to which you want to add a procedure or condition node.

2. From the Insert menu, click JobStream and then click either
   - Procedure Node
   - Condition Node

   Note: If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be inserted, and then the Properties window appears.

3. Click the General tab.

4. If you require, in the Business name box, type a name for the node and, in the Description box, type a description.

   Note: You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. Select or clear the Exclude this node from processing check box.

   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

6. If you are creating a procedure node and you want the node to run as a separate process rather than the default or running inline, select the Run as a separate process check box.

   For information, see "Execute Procedure and SQL Nodes as Separate Processes" (p. 278).

7. Click the Action tab.

8. In the right pane of the Action box, enter an expression to specify the action that Data Manager should take.

   You can type the expression, and you can use the tree structure in the left pane to select the items from the following folders:
   - Operators to select operators
   - Functions to select pre-defined and user-defined functions
   - Control to select controls
   - Variables to select variables and substitution variables

   For information about operators, controls, and pre-defined functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305). For information about variables and substitution variables, see "Variables" (p. 319).

   You can select an item from a folder by dragging it to the right pane or by double-clicking it.
9. If required, test the expression. For information, see "Testing Expressions and Scripts" (p. 341).

10. Click the Details tab.

11. In the Action on failure box, click the action to take should a problem be encountered when executing the node.

   For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

12. The Result variable box shows the name of the variable that is to receive the result of executing the node. By default, this is RESULT however, you can change this if you require. If you do specify another variable, you must add the variable to the Variables tab of the JobStream Properties window.

   For information, see "Define JobStream Variables" (p. 259).

13. Click OK.

Add an SQL Node to a JobStream

You can add SQL nodes to a JobStream.

**Steps**

1. Click the JobStream to which you want to add an SQL node.

2. From the Insert menu, click JobStream and then click SQL Node.

   **Note:** If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be inserted, and then the Properties window appears.

3. Click the General tab.

4. If you require, in the Business name box, type a name for the node and, in the Description box, type a description.

   **Note:** You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. Select or clear the Exclude this node from processing check box.

   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

6. If you want the node to run as a separate process rather than the default or running inline, select the Run as a separate process check box.

   For information, see "Execute Procedure and SQL Nodes as Separate Processes" (p. 278).

7. Click the SQL tab.

8. In the Database box, click the target database to which the SQL statements apply.

   **Tip:** Click New to create a connection. For information, see Create a Database Connection.

9. In the SQL box, enter the required SQL statements.
You can click SQL Helper to help you construct statements. For information about using SQL Helper, see "SQLTerm and SQL Helper" (p. 55).

10. Select the Cognos SQL check box to specify that you want to use Cognos® SQL when you construct the SQL statement. If you clear this check box, you must use native SQL for database you are accessing.

   Note: The default for the Cognos SQL check box is determined by whether or not you selected the Cognos SQL check box in the Connection Properties dialog box.

   For more information, see "Cognos SQL" (p. 40).

11. Click the Details tab.

12. In the Action on failure box, click the action to take should a problem be encountered when processing the node.

   For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

13. The Result variable box, shows the name of the variable that is to receive the result of processing the node. By default, this is RESULT however, you can change this if you require. If you do specify another variable, you must add the variable to the Variables tab of the JobStream Properties window.

   For information, see "Define JobStream Variables" (p. 259).

14. Click OK.

### Add an Alert Node to a JobStream

You can add alert nodes to a JobStream.

**Steps**

1. Click the JobStream to which you want to add an alert node.

2. From the Insert menu, click JobStream and then click Alert Node ⬤.

   Note: If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be inserted, and then the Properties window appears.

3. Click the General tab.

4. If you require, in the Business name box, type a name for the node and, in the Description box, type a description.

   Note: You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. Select or clear the Exclude this node from processing check box.

   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

6. Click the Action tab.
7. In the Item box, type the topic of the message.

8. In the Message box, type the text of the message.

9. Click the Details tab.

10. In the Action on failure box, click the action to take should a problem be encountered when processing the node.

   For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

11. The Result variable box shows the name of the variable that is to receive the result of processing the node. By default, this is RESULT however, you can change this if you require. If you do specify another variable, you must add the variable to the Variables tab of the JobStream Properties window.

   For information, see "Define JobStream Variables" (p. 259).

12. Click OK.

13. Specify that alert messages for the JobStream are to be recorded in the audit trail. For information, see "The Audit Trail" (p. 287).

**Add an Email Node to a JobStream**

You can add email nodes to a JobStream.

**Steps**

1. Click the JobStream to which you want to add an email node.

2. From the Insert menu, click JobStream and then click Email Node.

   **Note:** If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be inserted, and then the Properties window appears.

3. Click the General tab.

4. If you require, in the Business name box, type a name for the node and, in the Description box, type a description.

   **Note:** You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. Select or clear the Exclude this node from processing check box.

   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

6. Click the Action tab.

7. In the Profile box, type the email profile for the computer that you are using.

   For information, see the documentation for the operating system that you are using.

8. In the Password box, type the password that you use to access email.
Chapter 19: Managing Builds Using JobStreams

9. In the **To** box, type the email address of the person to whom you want to send the email. If there is more than one recipient, separate each name using a semicolon.

10. In the **Cc** box, type the email address of any recipients to whom you want to send a copy of the email. If there is more than one recipient, separate each address using a semicolon.

11. In the **Bcc** box, type the email address of any recipients to whom you want to send a copy of the email, without the recipient’s name being visible to other recipients of the email. If there is more than one recipient, separate each address using a semicolon.

12. In the **Subject** box, type the topic of the message.

13. In the **Attachment** box, type the full file path and name of the file that you want to send with the email message. You can send a maximum of 10 attachments, with each file separated by a comma.

   **Note:** Email attachments are not supported on the UNIX® operating system.

14. In the **Message** box, type the text of the message.

15. Click the **Details** tab.

16. In the **Action on failure** box, click the action to take should a problem be encountered when processing the node.

   For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

17. The **Result variable** box, shows the name of the variable that is to receive the result of processing the node. By default, this is RESULT however, you can change this if you require. If you do specify another variable, you must add the variable to the **Variables** tab of the **JobStream Properties** window.

   For information, see "Define JobStream Variables" (p. 259).

18. Click **OK**.

---

**Add a DataStage Node to a JobStream**

You can add DataStage® nodes to a JobStream.

**Steps**

1. Click the JobStream to which you want to add a DataStage® node.

2. From the **Insert** menu, click **JobStream** and then click **DataStage Node**.

   **Note:** If you use the toolbar button or right-click menu to add a node, the pointer changes to a cross-hair allowing you to choose the position where the new node is to be inserted, and then the **Properties** window appears.

3. Click the **General** tab.

4. If you require, in the **Business name** box, type a name for the node and, in the **Description** box, type a description.
Chapter 19: Managing Builds Using JobStreams

Note: You do not have to type a name (or ID) for the node, as IBM® Cognos® Data Manager allocates this automatically, and shows it at the bottom right of the window.

5. Select or clear the Exclude this node from processing check box.
   For information, see "Exclude Nodes from Processing in a JobStream" (p. 278).

6. Click the Action tab.

7. Enter the required connection details for IBM® DataStage® in the Domain, Username, and Password boxes.

8. Enter the details of the DataStage job to execute in the Server, Project, and Job boxes.

9. Select the Reset if required box to automatically reset a DataStage job that has stopped or aborted.

10. If the DataStage® job contains parameters, you must specify the parameter values to use when the DataStage® node is executed.
    Click Parameters.

11. Click Add, and enter the parameter name (as defined in the DataStage® job) and the required value.

12. Repeat step 11 add any further parameters as required.
    
    ● To edit the details of a parameter, select it and click Edit.
    
    ● To delete a parameter, select it and click Delete.

13. Click the Details tab.

14. In the Action on failure box, click the action to take should a problem be encountered when processing the node.
    For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

15. The Result variable box, shows the name of the variable that is to receive the result of processing the node. By default, this is RESULT however, you can change this if you require. If you do specify another variable, you must add the variable to the Variables tab of the JobStream Properties window.
    For information, see "Define JobStream Variables" (p. 259).

16. Click OK.

Delete a Node from a JobStream

You can delete a node that is no longer required from a JobStream.

Steps
1. Click the node that you want to delete.
2. From the Edit menu, click Delete.
3. In the message box, click **Yes**.

**Link Nodes in a JobStream**

Each node can have one or more nodes linked to it and must, in turn, link to at least one other node. The exception is a condition node, which must link to no more than two nodes; one link must be True, the other False.

Processing begins at the Start node and then progresses through the JobStream following the links that you add. When a JobStream node is encountered, the whole JobStream is processed before progressing to the next node.

You can link nodes using the Visualization pane or the Predecessors and Successors tabs in the node Properties windows.

**Steps to Link Nodes in the Visualization Pane**

1. In the Tree pane, click the JobStream for which you want to add links.

2. Click **Insert Link**.

   An arrow appears above the pointer to indicate the direction of the link.

   **Note:** Link mode is automatically cleared, when you next right-click the Visualization pane.

3. Click the node that is to be the starting point for the link, and drag the pointer to the node to which it is to link.

   If the link is from a condition node, a status of True is allocated to the first link that you create and False to the other. However, you can reverse the logic if you want. For information, see "Change the True and False Status for a Condition Node" (p. 268).

4. Repeat steps 2 and 3 to create additional links.

   - To delete a link, click the link in the Visualization pane, and from the Edit menu, click **Delete**, or press the Delete key.

   - To return to select mode, click **Select mode**.

**Steps to Link Nodes in the Node Properties Window**

1. Click the required node.

2. From the Edit menu, click **Properties**.

3. Click the Predecessors tab.

4. Select the check boxes adjacent to the nodes from which you want to link; that is, the nodes immediately before the selected node.

5. Click the Successors tab.

   **Note:** If you are linking from a condition node, the Successors tab is replaced with True and False tabs. Click the True tab to specify the true link, and the False tab to specify the false link.
6. Select the check boxes adjacent to the nodes to which you want to link; that is, the nodes immediately after the selected node.

7. Click OK.

**Change the True and False Status for a Condition Node**

Unlike other nodes, a condition node must link to two nodes. By default, when you link a condition node, a status of True is allocated to the first link that you create and False to the other. However, you can reverse the logic.

**Steps**

1. Click Select Mode.

2. Click either the True or False link.

3. From the Actions menu, click Reverse Logic.

   The True output link changes to False, and the False to True.

**Split Large JobStreams into Smaller JobStreams**

A JobStream node is one or more JobStreams within a JobStream. This allows larger jobs to be broken down into a series of smaller jobs.

You can convert any single node, or group of nodes within a JobStream to a JobStream node, or you can create a JobStream node.

For information about creating a JobStream node, see "Add a Fact Build, Dimension Build, or JobStream Node to a JobStream" (p. 259).

**Steps to Convert an Existing Node to a JobStream Node**

1. Click the node that you want to convert.

   **Tip:** To select more than one node, hold down the Ctrl key and, in the Visualization pane, click each node. Alternatively, click within the Visualization pane close to the first node that you want to select, keep the mouse button depressed, and drag the pointer so that all the required nodes are selected, and then release the mouse button.

2. From the Actions menu, click Convert to JobStream.

   The Visualization pane is updated to show details of the new JobStream. The JobStream is also added to the Builds and JobStreams folder in the Tree pane.

   **Tip:** By default, the JobStream is named JOBSTREAM. To give the JobStream a more meaningful name, from the Edit menu, click Properties, and type a new name.
Specify Error Handling in JobStream Nodes

You can specify how to handle errors that may be encountered while processing a node. This setting can be different for each node.

Steps

1. Click the node.
2. From the Edit menu, click Properties.
3. Click the Details tab.
4. In the Action on failure box, specify the action to take if a problem is encountered with the node by clicking
   - Terminate to stop processing the JobStream that contains the problem (default)
     If the JobStream has more than one flow, IBM® Cognos® Data Manager continues processing the remaining flows.
   - Abort to prevent further processing when all currently running nodes have completed
   - Continue to skip the selected node and continue processing all the remaining nodes in the JobStream

Specify the Load Required to Process a Fact Build or Dimension Build Node

If a JobStream is set up to execute nodes in parallel, you can restrict the execution of fact build or dimension build nodes to prevent overloading the computer. This is known as load control.

For each computer on which you install IBM® Cognos® Data Manager, you can set a variable that indicates the processing capacity of the computer. The value of this variable is an arbitrary number assigned by you. Within a JobStream, you can then assign a weight value to each fact build node or dimension build node that indicates the processing resource required to execute each node. The weight value is also an arbitrary value that you assign. When you execute the JobStream, the total load for the computer is calculated as the sum of the weight of all build nodes that are concurrently processing.

Note: If weight of a single node is greater than the capacity specified for a computer, Data Manager assumes the weight to be equal to the capacity.

When processing a JobStream containing a weighted fact or dimension build node, Data Manager calculates the current load prior to processing the node. If the weight of this node, added to the current load, exceeds the capacity of the computer, Data Manager defers processing the node until sufficient capacity becomes available.

For example, suppose a JobStream contains three fact build nodes: node1 (weight 40), node2 (weight 40), and node3 (weight 40). The processing capacity is 100.
When the JobStream is executed, node1 and node2 can run but node 3 is deferred until either node1 or node2 is complete, because the added weight of node3 exceeds the capacity of the computer.

Note: Load control does not affect the order of processing dependent nodes in a JobStream.

To enable load control, you must set the variable named DM_SERVER_CAPACITY, which is used to define the processing capacity of a computer. For more information, see "DM_SERVER_CAPACITY" (p. 326).

Steps
1. Click the appropriate fact build or dimension build node.
2. From the Edit menu, click Properties.
3. Click the Details tab.
4. In the Weight box, type a suitable value to indicate the total load required to process the node.
Chapter 20: Publishing Data Movement Tasks

A published data movement task can comprise a JobStream together with all its nodes, or a single fact build or dimension build. Each data movement task is self-contained. When you publish a data movement task from IBM® Cognos® Data Manager, you can then use IBM® Cognos® Connection to run or schedule it. IBM® Cognos® Connection is the portal to IBM® Cognos® Business Intelligence and provides a single access point to all corporate data available in IBM® Cognos® BI.

For more information on using IBM® Cognos® Connection, see the IBM® Cognos® Business Intelligence Administration and Security Guide.

Note: To use the Data Movement Service, the Data Manager engine and the IBM® Cognos® BI server components must be installed in the same location. For information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Steps
1. From the Actions menu, click Publish Data Movement Task.
2. Click the build or JobStream to publish.
   The name of the selected component is shown in the Enter the name for the published task box.
3. If required, amend the name for the published task.
   Note: If the name is not unique within the Data Movement Tasks folder in IBM® Cognos® Connection, a message appears asking if you want to replace the existing file. Click Yes or No as appropriate.
4. Click OK.
5. You can now use IBM® Cognos® Connection to
   - run the entry
   - view the run history
   - change the default properties

For information, see the IBM® Cognos® Administration and Security Guide.
You execute builds and JobStreams to deliver data to the target database.

When you execute a build or JobStream a command window appears showing the progress of the execution process. During execution, progress information is returned to a log file and audit tables.

Builds and JobStreams can be executed

- on your local computer
  For more information, see "Execute a Fact Build or Dimension Build on your Local Computer" (p. 273) or "Execute Procedure and SQL Nodes as Separate Processes" (p. 278).

- on a remote computer
  For more information, see "Execute a Build or JobStream on a Remote Computer" (p. 279).

- using the Data Movement service
  For more information, see "Execute a Build or JobStream using the Data Movement Service" (p. 280).

Note: You can also execute a DataStream. For information, see "Execute a DataStream" (p. 160).

**Build Execution Modes**

When you execute a build on either your local computer or on a remote computer, you can choose from these execution modes:

**Normal Execution Mode**
IBM® Cognos® Data Manager executes the whole build and delivers the required fact data and dimension data.

**Object Creation Execution Mode**
Data Manager creates the required tables but does not deliver any data.

**Check Only Execution Mode**
Data Manager executes the build without delivering data. Check only mode is useful for testing a build and for estimating the resources required.

**Execute a Fact Build or Dimension Build on your Local Computer**

After you define a fact build or a dimension build, you execute it to acquire, transform, and deliver the data.

When a build executes, IBM® Cognos® Data Manager
validates the build specification

checks that the columns in the target tables are the same as those being delivered

finds all the reference structures required by the build

builds the reference structures by running the structural source queries

acquires the transactional data

transforms (calculates, derives, and aggregates) the data in the context of the loaded reference structures

delivers the data

Tip: If you want to test the source data to be acquired, prior to executing a build, you can execute the DataStream for the build. For more information, see "Execute a DataStream" (p. 160).

You can execute a build using the default settings. The default settings are that normal execution mode is used and fact data and dimension data is delivered (if the build contains both). However, you can change these settings from the relevant build Properties window, or you can temporarily change the default settings from the Execute Build dialog box.

**Steps to Use the Default Settings to Execute a Build**

1. Click the build and then click the **Execute** button.

   If you made changes to the build and did not save them, a message appears.

2. Click **OK** to save the changes.

   The build executes using normal execution mode and fact data and dimension data is delivered.

3. When the execution process is complete, press Enter to close the command window and return to Data Manager.

**Steps to Change or Check the Default Settings and Execute a Build**

1. Click the build to execute.

2. From the **Actions** menu, click **Execute**.

   If you made changes to the build and did not save them, a message appears.

3. Click **OK** to save the changes.

4. In the **Options** section, click **Execute locally**, and then, in the **Execution mode** box, click the required execution mode.

5. If you are executing a fact build, you can choose to deliver only fact data, only dimension data, or both (providing the fact build contains both types). In the **Deliver** section, select the check boxes for the type of data to deliver.

6. To override the current log file or trace frequency information, in the **Trace** section, select the **Override build settings** check box. Select the type of information to include in the log file, and change the message frequency values as required.
For more information, see "Specify Log File Details" (p. 283), and "Specify the Message Frequency in a Build Log File" (p. 283).

**Note:** Any changes that you make alter the commands shown in the Command line box.

7. In the Additional options box, type any additional commands that you want to add to the command line.

The command that Data Manager will execute is shown in the Command line box.

**Tip:** By default, the command window remains open when execution is complete, and you must press Enter to close it. To close it automatically, clear the Pause on completion check box.

8. To use the catalog credentials instead of Designer credentials when executing the build, select the Use catalog credentials check box.

For more information on credentials, see "Manage Credentials" (p. 71).

9. Click OK.

A command window appears and shows the progress of the execution process.

---

### Check the Target Table Prior to Executing a Build

When IBM® Cognos® Data Manager executes a build, it automatically checks that the columns in the target table are the same as those being delivered. If they are different, a message appears informing you of the problem.

You can also check the target table independently of an execution process.

**Note:** When checking the target table, Data Manager ignores any tables for which you have specified that Data Manager should automatically add columns to the table. For information, see "Add Missing Columns to the Target Table" (p. 215).

**Tip:** To stop Data Manager automatically checking the tables during execution, from the Tools menu, click Options, and then click Skip table compatibility checks on builds. If you select this option and the tables are incompatible, the build fails.

#### Steps

1. Click the appropriate build.

2. From the Actions menu, click Perform Target Table Check.

   The columns in the target table are checked to ensure that they are the same as those being delivered. If they are the same, or if you have specified that Data Manager should automatically add columns to the table, a message appears stating that there are no known problems. For information, see "Add Missing Columns to the Target Table" (p. 215).

   If there are inconsistencies, this type of message appears.

   Inconsistencies found in the following tables:
   
   Connection 'DS_Marts'
   
   Table 'F_Additional'
   
   Column '"SalesTotal"' missing
3. Choose whether to correct the problem, delete the target table, or add the missing columns to the target table.

- If you do not want to delete the table, click **Cancel**, correct the problem, and repeat steps 1 to 2.
- If you want to add the missing columns, click **Cancel** and in the Properties window for the fact build delivery or the dimension build, click ** Automatically add columns to table**.
- If you want to delete the tables from the target database and recreate them upon delivery of the columns, click **Drop Tables**, and in the message box that appears, click **Yes**.

**Fine-Tune the Data Structure**

When you execute a fact build, the required fact and dimension tables are created if they do not exist in the target data mart. For most purposes, the structure of the created tables is acceptable. However, you can export to a script file data definition language (DDL) statements to create or drop the target tables and indexes. This allows you to fine-tune the DDL statements and create tables and indexes by running script files in SQLTerm or another suitable program.

**Steps**

1. Click the required build.

2. From the **Tools** menu, click **Show DDL**.

3. Click the tab for the required type of DDL statement.

4. Click **Save**.

5. In the **Save DDL** dialog box, type a file name for the script file, and click **Save**.
   - To return to the original DDL statements after you have made changes, click **Reset DDL**.
   - To copy one or more DDL statements to the Microsoft® Windows® clipboard, select the statement, right-click and then click **Copy**.

6. Click **Close**.

**Execute a JobStream on your Local Computer**

When you execute a JobStream, a command window appears showing the progress of the JobStream execution process. As each node is processed it returns progress information to the log file and audit tables.

If, during execution, IBM® Cognos® Data Manager encounters a problem with a node, the execution process may fail. You can specify the action that Data Manager is to take if this happens. When any problems have been resolved you can restart the JobStream starting with the node that failed. For information, see "Specify Error Handling in JobStream Nodes" (p. 269), and "Restart Processing a JobStream" (p. 279).

When you execute a JobStream, you can
• set the level of audit and logging information recorded
• specify how to handle any errors that may be encountered while processing a node
• exclude nodes from processing
• restart processing a JobStream

Steps to Use the Default Settings to Execute a JobStream
1. Click the JobStream and then click the Execute button.
   If you made changes to the JobStream and did not save them, a message appears.
2. Click OK to save the changes.
3. When the execution process is complete, press Enter to close the command window and return to Data Manager.

Steps to Change or Check the Default Settings and Execute a JobStream
1. Click the JobStream to execute.
2. From the Actions menu, click Execute.
   If you made changes to the JobStream and did not save them, a message appears.
3. Click OK to save the changes.
4. In the Options section, click Execute locally.
   Tip: If you want to start executing the JobStream from the point at which a previous attempt failed, select the Restart Last JobStream check box.
5. To override the current log file information, in the Trace section, select the Override Job/JobStream settings check box and then select the type of information to include in the log file.
   Tip: If you are using load control, you can check for deferred JobStream nodes by selecting internal log information.
   For more information, see "Specify Log File Details" (p. 283).
   Note: The trace options you select alter the commands shown in the Command line box.
6. In the Additional options box, type any additional commands that you want to add to the command line.
   The command that Data Manager will execute is shown in the Command line box.
   Tip: By default, the command window remains open when execution is complete, and you must press Enter to close it. To close it automatically, clear the Pause on completion check box.
7. To use the catalog credentials instead of Designer credentials when executing the build, select the Use catalog credentials check box.
   For more information on credentials, see "Manage Credentials" (p. 71).
Click OK. A command window appears and shows the progress of the execution process.

**Exclude Nodes from Processing in a JobStream**

You can specify that you want to exclude from processing either a single node or all the nodes that follow a specified point.

This technique can be used during testing to temporarily disable a node. When a node is disabled, it is shown in the Tree pane and the Visualization pane overlaid with a red line.

**Steps**

1. Click the node to exclude.

2. From the **Edit** menu, click **Properties**.

3. Click the **General** tab.

4. Choose whether you want to exclude only the selected node, or the selected node and all the nodes that follow it:
   - To exclude only the selected node, select the **Exclude this node from processing** check box.
   - To exclude all the nodes that follow the selected node, select the **Exclude subsequent nodes in this thread** check box.

5. Click OK.

**Execute Procedure and SQL Nodes as Separate Processes**

By default, a condition node, procedure node, SQL node, alert node, or email node is executed inline, that is, run in the JobStream process. The implication of running inline is that the nodes are run in series even if the JobStream design has parallel flows. However, for procedure nodes and SQL nodes there may be instances where the nodes may take a long time to process so parallel execution would be desirable. To facilitate this, you can specify that IBM® Cognos® Data Manager should create a separate process for a node.

**Note:** Executing a node as a separate process uses more memory than executing inline.

**Steps**

1. Click the appropriate procedure or SQL node.

2. From the **Edit** menu, click **Properties**.

3. Click the **General** tab.

4. Select the **Run as a separate process** check box.

5. Click OK.
### Restart Processing a JobStream

If a JobStream stops processing and you resolve the problem that caused the failure, you can restart the JobStream beginning with the node that failed.

**Note:** If the failure occurs in a nested JobStream (a JobStream within a JobStream), processing restarts with the top level JobStream node, not with the actual node that failed.

**Tip:** You can specify how IBM® Cognos® Data Manager should handle errors that may be encountered while processing a node. For information, see "Specify Error Handling in JobStream Nodes" (p. 269).

**Steps**

1. Click the JobStream.
2. From the **Actions** menu, click **Execute**.
   
   If you made changes to the JobStream and did not save them, a message appears.
3. Click **OK** to save the changes.
4. Select the **Restart last JobStream** check box, and click **OK**.

### Execute a Build or JobStream on a Remote Computer

To control and manage the execution of builds and JobStreams on remote servers you use IBM® Cognos® Data Manager Network Services.

Data Manager Network Services consists of server and client software. The client software is automatically installed when you install Data Manger, and the server software must be installed on the computer on which the build or JobStream is to be executed. However, both the client and server software can exist on the same computer.

For information on setting up Data Manager Network Services, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Communication is performed using either network sockets or web services.

**Steps**

1. Click the required build or JobStream.
2. From the **Actions** menu, click **Execute**.
   
   If you made changes to the build or JobStream and did not save them, a message appears.
3. Click **OK** to save the changes.
4. Select **Remote execute**, and in the **Server** box, enter the name of the server on which the build or JobStream is to be executed.
5. Select the **Submit only** check box if the client is to submit the command and then exit, rather than waiting for the command output.
6. Enter other details as required.
Chapter 21: Executing Builds and JobStreams

For information, see "Execute a Fact Build or Dimension Build on your Local Computer" (p. 273) or "Execute a JobStream on your Local Computer" (p. 276).

7. Click OK.

Execute a Build or JobStream using the Data Movement Service

Using the Data Movement Service to execute a build or JobStream allows you to run and schedule fact builds, dimension builds, and JobStreams on a remote computer using IBM® Cognos® Connection. IBM® Cognos® Connection is the portal to IBM® Cognos® Business Intelligence and provides a single access point to all corporate data available in IBM® Cognos® BI.

For more information on using IBM® Cognos® Connection, see the IBM® Cognos® Connection User Guide.

Notes

● To use the Data Movement Service, the IBM® Cognos® Data Manager engine and the IBM® Cognos® BI server components must be installed in the same location. For information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

● For information on publishing data movement tasks, see "Publishing Data Movement Tasks" (p. 271).

Steps

1. Click the build or JobStream to execute.

2. From the Actions menu, click Execute.

   If you made changes to the build or JobStream and did not save them, a message appears.

3. Click OK to save the changes.

4. In the Options section, click Execute via Data Movement Service.

   Note: You cannot override the current log file information while executing a build or JobStream via the Data Movement service.

5. Click OK.

   The Execute Data Movement Task window appears. When execution is complete, the top pane shows the name of the fact build, dimension build, or the JobStream and all its nodes, with an icon indicating whether execution was successful or failed, and the start and end timestamps.

   Tip: To stop execution, click Stop Executing.

6. You can choose the audit or log details that you want to view.

   Note: For a JobStream you must first select the required fact build node of dimension build node.

   From the drop down box, click either:

   ● Audit Trail
Configure a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing

If a JobStream contains fact build or dimension build nodes, you can execute these nodes as individual data movement tasks. If your IBM® Cognos® Business Intelligence installation is configured to use more than one dispatcher, it then distributes and executes the Data Movement tasks using standard IBM® Cognos® BI load balancing for parallel builds.

For more information on data movement tasks, see "Publishing Data Movement Tasks" (p. 271).

You can configure a JobStream execution to use IBM® Cognos® BI load balancing:

- using the command line
  You must update the rundsjob options to enable this feature. For more information, see "rundsjob" (p. 419).

- using the Data Movement Service
  You must enable the parameter DMS.DistributeBuilds in IBM® Cognos® Administration. For more information see the IBM® Cognos® Administration and Security Guide.

Run-time Considerations

When configuring JobStream executions to use IBM® Cognos® BI load balancing, you should consider the following:

- Any changes to JobStream variables made by fact build or dimension build nodes executed as separate Data Movement tasks are not reflected in the JobStream.

- Inline JobStream processing always takes place on the host computer. This may cause communications with the Data Movement Service (executing other nodes in the JobStream) to time out. You can overcome this issue by redesigning the JobStream so that the fact and dimension build nodes are moved into a separate JobStream node within the main JobStream.

Set the Default Execution Method

You can specify the default method to use for build and JobStream execution.

Steps

1. From the Tools menu, click Options.
2. Click the General tab.
3. In the **Default execution method** box, click either **Local**, **Remote**, or via **Data Movement Service** as required.

4. If you selected **Remote**, in the **Default server** box, enter the name of the server on which builds and JobStreams are to be executed.

5. Click **OK**.

### Execution Log Files

When you execute a build or JobStream, a command window appears and shows progress information. This information is saved automatically as a log file. By default, log files are stored in the directory named Program Files\ibm\cognos\c10\datamanager\log, however you can change this. For more information, see "Change the Default Location for Log Files" (p. 282).

The file name of each log file is `<component_type>_<component_name>_<number>.log`. Where `<component_type>` is the type of component (either Build, DimBuild, or Job), `<component_name>` is the name of the component, and `<number>` is a four digit number that identifies the execution. For example, the log file for a fact build named NewBuild might be Build_NewBuild_0001.log.

**Note:** If a component name contains characters other than letters and numbers, these are replaced with an underscore in the log file name.

You can specify the level of detail to include in the log files, and view the log files. For information, see "Specify Log File Details" (p. 283), and "View the Log Files" (p. 284).

If you are using IBM® Cognos® Data Manager with multilingual or unicode data, you should specify unicode encoding for your log files. By default, the encoding used is platform dependent. For more information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

### Change the Default Location for Log Files

By default, log files are stored in the directory named Program Files\ibm\cognos\c10\datamanager\log. You can change this using

- the -V parameter with the databuild, dimbuild, or rundsjob command
  
  The following example uses a command-line parameter to direct text data files to directory c:\log\datamanagerr
  
  `databuild ODBC DSN=TestCat example -VDS_LOG_DIR=c:\log\datamanager`

  For information, see "Commands" (p. 413).

- the DS_LOG_DIR system variable
  
  For information, see "DS_LOG_DIR" (p. 327).

- IBM® Cognos® Configuration
  
  For information, see the IBM® Cognos® Configuration User Guide.

Setting the location using the databuild, dimbuild, or rundsjob command takes precedence over a location set using the DS_LOG_DIR system variable or IBM® Cognos® Configuration, and using the DS_LOG_DIR system variable takes precedence over IBM® Cognos® Configuration.
Specify Log File Details

You can specify that IBM® Cognos® Data Manager records these types of information in the log file.

<table>
<thead>
<tr>
<th>Information type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress</td>
<td>Shows the overall progress of the application. In the log file, Data Manager may show a value for Average. For dimension builds, a single Type 1 change may affect many rows. The Average, is the total number of rows affected by all the Type 1 changes, divided by the number of individual Type 1 changes.</td>
</tr>
<tr>
<td>Detail</td>
<td>Shows more detailed progress messages.</td>
</tr>
<tr>
<td>Internal</td>
<td>Shows internal Data Manager activity messages, including resource usage. For example, memory usage, dimension sizes, and paging information.</td>
</tr>
<tr>
<td>SQL</td>
<td>Shows all the SQL statements used at each stage of execution. This information can help resolve database errors.</td>
</tr>
<tr>
<td>ExecutedSQL</td>
<td>Shows the executed SQL for SELECT statements.</td>
</tr>
<tr>
<td>User</td>
<td>Shows all the application messages written to the log file. You can include additional application messages using the LogMsg function.</td>
</tr>
<tr>
<td>Variable</td>
<td>Shows all the variables used in the build or JobStream.</td>
</tr>
</tbody>
</table>

Steps

1. Click the required build or JobStream.
2. From the Edit menu, click Properties.
3. Click the Logging tab.
4. In the Trace box, select the check boxes for the types of information that you want to include in the log file.

When you execute the build or JobStream, you can override the values that you select here. For more information, see "Execute a Fact Build or Dimension Build on your Local Computer" (p. 273) or "Execute a JobStream on your Local Computer" (p. 276).
5. Click OK.

Specify the Message Frequency in a Build Log File

For builds, you can specify the frequency of input, output, and domain messages.
The type of messages used in the log files differ between fact builds and dimension builds.

<table>
<thead>
<tr>
<th>Message type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Used in fact builds only.</td>
</tr>
<tr>
<td>Output</td>
<td>Used in fact builds when writing cached rows, and in dimension builds as an indication of general progress.</td>
</tr>
<tr>
<td>Domain</td>
<td>Used in fact builds and dimension builds when caching reference data. They are also used in fact builds with dimension elements that use a reference domain.</td>
</tr>
</tbody>
</table>

**Steps**

1. Click the required build.
2. From the **Edit** menu, click **Properties**.
3. Click the **Logging** tab.
4. In the **Message frequency (input)** box, type the number of input rows that IBM® Cognos® Data Manager is to process between Detail and Internal messages.
   
   **Note:** This box is not available for dimension builds.
5. In the **Message frequency (output)** box, type the number of output rows to process between Detail messages.
6. In the **Message frequency (domain)** box, type the number of domain members that Data Manager is to process between Detail messages.
7. Click **OK**.

**Include User Log Messages for JobStream Nodes**

When executing a master JobStream, you may want to include user messages for a JobStream node being executed within it.

**Steps**

1. For the master JobStream and the appropriate JobStream node, specify that User detail messages are to be included in the log files.
   
   For information, see "Specify Log File Details" (p. 283).
2. In the **Execute JobStream** dialog box, type `-M` in the **Additional options** box.
   
   For information, see "rundsjob" (p. 419).

**View the Log Files**

You can view log files from in the Audit window, or directly from the Log directory.
If the build or JobStream was remotely executed, the log file is retrieved from the remote computer. To retrieve remote log files, you must be using IBM® Cognos® Data Manager Network Services on the server and it must be configured correctly. For more information on Data Manager Network Services, see “Execute a Build or JobStream on a Remote Computer” (p. 279), and for more information on configuration, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Steps to View Log Files From the Audit Window
1. Click the required build or JobStream.
2. From the Actions menu, click Audit.
3. Select the execution for which you want to view the log files, and then click Audit Log.
   - Click Find to search for specific information in the log file.
   - Click Save to save the log information to a text file.
   For information on the Audit window, see "The Audit History" (p. 285).
4. When you have finished, click Close.

Step to View Log Files From the Log Directory
- From the Tools menu, click Browse Log Files.

Save Audit Details
When you execute a build or JobStream, you can specify that the events performed are recorded in a series of audit tables.

The Audit window shows the full audit history for the selected build or JobStream. From the Audit window, you can choose to view audit trail details, audit messages, and JobStream details.

You can also access the audit tables using reporting tools. For details on the structure of the IBM® Cognos® Data Manager audit tables, see "Create an IBM Cognos Data Manager Catalog Schema" (p. 46).

The Audit History
The audit history provides information for each occurrence of a build or JobStream execution. The execution process can be performed locally or remotely. If a JobStream calls itself during execution, each execution is listed separately.

You can use a filter to restrict the list of execution processes to those performed
- in the last 24 hours
- in the 7 days
- in the last month
- in the last 3 months
Chapter 21: Executing Builds and JobStreams

- in the last year

Each filter that you set up is automatically applied to all fact builds, dimension builds, and JobStreams.

For each execution process listed in the audit history, the following information is provided.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status icon</td>
<td>- ✅ indicates the node is running</td>
</tr>
<tr>
<td></td>
<td>- ✔ indicates the node reached a successful completion</td>
</tr>
<tr>
<td></td>
<td>- ❌ indicates the node failed due to errors</td>
</tr>
<tr>
<td></td>
<td>- ☐ indicates the execution process was successfully manually stopped</td>
</tr>
<tr>
<td></td>
<td>- ☐ indicates the execution process had failed prior to being manually</td>
</tr>
<tr>
<td></td>
<td>stopped</td>
</tr>
<tr>
<td></td>
<td>For more information on stopping a JobStream execution, see &quot;Stop</td>
</tr>
<tr>
<td></td>
<td>Execution&quot; (p. 291).</td>
</tr>
<tr>
<td>Start TimeStamp</td>
<td>The start date and time of the execution process.</td>
</tr>
<tr>
<td>Elapsed</td>
<td>The elapsed time for the execution process.</td>
</tr>
<tr>
<td></td>
<td>Note: The elapsed time is only available when the execution process is</td>
</tr>
<tr>
<td></td>
<td>complete.</td>
</tr>
<tr>
<td>Run Mode</td>
<td>The run mode of the build and the delivery types:</td>
</tr>
<tr>
<td></td>
<td>- N indicates normal execution mode</td>
</tr>
<tr>
<td></td>
<td>- C indicates check only execution mode</td>
</tr>
<tr>
<td></td>
<td>- O indicates object creation execution mode</td>
</tr>
<tr>
<td></td>
<td>- F indicates a fact delivery</td>
</tr>
<tr>
<td></td>
<td>- D indicates a dimension delivery</td>
</tr>
<tr>
<td>Full Refresh</td>
<td>A Boolean value to indicate whether all the data in the table was</td>
</tr>
<tr>
<td></td>
<td>refreshed:</td>
</tr>
<tr>
<td></td>
<td>- Y indicates a full refresh was performed</td>
</tr>
<tr>
<td></td>
<td>- N indicates a full refresh was not performed</td>
</tr>
<tr>
<td>Rows Inserted</td>
<td>The number of rows that were successfully inserted.</td>
</tr>
<tr>
<td>Rows Updated</td>
<td>The number of rows that were successfully updated.</td>
</tr>
<tr>
<td>Connection</td>
<td>The name of the database connection to which the data was delivered.</td>
</tr>
</tbody>
</table>
### Item | Description
--- | ---
Table Name | The name of the table into which the data was delivered.
Host Name | The computer on which the build or JobStream was executed.

**Note:** For JobStreams, information is provided for only the status icon, Start TimeStamp, Elapsed, and Host Name.

The audit history retrieves information from the dsb_component_run, dsb_run_context, and dsb_delivery history tables. For more information, see "Component Run Table (dsb_component_run)" (p. 531), "Run Context Table (dsb_run_context)" (p. 532), and "Delivery History Table (dsb_delivery_hist)" (p. 533).

### Steps to View the Audit History
1. Click the required build or JobStream.
2. From the Actions menu, click Audit.
3. In the Date Range box, click the filter you want to use.
4. Click the Execute button to execute the filter.
   **Tip:** To interrupt a filter from executing, click the Interrupt button.
5. When you have finished, click Close.

### The Audit Trail
For each fact build, dimension build, or JobStream you can specify that audit trail information is recorded during execution. You do this from the Properties window for the selected build or JobStream.

### Specify the Audit Trail Details to be Recorded
You can specify that these types of information are recorded:

- **Timing** records overall timing information. For example, the start, end, and elapsed times for the execution process.
  This information is available for fact builds, dimension builds, and JobStreams.

- **Acquire** records information about the source data acquired. For example, the number of data rows that each data source contributes.
  This information is available for fact builds only.

- **Transform** records information about the data processed. For example, the number of data rows received from the data sources.
  This information is available for fact builds only.

- **Deliver** records information about data delivery. For example, the number of data rows delivered.
This information is available for fact builds and dimension builds.

- Internal records information about the internal measures of performance. For example, the size of the hash table.
  
  This information is available for fact builds only.

- Alert records a user-defined audit record of type ALERT, into the IBM® Cognos® Data Manager audit tables.
  
  This information is available for fact builds, dimension builds, and JobStreams.

- User records user-controlled audit messages.
  
  This information is available for fact builds, dimension builds, and JobStreams.

**Steps to Specify that Audit Trail Information is to be Recorded**

1. Click the required build or JobStream.

2. From the Edit menu, click Properties.

3. Click the Audit tab.

4. Select the check boxes for the events that you want to record.

5. Click OK.

**View the Audit Trail**

You can view the following audit trail details.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audit Timestamp</td>
<td>The start time that each row of data was written.</td>
</tr>
<tr>
<td>Audit Group</td>
<td>The audit group to which information is written.</td>
</tr>
<tr>
<td>Audit Item</td>
<td>The audit item of the group to which information is written.</td>
</tr>
<tr>
<td>Audit Message</td>
<td>The message from the audit item.</td>
</tr>
</tbody>
</table>

The audit trail retrieves information from the dsb_audit_trail and dsb_component_run tables. For more information, see "Audit Trail Table (dsb_audit_trail)" (p. 533), and "Component Run Table (dsb_component_run)" (p. 531).

**Steps to View the Audit Trail Details**

1. Click the required build or JobStream.

2. From the Actions menu, click Audit.

3. Select the execution process for which you want to view the audit trail, and then click Audit Trail.
4. When you have finished, click Close.

**The Audit Messages**

You can drill down from a build or JobStream execution in the audit history to view any audit messages that were recorded during the execution process.

By default, all the audit messages recorded during the execution process are included. You can use a filter to restrict the number of audit messages shown.

**Note:** When you set up a filter, it is automatically applied to all fact builds, dimension builds, and JobStreams.

The following details are provided for each audit message.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>A unique number that identifies the message that is recorded.</td>
</tr>
<tr>
<td>Severity</td>
<td>The severity of the message:</td>
</tr>
<tr>
<td></td>
<td>● F indicates a fatal message</td>
</tr>
<tr>
<td></td>
<td>● E indicates an error</td>
</tr>
<tr>
<td></td>
<td>● W indicates a warning</td>
</tr>
<tr>
<td>Message Text</td>
<td>The message text.</td>
</tr>
</tbody>
</table>

The audit message retrieves information from the dsb_audit_msg_line and dsb_audit_msg tables. For more information, see "Audit Message Line Table (dsb_audit_msg_line)" (p. 534), and "Audit Message Table (dsb_audit_msg)" (p. 534).

**Steps to View the Audit Messages**

1. Click the required build or JobStream.

2. From the **Actions** menu, click **Audit**.

3. Select the execution process for which you want to view audit messages, and then click **Audit Messages**.

4. In the **Number of rows to return** box, type the number of messages you want to view.

5. Click the **Execute** button to execute the filter.
   - To interrupt a filter from executing, click the **Interrupt** button.
   - To clear a filter, delete the value in the **Number of rows to return** box, and then click the **Execute** button.

6. When you have finished, click **Close**.
## The JobStream Details

You can drill down from a JobStream execution in the audit history to view the individual nodes executed. For each node listed, the following details are provided.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status icon</td>
<td>Indicates the node is running</td>
</tr>
<tr>
<td></td>
<td>Indicates the node reached a successful completion</td>
</tr>
<tr>
<td></td>
<td>Indicates the node failed due to errors</td>
</tr>
<tr>
<td></td>
<td>Indicates the execution process was successfully manually stopped</td>
</tr>
<tr>
<td></td>
<td>Indicates the execution process had failed prior to being manually stopped</td>
</tr>
<tr>
<td></td>
<td>For more information on stopping a JobStream execution, see &quot;Stop Execution&quot; (p. 291).</td>
</tr>
</tbody>
</table>

### Notes

- Nodes are listed in their order of execution.
- If the JobStream includes a JobStream node, the individual nodes in that JobStream are included.
- If the JobStream includes a node that is repeatedly executed using a loop, the node is listed each time it is executed.

Information about executed nodes is retrieved from the dsb_jobnode_run table. For more information, see "Job Node Run Table (dsb_jobnode_run)" (p. 535).

### Steps to View Node Details

1. Click the required JobStream.
2. From the **Actions** menu, click **Audit**.
3. Click the JobStream execution for which you want to view executed nodes, and then click JobStream Details.

   Tip: Click Refresh to update the list of executed nodes whilst a JobStream is still executing.
   If there are fact build or dimension build nodes listed, you can view further audit information for these.

4. Click the required fact build node or dimension build node and click one of the following audit options:
   ● Audit Trail
   ● Audit Log
   ● Audit Messages

   For more information on the audit information available for a build, see "View the Log Files" (p. 284), "View the Audit Trail" (p. 288), and "The Audit Messages" (p. 289).

5. When you have finished, click Close.

Stop Execution

If a build or JobStream is being remotely executed, you can manually stop it if the status is running (indicated by a white triangle in a blue circle). When you stop a build or JobStream, the status is updated in the audit history and the audit tables.

For more information on how the status is updated, see "The Audit History" (p. 285) and "The Audit Tables" (p. 291)

Steps
1. Click the required build or JobStream.
2. From the Actions menu, click Audit.
3. Click the component that is currently running, and then click Stop Execution.

The Audit Tables

The audit tables are used to provide summary information about the audit history, the audit trail, the audit messages, and the JobStream details.

There are the following audit tables:

- component run (dsb_component_run)
- context run (dsb_run_context)
- delivery history (dsb_delivery_hist)
- audit trail (dsb_audit_trail)
- audit message (dsb_audit_msg)
- audit message line (dsb_audit_msg_line)
Create a Log File for Expression and Script Tracing

When you execute a fact build, dimension build, or JobStream, it is possible to create an additional log file that traces any expressions and scripts, user-defined functions, and IBM® Cognos® Data Manager predefined functions. This is useful when a lower level of debugging is required for complex transformation logic.

If you are executing a fact build or a dimension build, Data Manager creates one log file for the trace information. If you are executing a JobStream, Data Manager creates a separate log file for each executed node.

By default, trace log files are stored in the directory named Program Files\ibm\cognos\c10\datamanager\log, however you can change this. For more information, see "Change the Default Location for Log Files" (p. 282).

The file name of each trace log file is <component_type>_ <build_name>_<number>_trace.log, where <component_type> is either Build, DimBuild, or Job, <component_name> is the name of the build and <number> is a four digit number that identifies the execution. For example, the trace log file for a fact build named NewBuild might be Build_NewBuild_0001_trace.log.

Note: If a build name contains characters other than letters and numbers, these are replaced with an underscore in the trace log file name.

Specify Trace Log File Details

These levels of detail can be included in the trace log file:

- Basic shows the input and output values for items contained in the expression, script, or user-defined function.
- Datatype shows the data type for all input and output values displayed.
- Builtin shows the IBM® Cognos® Data Manager predefined functions used in the build or JobStream, the components from which they are called, and the values assigned to those functions.
- Variable shows the variables called by an expression, script or function, the components in which the variables are declared, and values assigned to those variables.

Steps

1. If the build or JobStream contains a lot of source data, you should apply a maximum row limit so that the resulting trace log file is smaller.
   For information, see "Apply a Maximum Row Limit" (p. 158).

2. Click the required build or JobStream.

3. From the Actions menu, click Execute.
If you made changes to the build or JobStream and did not save them, a message appears.

4. Click **OK** to save the changes.

5. In the **Additional options** box, type the following string to invoke trace logging, where `<OPTION>` is the level of detail you require: BASIC, DATATYPE, BUILTIN, or VARIABLE.

   - `VEXP_TRACE_VALUES=<OPTION, [OPTION]>`

   For example, to include the BASIC and DATATYPE levels, type

   - `VEXP_TRACE_VALUES=BASIC,DATATYPE`

   - You can type `*` to include all levels of detail.
   - By default, the command window remains open when execution is complete, and you must press Enter to close it. To close it automatically, clear the **Pause on completion** check box.

6. Click **OK**.

A command window appears and shows the progress of the execution process.

**Example Trace Log Files**

The following example illustrates a trace log file that IBM® Cognos® Data Manager generates if the basic level of detail is selected. The fact build named Example build contains one user-defined function named Margin, and two DataStream derivations, named SalesTotal and GrossProfit.

```
***** START TRACING Basic *****
=> Datastream SalesTotal
<= Datastream SalesTotal -> 553.56
=> Datastream GrossProfit
<= Datastream GrossProfit -> 185.64
=> Example GrossMargin
   => Margin( 6.59, 4.38 )
     <= Margin -> 0.33535660091047
     <= Example GrossMargin -> 0.33535660091047
<= Datastream SalesTotal
<= Datastream SalesTotal -> 830.34
<= Datastream GrossProfit
<= Datastream GrossProfit -> 278.46
=> Example GrossMargin
   => Margin( 6.59, 4.38 )
     <= Margin -> 0.33535660091047
     <= Example GrossMargin -> 0.33535660091047
***** END TRACE *****
```

In lines 1 and 2, you can see that the initial input value of SalesTotal is Null, and the output value is 553.56. Similarly, the initial input value of GrossProfit is Null, and the output value is 185.64.

In lines 5 to 8, you can see that the GrossMargin expression calls the Margin user-defined function. The input value and output values of Margin are shown, and the resulting output of GrossMargin is shown.
In the next example, the fact build named Example is executed again, but with the basic and datatype levels selected.

***** START TRACING Basic, Datatype *****

==>> Datastream SalesTotal
<<== Datastream SalesTotal -> FLOAT 553.56
==>> Datastream GrossProfit
<<== Datastream GrossProfit -> FLOAT 185.64
==>> Example GrossMargin

==>> Margin( Price INTEGER, FLOAT or NUMBER 6.59,
Cost INTEGER, FLOAT or NUMBER 4.38 )
<<== Margin -> FLOAT 0.33535660091047
<<== Example GrossMargin -> FLOAT 0.33535660091047
==>> Datastream SalesTotal
<<== Datastream SalesTotal -> FLOAT 830.34
==>> Datastream GrossProfit
<<== Datastream GrossProfit -> FLOAT 278.46
==>> Example GrossMargin

==>> Margin( Price INTEGER, FLOAT or NUMBER 6.59,
Cost INTEGER, FLOAT or NUMBER 4.38 )
<<== Margin -> FLOAT 0.33535660091047
<<== Example GrossMargin -> FLOAT 0.33535660091047
***** END TRACE *****

In addition to the basic details, the data type for each derivation and user-defined function is shown.

In the following extract, a build containing a Data Manager predefined function is executed, and the basic and builtin levels are selected.

==>> Datasource DataSource1
<==> Concat( 'TrailChef Water Bag', ' :', 'Cost :',
4 ) -> 'TrailChef Water Bag :Cost :4'
<<== Datasource DataSource1 -> 'TrailChef Water Bag :Cost :4'

You can see that DataSource1 calls the Concat function. The output values of Concat are shown, and the resulting output of DataSource1 is also shown.

The final example shows the trace log file extract for an executed build, where the basic and variable levels are selected.

==>> Datastream SalesTotal
<<== Datastream SalesTotal -> 553.56
==>> Datastream GrossProfit
<<== Datastream GrossProfit -> 185.64
==>> Datastream Derivation1

TestVariable [Additional] = '2005-08-16 00:00:00'
<<== Datastream TestDerivation -> '2005-08-16 00:00:00'

You can see that the variable named TestVariable is declared in the fact build named Additional, and assigned a value of 2005-08-16 00:00:00. It is then called in the DataStream derivation named TestDerivation.
Chapter 22: Exporting Metadata

You can create descriptions of the conformed models in a data mart or data warehouse. When you have created the conformed model descriptions, you export the metadata to an XML file. This file can then be used to produce a model of your target data mart or data warehouse in IBM® Cognos® Framework Manager, without the need to recreate the complete model.

A conformed model contains dimensions that are shared by more than one fact table. Fact tables are grouped together using these shared dimensions to form stars. Similarly, stars are grouped together into subject areas to form collections.

An example of a conformed model is a customer dimension used by a fact table within a business subject area named Sales Analysis Reporting.

When you have created export models, they can be used by Framework Manager for deployment to IBM® Cognos® Business Intelligence.

To set up metadata you must

- create the metadata dimensions
- create the metadata collections
- add star models to a metadata collection

View Metadata

You can view metadata in the Tree pane or the Visualization pane.

View Metadata in the Tree Pane

Metadata consists of a number of objects grouped together in the Metadata folder. Metadata is represented as follows.

Tip: You can view just the metadata contained in a catalog by clicking the Metadata tab at the bottom of the Tree pane.

View Metadata in the Visualization Pane

When you click on a metadata object in the Tree pane, a visual representation appears in the Visualization pane.
If you click the **Metadata** folder, a visual representation of the whole target conformed model appears.

![Diagram 1](image1)

If you click a specific metadata dimension, a visual representation of the dimension, together with the fact tables referenced to it, appears.

![Diagram 2](image2)

If you click a specific metadata collection, a visual representation of the star models contained in the collection appears.

![Diagram 3](image3)

If you click a specific metadata star, a visual representation of the star, together with the metadata dimensions it references, appears.

![Diagram 4](image4)
Create a Metadata Dimension

A metadata dimension contains the description of the conformed dimension that you want to include in a metadata export.

You can create a metadata dimension from the Metadata folder or from a dimension build.

When you create a metadata dimension from a dimension build, you are indicating that the dimension tables form part of the conformed mart, and that they are available to be referenced by fact tables. Any dimension tables that you add become available for export. You should identify which dimensions you would like to include in your model before creating fact table stars. This makes it easier to automatically associate fact tables with existing metadata dimension tables.

You must create a separate metadata dimension for each conformed dimension that you want to export.

The metadata dimension is a representation of the dimension build delivery table. It does not matter if the dimension is a star (single table delivery for all levels) or some form of snowflake (multiple table delivery), it is still logically represented in the model as a single metadata dimension. The exported metadata will include an object for each of the levels in the snowflake.

Note: It is not possible to directly model parent-child relationships using IBM® Cognos® Framework Manager. If your catalog contains a conformed dimension that references an auto-level hierarchy, you must create a lookup instead of the auto-level hierarchy, and reference the lookup from the conformed dimension. You can then correctly include the description of the conformed dimension in the metadata export.

Steps to Create a Metadata Dimension from the Metadata Folder

1. In the Metadata folder, click Dimensions.
2. From the Insert menu, click Metadata, and then click Dimension.
   The Metadata Dimension Properties window appears.
3. Type a name for the metadata dimension and, if required, a business name and description.
4. Click the Details tab.
5. In the Select Dimension Table(s) box, expand the appropriate folder if required, and then expand the appropriate dimension build to see the dimension tables it contains.
6. If you want to include all the columns in a dimension table by default, select the dimension tables used to deliver the conformed dimension.
   If you want to include only specific columns from a dimension table, expand the appropriate dimension tables, and then select the required columns.
7. Click OK.

Steps to Create a Metadata Dimension from a Dimension Build

1. Click the dimension build for which you want to add metadata.
2. From the Actions menu, click Add to Metadata.
The **Add Metadata Dimension** window appears.

By default, IBM® Cognos® Data Manager selects all the dimension tables used by the dimension build. All the columns contained in each dimension table are also selected by default.

3. If you want to exclude specific columns from a dimension table, expand the appropriate dimension table, and then clear the required columns.

4. Click **Next**.

5. Type a name for the metadata dimension and, if required, a business name and description.

6. Click **Finish**.

Data Manager adds the metadata dimension to the Metadata folder.

### Create a Metadata Collection

A metadata collection groups together all the metadata star models that you want to include in a metadata export. A metadata collection can contain an entire data warehouse model, or a subset of a model, for example a single subject area.

Although it is not necessary to create a separate collection for each subject area, it is advisable to do so. Collections facilitate grouping together stars that have a common meaning. For example, a Sales Analysis Reporting subject area might contain one star for sales orders, and another star for sales shipments. By grouping these stars together in a collection, it is easier to view the content of a large metadata export, together with its associated references.

Note: Collections are only used to group together stars within IBM® Cognos® Data Manager Designer. They are not included in a metadata export file.

**Steps**

1. In the **Metadata** folder, click **Collections**.

2. From the **Insert** menu, click **Metadata**, and then click **Collection**.

   The **Metadata Collection Properties** window appears.

3. Click **OK**.

   You can now add one or more metadata star models to the collection. For more information, see "Add a Star Model to Metadata Collection" (p. 298).

### Add a Star Model to Metadata Collection

A metadata star model contains the description of the fact table that you want to include in the metadata export, with reference to the metadata dimensions you have set up. You must set up a separate star model for each fact table sharing a conformed dimension.

When you are setting up a metadata star model, you must include the details of the star elements.
You use the following columns to specify the star details:

- **Element**
  IBM® Cognos® Data Manager automatically includes a star element for each corresponding transformation model element in the selected fact table.

- **Is Dimensional**
  By default, any transformation model elements set up as dimension elements in the fact table, are marked as dimensional columns in a metadata star description. This indicates that, in the model, these items should be considered as foreign key columns to other tables.

- **Details**
  Data Manager automatically links any dimensional column to its associated column in a dimension table if
  - the dimension table is already set up as a metadata dimension in the Metadata folder
  - the element in the fact build references a hierarchy or a lookup that uses a template for data access

**Note:** By setting up shared dimensions as metadata dimensions in the Metadata folder before creating stars, Data Manager can automatically add the links for you. For more information, see "Steps to Create a Metadata Dimension from a Dimension Build" (p. 297).

If the fact table that you want to include in the metadata export has been partitioned, the columns in the table with the primary partition are automatically selected. However, you can also choose to include those columns that are in tables with secondary partitioning. For more information, see "Partition Columns Horizontally Across a Table" (p. 221).

You can create a star model from the Metadata folder or from a fact build.

**Steps to Set Up a Star Model from the Metadata Folder**

1. Click the metadata collection to which you want to add a star model.
2. From the Insert menu, click Metadata, and then click Star.
   The Metadata Star Properties window appears.
3. Type a name for the metadata star, and if required, a business name and description.
4. Click the Details tab.
5. In the Fact table box, click the Browse button.
6. Expand the appropriate fact build, select the required fact table, and then click OK.
Only fact table deliveries are shown, as you cannot use text deliveries to create a star model.

If the table has partitioned deliveries, all the elements for all the partitioned tables are shown.

The Star Elements box lists all the transformation model elements from the fact table that will be included in the star model.

Tip: To exclude an element from the model, clear the check box to the left of the element name.

For each dimension element listed, a check mark appears in the Is Dimensional column.

7. If required, you can manually link elements marked as dimensional columns. Select the relevant dimension element, and click the Browse button that appears in the Details column.

A list of all metadata dimensions is shown in the Select the Dimension Column dialog box.

8. Expand the appropriate dimension table, select the required dimensional column, and then click OK.

Note: Only those columns that are included in the metadata dimension table are shown.

9. Click OK.

Steps to Set Up a Star Model From a Fact Build

1. Click the fact build for which you want to add metadata.

2. From the Actions menu, click Add to Metadata.

   The Add Metadata Star window appears.

3. Expand the build, and click the required fact table.

   Note: Only fact table deliveries are shown, as you cannot use text deliveries to create a star model.

4. In the Metadata Collection box, click the collection to which you want to add the star model.

   Tip: Click New to create a collection.

5. Click Next.

   All the transformation model elements from the fact table that will be included in the star model are listed. If the table has partitioned deliveries, all the elements for all the partitioned tables are shown.

   Tip: To exclude an element from the model, clear the check box to the left of the element name.

   For each dimension element listed, a check mark appears in the Is Dimensional column.

   If required, you can manually link elements marked as dimensional columns.

6. Select the relevant dimension element, and click the Browse button in the Details column.

   A list of all metadata dimensions is shown in the Select the Dimension Column dialog box.

   Note: Only those columns that are included in the metadata dimension table are shown.
7. Expand the appropriate dimension table, select the required dimensional column, and then click OK.

8. Click Next.

9. Type a name for the metadata star and, if required, a business name and description.

10. Click Finish.

   Data Manager adds the metadata star to the Metadata folder.

**Export Metadata**

When you have set up the metadata, you can export the entire conformed model, or a subset of it, to an XML model.

**Steps**

1. Click the Metadata folder in the Tree pane.

2. From the Actions menu, click Export Metadata.

   The Export Metadata window appears, listing all the metadata collections in the conformed model. By default, all collections are selected for export, and unreferenced dimensions are included in the export.

3. To exclude a metadata collection from the export, in the Metadata Collection box, clear the appropriate check box.

4. Type a name for the metadata export and, if required, add a description.

5. If there are metadata dimensions included in the Metadata folder that are not linked to any stars (they are unreferenced) you can choose whether or not to include them in the export file. By default, unreferenced dimensions are included.

   You should include unreferenced dimensions if you want to export definitions for dimensions that may link to tables that already exist in the data mart but are not managed by IBM® Cognos® Data Manager. In this case, you can use IBM® Cognos® Framework Manager to link these tables.

   To exclude unreferenced dimensions from the export, clear the Include referenced dimensions box.

6. Click OK.

   The Export Metadata dialog box appears.

7. In the File name box, type a name for the metadata export file.

8. In the Save as type box, click Metadata Export Files (*.xml).

9. Click Save.

   You can now use Framework Manager to import the conformed model.
The following table defines how metadata model attributes that are published to Framework Manager are derived from Data Manager catalog components. You can use the following tables to determine which attributes must be maintained for them to be automatically included as part of the target XML based model.

<table>
<thead>
<tr>
<th>Object</th>
<th>Item</th>
<th>Comes from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Model name</td>
<td>The name attribute provided when the model is exported. It is not persisted in the source catalog.</td>
</tr>
<tr>
<td>Connections</td>
<td>Connection name</td>
<td>The alias name of the connection.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Fact table name</td>
<td>The name of the fact table delivery. When a table is part of a partition, the logical table name is taken from the name of the primary partition.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Dimension table name</td>
<td>The name of the table or tables defined in the dimension table delivery in the dimension build.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Fact table column name</td>
<td>The column name attribute from the fact table delivery.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Dimension table column name</td>
<td>The delivery template attribute name from the dimension build that delivers the dimension.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Dimension column usage</td>
<td>The behavior attribute from the delivery template.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Fact column usage</td>
<td>The transformation model element type.</td>
</tr>
<tr>
<td>Physical metadata</td>
<td>Fact/dimension column joins</td>
<td>The metadata star definition which is derived automatically from the fact build references to the reference dimensions. Can be overridden in the metadata star definition.</td>
</tr>
</tbody>
</table>
### Chapter 22: Exporting Metadata

#### Object | Item | Comes from...
--- | --- | ---
Physical metadata | Column data types | Derived automatically by Data Manager based on the target database mapping rules.

#### Object | Item | Comes from...
--- | --- | ---
Business metadata | Fact table business name | The business name attribute of the metadata star.
Business metadata | Fact table description | The description name attribute of the metadata star.
Business metadata | Dimension table business name | The business name attribute of the metadata dimension.
Business metadata | Dimension table description | The description attribute of the metadata dimension.
Business metadata | Fact business name | The business name attribute of the transformation model element.
Business metadata | Fact description | The description attribute of the transformation model element.
Business metadata | Fact aggregation | For measure transformation model elements only. The aggregation rule of the transformation model element.
Business metadata | Dimension attribute business name | The business name attribute of the delivery template attribute.
Business metadata | Dimension attribute description | The description attribute of the delivery template attribute.
Business metadata | Hierarchy name | The business name attribute of the source hierarchy.
Business metadata | Hierarchy description | The description attribute of the source hierarchy.
Business metadata | Hierarchy level name | The business name attribute of the source hierarchy level.
<table>
<thead>
<tr>
<th>Object</th>
<th>Item</th>
<th>Comes from...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business metadata</td>
<td>Hierarchy level description</td>
<td>The description attribute of the source hierarchy level.</td>
</tr>
</tbody>
</table>
Chapter 23: User-Defined Functions

You can use functions to
- calculate derivations and derived dimensions
- define DataStream filters
- define output filters for fact deliveries
- define JobStream procedure nodes and condition nodes
- initialize build or JobStream variables

You can add to the extensive library of predefined IBM® Cognos® Data Manager functions by creating user-defined functions that you can use as often as you require. You do not have to copy user-defined functions between Data Manager actions, or retype them, you can simply call functions by selecting them from the pop-up menu in the relevant Data Manager window.

**Note:** You can also call a user-defined function from within itself.

There are two types of user-defined functions
- **internal** functions that are defined within Data Manager
- **external** functions that are defined within run-time libraries

User-defined functions for the open catalog are shown in the Functions folder in the Library tree.

**Note:** You can find samples of both internal and external user-defined functions in the sample catalogs described in IBM® Cognos® Data Manager *Getting Started*.

**Internal User-Defined Functions**

Internal user-defined functions are defined within IBM® Cognos® Data Manager. They are nestable, reusable functions. You can create them by combining existing Data Manager functions, creating functions, or a combination of both these methods.

**Create an Internal User-Defined Function**

You create an internal user-defined function within IBM® Cognos® Data Manager.

**Steps**

1. In the **Library** folder, click **Functions**.
2. From the **Insert** menu, click **Library**, and then click **Function**.
3. Click the **General** tab.
4. In the **Name** box, type a unique name for the function.
Tip: To ensure that this name is different from the built-in Data Manager functions, it is recommended that you precede the name of each user-defined function with the letters udf, for example, udf.newfunction

5. If required, type a business name and description.

6. In the Return type box, click the data type to be returned by the function.
   For a description of each data type, see "Data Types" (p. 543).

7. Click the Interface tab.

8. For each argument that you want to add, click Add.

   Note: Each user-defined function can contain a maximum of 16 arguments. If you want to use more than 16 arguments, or define a user-defined function that has a variable number of arguments, then you can use an argument of type array.

   By default, Data Manager names each argument, Argumentn (where n is a unique sequential number), and allocates an argument type of CHAR.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Argument Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument1</td>
<td>CHAR</td>
</tr>
<tr>
<td>Argument2</td>
<td>CHAR</td>
</tr>
</tbody>
</table>

9. Change each argument name to something more meaningful, because this is the name that you will use when you implement the function.

10. If a function has more than one argument, the order of the arguments on this tab indicates the order in which you must enter the values when you use the function, you can change this order if you require by clicking an argument and then clicking Move Up or Move Down.

11. For each argument, click within the Argument type column, then click on the arrow that appears to show the available data types. Click the required data type.

   For information about argument types, see "Data Types" (p. 543).

   Tip: To delete an argument, click the argument and then click Delete.

12. If required, create the variables required for use within the function.

    For more information, see "Variables" (p. 319).

13. Click the Implementation tab.

14. Click Internal.

15. In the right pane, enter the syntax for the function.

    You can type the syntax, and you can use the tree structure in the left pane to select the items.

    You can select an item from a folder by dragging it to the right pane, or by double-clicking it.

    For information on operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide.

16. If required, test the function.

    For information, see "Testing Expressions and Scripts" (p. 341).
17. Click OK.

Example

This example describes how to create an internal user-defined function that determines whether telephone numbers are European toll-free numbers.

Steps

1. In the Library folder, click Functions.
2. From the Insert menu, click Library, and then click Function.
3. Click the General tab. In the Name box, type FreephoneNumbers
4. In the Return type box, click Boolean.
5. Click the Interface tab. Click Add to add an argument, and name it TelephoneNo. Click String as the argument type.

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Argument Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TelephoneNo</td>
<td>String</td>
</tr>
</tbody>
</table>

6. Click the Implementation tab. Select Internal and, in the Calculation box, type this syntax.

   Left(TelephoneNo,4)='0800'

   This function determines whether the first four characters of the telephone number are 0800, which designates a toll-free number.

7. Click Test.

   The Values for Expression window appears.

8. In the Input values box, click the Type column and click Char, and then click the Value column and type a number that starts with 0800; for example, 0800101010

9. Click Calculate.

   If the function works correctly, TRUE appears in the Result box.

For more information about testing functions, see "Testing Expressions and Scripts" (p. 341).

External User-Defined Functions

External user-defined functions are defined within run-time libraries, each of which must adhere to special rules and conventions. When you have created an external user-defined function, you must register it within IBM® Cognos® Data Manager before you can use it.

External user-defined functions must adhere to these rules and conventions:

- Libraries of external user-defined functions must use C calling conventions. On the Microsoft® Windows® operating system, you must use dynamic link libraries (DLLs); and on the UNIX® operating system, shared libraries.
- You must store the results of execution in a formal parameter. This is because Data Manager ignores any value that the external user-defined functions return.
All formal parameters except the second must be of type DS_VALUE_T, a structured data type that supports values of several standard C data types. The first parameter contains the results of executing the function. All other parameters except the second supply input values to the function.

The second formal parameter must be of type DS_BOOLEAN. It specifies in which mode to execute the function.

For information, see "Special Data Types for External User-Defined Functions" (p. 309).

Data Manager can execute functions in either validation or normal execution mode, and any functions that you write must support this.

For information, see "Execution Modes for External User-Defined Functions" (p. 314).

An external user-defined function must be registered within Data Manager before you can use it.

For information, see "Register External User-Defined Functions" (p. 315).

To help you create external user-defined functions, Data Manager provides the header file named dsudftypes.h. This file defines the special data types and macros that you need, and is located in the directory named <install directory>\webcontent\samples\DataManager\udfs.

### Declare an External User-Defined Function

The simplest way to declare an external user-defined function is

```c
void <funcname>(
    DS_VALUE_T *<result>,
    DS_BOOLEAN <mode_flag>{,
    DS_VALUE_T *<input_parameter>} )
```

However, the compiler that you use and the operating system on which you want to use the function dictate how you must write function declarations. For example, using Microsoft® C++ version 6, you may use

```c
void__declspec (dllexport) __cdecl <funcname>(
    DS_VALUE_T *<result>,
    DS_BOOLEAN <mode_flag>{,
    DS_VALUE_T *<input_parameter>} )
```

**Note:** For convenience, DS_UDF_CALL has been defined to _declspec(dllexport) _cdecl on Microsoft® Windows® operating system, so functions can be defined as follows:

```c
void DS_UDF_CALL <funcname>
```

The parts of the declaration have these meanings:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;funcname&gt;</td>
<td>The name of the function.</td>
</tr>
</tbody>
</table>
The name of the parameter that stores the result of executing the function.

The name of the parameter that specifies the mode in which to execute the function.

The name of a parameter that provides an input value to the function.

Note: All parameters except the second are passed by reference.

Special Data Types for External User-Defined Functions

The header file named dsudftypes.h defines the following:

**DS_BOOLEAN**

A Boolean data type that can take either the value TRUE or the value FALSE. This data type is used for the `<mode_flag>`.

**DS_VALUE_T**

DS_VALUE_T is a structured data type that provides support for several standard C data types. This data type is used for results and parameters.

```
typedef struct {
    DS_DATATYPE_T t;
    DS_DATA_T v;
} DS_VALUE_T
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Identifies the IBM® Cognos® Data Manager data type.</td>
</tr>
<tr>
<td>v</td>
<td>Stores the value.</td>
</tr>
</tbody>
</table>

**DS_DATATYPE_T**

```
typedef struct {
    DS_DATATYPE_E eDataType;
    size_t Precision;
    int Scale;
} DS_DATATYPE_T;
```
An enumeration of the data type.

For numeric values, this part identifies the maximum number of digits that the value can hold.

For string values, this part identifies the maximum number of characters that the string can hold.

For numeric values, this part identifies the number of digits to the right of the decimal point.

For string values, no meaning.

**DS_DATATYPE_E**

The value of eDataType denotes the data type that a variable holds. The following are the values that variables of type DS_DATATYPE_E can take, together with the IBM® Cognos® Data Manager data type that each value denotes. For more information, see DS_DATA_U.

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_UNKNOWN_VTYPE</td>
<td>The data type is undefined.</td>
</tr>
<tr>
<td>DS_CHAR_VTYPE</td>
<td>The value is a string of UTF-8 characters.</td>
</tr>
<tr>
<td>DS_INTEGER_VTYPE</td>
<td>The value is a 64-bit integer.</td>
</tr>
<tr>
<td>DS_DOUBLE_VTYPE</td>
<td>The value is a double precision, floating point number.</td>
</tr>
<tr>
<td>DS_DATE_VTYPE</td>
<td>The value is a string representation of a date.</td>
</tr>
<tr>
<td>DS_BOOLEAN_VTYPE</td>
<td>The variable holds a Boolean value.</td>
</tr>
<tr>
<td>DS_BINARY_VTYPE</td>
<td>The variable holds binary data.</td>
</tr>
<tr>
<td>DS_ARRAY_VTYPE</td>
<td>The value contains a pointer to an array of values of type DS_VALUE_T.</td>
</tr>
</tbody>
</table>

**DS_DATA_T**

typedef struct
{
    DS_DATA_U u;
    DS_BOOLEAN bNull;
    DS_BOOLEAN bAlloc;
} DS_DATA_T;

310 IBM Cognos Data Manager
### Part

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>A union of C and IBM® Cognos® Data Manager data types.</td>
</tr>
<tr>
<td>bNull</td>
<td>A TRUE value indicates that the variable holds a null value, overriding any value that the union may hold.</td>
</tr>
<tr>
<td>bAlloc</td>
<td>Set this to TRUE to indicate that you have dynamically allocated memory to store the value, and that Data Manager should recover this memory during cleanup. For more information, see &quot;Allocating and Freeing Memory&quot; (p. 313).</td>
</tr>
</tbody>
</table>

#### DS_DATA_U

typedef union
{
    char *s;
    DS_INTEGER i;
    double d;
    DS_BOOLEAN b;
    char dt[80];
    DS_BINARY_T bi;
    DS_VALUE_T *p;
} DS_DATA_U;

#### DS_BINARY_T

typedef struct
{  
    size_t nBytes;
    DS_UBYTE *p;
} DS_BINARY_T;

### Represent Date Values

You must represent date values as null-terminated strings in this format:
syyyy-mm-dd hh:mi:ss
Meaning

- The sign of the year value (+ or -) (optional)
- The year as a four digit number
- The month as a two digit number
- The day of the month as a two digit number
- The hour in two digit, twenty-four hour format
- The minutes as a two digit number
- The seconds as a two digit number

Note: The time component, that is hh:mi:ss, is optional.

Data Type Mapping and Conversions

IBM® Cognos® Data Manager provides type conversion functions that can assist when creating and using user-defined functions, for example, ToDouble, ToInteger, and ToChar. In addition, Data Manager provides the TypeInfo function that is useful because NUMBER (10,2) does not necessarily mean that Data Manager will use a precise number to hold the values. Where possible, Data Manager substitutes a type such as Integer or Float. The TypeInfo function tells you the type actually in use.

This table describes the mapping between the parts of this union and the Data Manager native data types.

<table>
<thead>
<tr>
<th>Data Manager data type</th>
<th>Converted to eDataType</th>
<th>DS_DATA_U</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>DS_CHAR_VTYPE</td>
<td>u.s</td>
</tr>
<tr>
<td>INTEGER</td>
<td>DS_INTEGER_VTYPE</td>
<td>u.i</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DS_DOUBLE_VTYPE</td>
<td>u.d</td>
</tr>
<tr>
<td>NUMBER</td>
<td>DS_INTEGER_VTYPE or DS_DOUBLE_VTYPE</td>
<td>INTEGER if precision is &lt;19 and scale=0, otherwise, other precision/scale combination</td>
</tr>
<tr>
<td>DATE</td>
<td>DS_DATE_VTYPE</td>
<td>u.dt</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DS_BOOLEAN_VTYPE</td>
<td>u.b</td>
</tr>
<tr>
<td>BINARY</td>
<td>DS_BINARY_VTYPE</td>
<td>u.bi</td>
</tr>
</tbody>
</table>
IBM® Cognos® Data Manager handles arrays differently to other data types. The argument passed to the function is of type DS_VALUE_T with t.eDataType set to DS_ARRAY_VTYPE. The part v.u.p is a pointer to an array of values of type DS_VALUE_T, the first of which (with index value zero) denotes the number of items in the array. Subsequent elements contain the items that the array stores.

Your function should access the array by explicitly casting it to type DS_VALUE_T. For example

```c
DS_VALUE_T *pArgument; /* This is the passed argument */
DS_VALUE_T *pArray; /* This will point to the actual array*/
DS_VALUE_T *pElement; /* This will point to each element */
if (pArray = (DS_VALUE_T *)pArgument->v.u.p)
{
    for (i = 1; i<=pArray->v.u.i; i++)
    {
        pElement = &(pArray[i]);
        /* etc. */
    }
}
```

### Allocating and Freeing Memory

For certain data types it may be necessary for an external user-defined function to allocate memory for the result.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Union</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS_CHAR_VTYPE</td>
<td>v:u.s</td>
</tr>
<tr>
<td>DS_BINARY_VTYPE</td>
<td>v.u.bi</td>
</tr>
<tr>
<td>DS_ARRAY_VTYPE</td>
<td>v.u.p and individual array items</td>
</tr>
</tbody>
</table>

To ensure that IBM® Cognos® Data Manager can successfully free the memory, the function udffree should be present. For example

```c
void DS_UDF_CALL udffree (void *p)
{if (p) free (p);}
```
Execution Modes for External User-Defined Functions

IBM® Cognos® Data Manager can call external user-defined functions in one of two execution modes. These are

- validation execution mode, in which Data Manager calls the function at the start of build execution to check data types and to gauge memory requirements
- normal execution mode, in which Data Manager calls the function to perform the required calculation

Data Manager calls functions in validation execution mode at the start of build execution and in normal execution mode thereafter.

The second actual parameter specifies which execution mode is intended. If this parameter is FALSE, then the function must execute in validation execution mode; otherwise, it must execute in normal execution mode.

Validation Execution Mode (<mode_flag> = FALSE)

IBM® Cognos® Data Manager uses Validation mode to perform data type checking and to gauge memory requirements. When Data Manager calls a function in validation mode, for each input parameter of type DS_CHAR_VTYPE or DS_BINARY_VTYPE, t.Precision contains the maximum length of the data, or zero if this length is unknown. The function must, within the first parameter, return at least the following results:

- The t.eDataType member must indicate the expected type of the returned data.
- For variable length data only (for DS_CHAR_VTYPE and DS_BINARY_VTYPE), the function should set the t.precision member to the length of the data that the function would return if it actually performed the calculation. Where it is impossible to determine this value, the function should set t.Precision to zero.

Normal Execution Mode (<mode_flag> = TRUE)

IBM® Cognos® Data Manager calls functions in normal execution mode to perform the calculation that the function provides. It expects the function to exit with the first parameter set as follows:

- The eValueType component indicates the type of data that the union, u, holds.
- If the result is null, the v.bNull member must be TRUE; otherwise, this member must be FALSE.
- If the function dynamically allocated memory, the v.bAlloc member must be TRUE; otherwise, this member must be FALSE. For more information, see "Allocating and Freeing Memory" (p. 313).
- If the function returns a variable length value (t.eValueType is DS_CHAR_VTYPE or DS_BINARY_VTYPE), the function must set t.Precision to the string length (excluding any null terminator).

 Macros in External User-Defined Functions

The header file named dsudftypes.h defines these macros for use with the DS_VALUE_T data type.
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETURN_NULL(Result)</td>
<td>Sets Result to null and returns.</td>
</tr>
<tr>
<td>RETURN_NULL_WITH_TYPE(Result, Type)</td>
<td>Sets the type of Result to Type, sets the result value to null and returns.</td>
</tr>
<tr>
<td>RETURN_NULL_ON_NULL(Param, Result)</td>
<td>Sets the result value to null if the indicated parameter (Param) is null.</td>
</tr>
</tbody>
</table>

**Example External User-Defined Functions**

To see example external user-defined functions, see the file named sampleUDF.c which is located in the directory named <install directory>/webcontent/samples/ DataManager/udfs.

**Register External User-Defined Functions**

When you create external user-defined functions, you must register them before using them within IBM® Cognos® Data Manager.

**Steps**

1. In the Library folder, click Functions.
2. From the Insert menu, click Library, and then click Function.
3. Click the General tab.
4. In the Name box, type a unique name for the function.
   **Tip:** To ensure that this name is different from the built-in Data Manager functions, it is recommended that you precede the name of each user-defined function with the letters udf, for example, udf.newfunction
5. If you require, type a business name and description.
6. In the Return type box, click the data type to be returned by the function. It is this data type that is returned in the pResult parameter.
7. Click the Interface tab.
8. For each argument that you want to add, click Add.
   **Note:** Each user-defined function can contain a maximum of 16 arguments.
   By default, Data Manager names each argument, Argumentn (where n is a unique sequential number), and allocates an argument type of char.
9. Change each argument name to something more meaningful, because this is the name that you will use when you implement the function.
   The type and order of the arguments must match the type and order of the actual arguments. The names are unimportant.
Tip: To change the order in which you are prompted for argument values, click Move Up or Move Down.

10. For each argument, click within the Argument type column, then click on the arrow that appears to show the available data types. Click the required data type.

For information about supported argument types, see "Data Types" (p. 543).

11. Click the Implementation tab.

12. Click External.

13. In the Library name box, type the name of the library file which contains the function you are registering. On the Microsoft® Windows® operating system this would be a .dll file, on the UNIX® operating system, an .so or .sl file.

You do not have to type the full directory path for the file, the standard rules for locating dynamic libraries for your platform apply.

14. In the Function name box, type the actual name of the function. This may be the decorated name.

For more information, see "Cross-Platform Portability" (p. 316).

15. Click OK.

Cross-Platform Portability

Some compilers decorate function names (for example, with prefixes or suffixes) on export. This can lead to portability issues between compilers and between operating systems.

Depending on the compiler, you may be able to work around these issues by disabling name decoration or by creating aliases between the external and internal names. For example, Microsoft® C++ version 6 supports module definition files (*.def) that can include such aliases.

For more information about decorations or aliases, see the documentation for your compiler.

Maintain User-Defined Functions

You can move a user-defined function between catalogs by creating a package and importing it into the target catalog. For information, see "Component Packages" (p. 354).

Use a User-Defined Function

After you create a user-defined function and, in the case of an external user-defined function, register it, you can use it within derivations, derived dimensions, output filters, DataStream filters, and JobStream procedure and condition nodes.

Steps

1. Open the relevant Properties window.
2. For a derivation or derived dimension click the Calculation tab, for an output filter click the Output Filter tab, for a DataStream filter click the Filter tab, and for a JobStream procedure or condition node click the Action tab.

3. In the left pane, expand Functions, then User Defined, and then double-click the user-defined function to use.

   For more information, see "Add a Calculation to a Derivation or Derived Dimension" (p. 178), "Use an Output Filter to Deliver Specific Data Rows" (p. 220), "Filter Source Data" (p. 159), or "Add a Procedure or Condition Node to a JobStream" (p. 261).

**Delete a User-Defined Function**

You can delete a user-defined function that is no longer required.

**Steps**

1. In the Library folder, click the required function.
2. From the Edit menu, click Delete.
3. In the message box, click Yes.

**Establish the Dependencies of a User-Defined Function**

If a fact build contains a derivation or derived dimension that references a user-defined function, the fact build is dependent on that function. This dependency extends to the following objects in the catalog:

- packages
  If you add a fact build to a package, referenced functions must also be included.

- source code control
  If you check out a fact build, referenced functions must also be checked out.

- object deletion
  You cannot delete a function that is referenced by a fact build.

- the build visualization

To establish dependencies between the components and a function, you must be able to parse the scripts that reference the function. If the script is successful, dependency analysis is established.

If parsing a script fails, dependency analysis is not established. For example, if a function requires one parameter, but appears in the script with zero or two parameters, the parse fails and dependency analysis is not established.

To ensure that the dependency information is reported correctly, and that dependency analysis is established, test each script that you write.

For more information, see "Testing Expressions and Scripts" (p. 341).
Chapter 23: User-Defined Functions
Chapter 24: Variables

There are two types of variable that you can use in IBM® Cognos® Data Manager:

- **user-defined variables** which store custom values
- **property variables** which are intrinsic to Data Manager and affect the operation of Data Manager programs

### Declare Variables

You can declare variables within

- a build specification (build variables)
- a data source (data source variables)
- a JobStream (JobStream variables)
- a user-defined function (user-defined function variables)
- the environment of the operating system or in the file dm.ini (environment variables)
- in the configuration file dm.ini
- a command that invokes a Data Manager program (command line parameters)

The variable type determines the way in which you set it. For more information, see Declaration of Variables in Scripts in the IBM® Cognos® Function and Scripting Reference Guide.

You can declare variables that you later use within builds, JobStreams, derivations, derived dimensions, user-defined functions, and the Data Manager scripting language. You can declare these variables

- within the environment of the operating system
- within the definition of a build, JobStream, or user-defined function
- as command-line parameters

### Scope of Variables

The scope of a Data Manager variable is the collection of Data Manager objects for which that variable is visible. A variable becomes visible with its declaration and is visible for the object in which it is declared, together with other objects contained within that object. For example, a variable that you declare in the environment of the operating system is visible for all executions of all Data Manager programs, but a variable that you declare as a command line parameter is visible only for the specific execution of the Data Manager program for which you declared it.

A variable remains visible until one of these events occurs:

- The object that contains the declaration of the variable terminates.
Within the scope of the original variable, you declare another variable that has the same name. The original variable is displaced for the scope of the new variable.

Scoping can become quite complex where you have multiple objects with variables declared against them, because each object has its own scope. This allows recursive calls of Data Manager objects such as user-defined functions.

### User-Defined Variables

A user-defined variable is a name and value pair that resides in the memory of the computer. User-defined variables affect the operation of IBM® Cognos® Data Manager programs, store values for use in builds and JobStreams, and control the flow of JobStreams.

### User-Defined Variable Names

The name of a variable must

- start with an alphabetic character
- contain only alphanumeric characters, and underscores

Names of variables are not case sensitive and you can use a mixture of upper-case and lower-case characters in the name of a variable.

### Notation of Variables and Substitution Variables

Variables may be referenced within builds and JobStreams using one of two notations.

The notation $variable means that the current value of variable is applied whenever $variable is encountered.

For example, a variable X may be defined and initialized to 0. In a build, a derivation may increment the value of X by 1 as each row is processed, that is, $X := $X + 1. Therefore, X represents the number of rows processed. However, the following expression would result in X always having a value of 1 after the first invocation because 0 would have been substituted for $X at the outset, $X := {$X} + 1

Note: The $variable syntax cannot be used within SQL SELECT statements.

The notation {$variable} means that the value of variable at the start of the process is used to perform a one-off replacement wherever {$variable} occurs. Even if the value of variable changes throughout the process, this is not reflected wherever {$variable} was used. This is known as a substitution variable.

Note: The {$variable} syntax cannot be used within user-defined functions.

### Order of Precedence

The order of precedence for user-defined variables differs from the order of precedence for property variables. If you declare variables with the same name in more than one place

- build and JobStream variables take precedence over command-line parameters and environment variables
• command-line parameters take precedence over environment variables

Because of this, you cannot override user-defined build variables on the command line.

Add a User-Defined Variable

When you add user-defined variables, you can use any of the built-in IBM® Cognos® Data Manager functions.

For more information about Data Manager functions, see the IBM® Cognos® Function and Scripting Reference Guide.

Steps

1. Click the Data Manager component to which you want to add a user-defined variable.

2. From the Edit menu, click Properties.

3. Click the Variables tab.

4. Click Add.

The Variable window appears.

5. In the Name box, type a name for the new variable.

6. In the Type box, click the expected data type for the value that the variable is to return. For more information, see "Data Types" (p. 543).

   If you select Char or Number as the return type, the Precision box becomes available.

7. In the Precision box, type the maximum number of characters or numbers permitted for the value.

   Note: The precision can be up to 77 for the Number data type, and is only limited by the limitations of the operating system for the Char data type.

   If you select Number as the return type, the Scale box becomes available.

8. In the Scale box, type the number of decimal places permitted for the value.

9. In the right pane of the Initial expression box, enter an initial value for the variable if required.

   For information, see "Initial Values for Variables" (p. 322)

   The initial value can be an expression or a literal value. If you are entering an expression, you can type the expression, and you can use the tree structure in the left pane to select the items from the following folders:

   • Operators to select operators

   • Functions to select pre-defined and user-defined functions

   • Control to select controls

   • Variables to select variables and substitution variables
Chapter 24: Variables

**Note:** You can only reference other variables that are defined within the scope of the Data Manager component.

**Tip:** If a variable references another variable in the expression, the referenced variable must be positioned before the variable in which it is used. Use **Move Up** or **Move Down** to reposition variables.

You can select an item from a folder by dragging it to the right pane or by double-clicking it. For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information on user-defined functions, see "User-Defined Functions" (p. 305).

10. If required, test the expression.
   For information, see "Testing Expressions and Scripts" (p. 341).

11. Click **OK**.
   The variable is added to the **Variables** tab in the **Properties** window.
   **Tip:** To show the full variable expression, select the **Show all expression text** check box.

**Initial Values for Variables**

When you create a user-defined variable, you can enter an initial value for the variable in the right pane of the Initial expression box.

The initial value can be a simple expression or a literal value. For example, a variable named Variable1 can have the following initial values:

- 50 (which is a literal value)
- $var2 + $var3 (which is an expression)

It cannot however, include expressions of this type, as the result must be assigned to Variable1:

- $var4 := 10;

You can type an expression, and you can use the tree structure in the left pane to select the items from the following folders:

- Operators to select operators
- Functions to select pre-defined and user-defined functions
- Control to select controls
- Variables to select variables and substitution variables

You can select an item from a folder by dragging it to the right pane or by double-clicking it. For information about operators, controls, and functions, see the IBM® Cognos® Function and Scripting Reference Guide. For information about user-defined functions, see "User-Defined Functions" (p. 305).

**Note:** You can only reference other variables that are defined within the scope of the IBM® Cognos® Data Manager component.
Reposition User-defined Variables

The order in which the variables are shown on the Variables tab is the order in which IBM® Cognos® Data Manager evaluates them.

If a variable references another variable in the expression, the referenced variable must be positioned before the variable in which it is used.

Steps
1. Click the Data Manager object that contains the user-defined variable.
2. From the Edit menu, click Properties.
3. Click the Variables tab.
4. Click the variable to reposition.
5. Click Move Up or Move Down as appropriate.

Edit or Delete a User-Defined Variable

You can edit an existing user-defined variable or delete it if it is no longer required.

Steps
1. Click the IBM® Cognos® Data Manager component that contains the user-defined variable.
2. From the Edit menu, click Properties.
3. Click the Variables tab.
4. Click the required variable.
5. Click Edit or Delete.
6. If you chose Edit, in the Variable window make the required changes.
   Note: If you rename a variable, you must update all references to that variable.

Property Variables

Property variables are intrinsic to, and affect the operation of IBM® Cognos® Data Manager. You can set most property variables within a build or JobStream or in the environment of the operating system.

Portability

Environment variables can aid build portability between computers and operating systems because they store properties specific to a particular computer.

Environment variables can also help optimize performance. For example, you can store performance-related parameters as environment variables. These settings reflect the optimum for executing the majority of builds on a particular computer. To obtain optimum performance on a different com-
puter, modify only the batch file or shell script that prepares the computer for use with Data Manager.

**Order of Precedence**

If you declare a property variable in more than one place, the scope rules provide an order of precedence to resolve any potential conflict:

- Command-line parameters override build variables and JobStream variables.
- Build and JobStream variables override environment variables.
- Environment variables override variables defined in the Microsoft® Windows® operating system registry.

The order of precedence feature provides a convenient method for controlling build execution. For example, within the environment of the operating system, you can store the name of a text file that defines your default database aliases. You can use a command-line variable to override this setting, thus executing a particular build with alternative databases.

**Property Variable Categories**

For convenience, property variables are placed in the following categories:

- system variables
- run-time variables
- debug-related variables
- audit-related variables
- performance-related variables
- DBMS-related variables

**System Variables**

These variables affect other system issues in IBM® Cognos® Data Manager.

**COG_ROOT**

This environment variable is specific to the UNIX® operating system. It specifies the directory in which IBM® Cognos® Data Manager is installed.

The following example uses Bourne shell syntax to specify that Data Manager resides in /usr/cognos/c10

```
COG_ROOT="/usr/cognos/c10"; export $COG_ROOT
```

You can declare this variable in the environment of the operating system.

**DM_DATASOURCE_CONNECTION_SIGNONS**

When using IBM® Cognos® Connection with multiple data source connections, this variable specifies the data source or data sources to use.
The following example specifies, using a comma delimited list, that the data source details are /data source name/connection/signon name

```
DM_DATASOURCE_CONNECTION_SIGNONS=/gosales/oracle/signon1, gosales/oracle/signon2, gosales/sybase/signon1
```

You can declare this variable in the environment of the operating system or in the build specification.

**DM_DISTRIBUTED_HANDLING**

This variable enables distributed handling of unmatched members in fact builds. Defining this variable as an empty string or NULL uses the following default settings:

- Allow more than one fact build to access the lock at a time (non-exclusive locking).
- Use two separate SELECT statements. The first is used to determine the existence of an unmatched member. The second is used to determine the maximum surrogate. This option overrides the default.
- Use a lock only for those dimensions for which it has been set up.
- Do not use the lock name as a MUTEX/semaphore (node locking).

The following table shows optional values that you can use. These values can be combined.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Use exclusive locking. This option allows only one fact build to access the lock at a time. Within a JobStream, this forces fact build nodes to be executed in sequence. Note: For fact builds that contain a large number of unmatched members, this can improve performance considerably.</td>
</tr>
<tr>
<td>C</td>
<td>Use a combined SELECT statement. This option overrides the default use of two SELECT statements, which can improve performance. Tip: If you are using Oracle, it is recommended that you use this option.</td>
</tr>
<tr>
<td>D</td>
<td>Use a default lock. This option uses a generic lock, named &quot;dm_distributed_lock&quot; for all dimensions, overriding any locks previously set up.</td>
</tr>
<tr>
<td>N</td>
<td>Use node locking. For simultaneous builds executed from a single computer, use the lock name as a MUTEX/semaphore. Note: This can improve performance considerably and removes the need to create database tables.</td>
</tr>
</tbody>
</table>
You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

For more information about distributed handling of unmatched members in fact builds, see "Distributed Handling of Unmatched Members" (p. 181).

**DM_EMAIL_VIA_SERVICE**

This variable specifies that when you execute a JobStream containing an email node, the email is sent using the IBM® Cognos® Business Intelligence delivery service rather than MAPI. The value of this variable should be set to 1.

You can declare this variable in the environment of the operating system or in the Variables section of the file dm.ini.

**DM_SERVER_CAPACITY**

This variable indicates the processing capacity of the computer. The value of this variable is an arbitrary number assigned by you. When you set this variable to a value greater than zero, this enables load control for JobStreams. When set to zero, load control is disabled.

You can declare this variable in the file dm.ini.

For more information about load control in JobStreams, see "Specify the Load Required to Process a Fact Build or Dimension Build Node" (p. 269).

**DM_TM1_LOAD_SUPPRESS_ERROR_CODES**

(Default value: 136)

This variable specifies the error codes to ignore when delivering data to TM1®. You specify error codes using a comma delimited list.

**Note:** If you decide to ignore minor errors, information about the number of rows successfully delivered may not be accurate.

You can declare this variable in the environment of the operating system, in the file dm.ini.

**DS_CHAR_DEFAULT_PRECISION**

( Default value: 255)

When IBM® Cognos® Data Manager creates delivery tables and the data type of a value, or the precision of a character value is not known, Data Manager uses a default precision of 255. This variable overrides the default, up to a limit of 8000. It must be less than or equal to the maximum CHARACTER precision of all target delivery DBMS.

You can declare this variable in the environment of the operating system, in the file named dm.ini, in a build or JobStream, or on the command line.

**DS_DATA_DIR**

This variable specifies the default directory into which text data files are delivered. If you do not declare this variable, the default data directory is as follows.
### Default data directory

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Default data directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft® Windows® operating system</td>
<td>The data subdirectory of the IBM® Cognos® Data Manager directory</td>
</tr>
<tr>
<td>Unix® operating system</td>
<td>The current directory</td>
</tr>
</tbody>
</table>

The following example uses a command-line parameter to direct text data files to directory `c:\temp\ds`

databuild ODBC DSN=TestCat example -VDS_DATA_DIR=c:\temp\ds

You can declare this variable in the environment of the operating system, or on the command line.

You can also set the default location for data files using the -V option with the databuild, dimbuild, or rundsjob command, or using IBM® Cognos® Configuration.

Using the databuild, dimbuild, or rundsjob command takes precedence over a location set using the DS_DATA_DIR variable or IBM® Cognos® Configuration, and using the DS_DATA_DIR variable takes precedence over IBM® Cognos® Configuration.

For information on using the databuild, dimbuild, or rundsjob commands see, "Commands" (p. 413).
For information on using IBM® Cognos® Configuration, see the IBM® Cognos® Configuration User Guide.

### DS_LOG_DIR

This variable specifies the default directory into which log files are delivered when you execute builds from a catalog. If you do not declare this variable, the default log file directory is as follows.

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Default data directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft® Windows® operating system</td>
<td>The log subdirectory of the datamanager directory</td>
</tr>
<tr>
<td>UNIX® operating system</td>
<td>The current directory</td>
</tr>
</tbody>
</table>

The following example uses Windows® syntax to create an environment variable that specifies `C:\Temp\Log` to be the default log directory.

```shell
SET DS_LOG_DIR="C:\Temp\Log"
```

You can declare this variable in the environment of the operating system, or on the command line.

You can also set the default location for log files using the -V option with the databuild, dimbuild, or rundsjob command, or using IBM® Cognos® Configuration.

Using the databuild, dimbuild, or rundsjob command takes precedence over a location set using the DS_LOG_DIR variable or IBM® Cognos® Configuration, and using the DS_LOG_DIR variable takes precedence over IBM® Cognos® Configuration.
For information on using the databuild, dimbuild or rundsjob commands, see "Commands" (p. 413). For information on using the IBM® Cognos® Configuration, see the IBM® Cognos® Configuration User Guide.

**DS_PIPE_CREATE_WAIT**

Default: 30

This variable specifies the maximum time (in seconds) that IBM® Cognos® Data Manager should wait for a pipe to be created. If you enter a negative value, Data Manager waits indefinitely.

You can declare this variable in the environment of the operating system.

**DS_TMP_DIR, TMPDIR, and TEMP**

These variables specify the directory in which IBM® Cognos® Data Manager should store any temporary files. If DS_TMP_DIR is not found, then TMPDIR is used on the UNIX® operating system, and TEMP is used on the Microsoft® Windows® operating system.

The following example uses Cshell syntax to specify /usr/tmp to be the directory where Data Manager stores temporary files.

```
setenv DS_TMP_DIR /usr/tmp
```

You can declare this variable in the environment of the operating system, or on the command line.

**DS_UDA_BULKWRITE_ROWS**

(Default value: 50)

Array operations improve performance by reducing the number of round-trip operations between client and server components.

This variable indicates the number of rows to use for array writes if they are supported by the underlying IBM® Cognos® database driver. If they are not supported, they have no effect. If the driver does support array operations, they can be disabled by setting the variable value to 1.

If supported and enabled, array writes apply to INSERTs performed by relational table fact deliveries.

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_UDA_FETCH_ROWS**

(Default value: 50)

Array operations improve performance by reducing the number of round-trip operations between client and server components.

This variable indicates the number of rows to use for array reads if they are supported by the underlying IBM® Cognos® database driver. If they are not supported, they have no effect. If the driver does support array operations, they can be disabled by setting the variable value to 1.

If supported and enabled, array reads apply to most queries performed by IBM® Cognos® Data Manager. For data source queries, you can override the variable default value by using the Fetch...
row count option in Data Manager Designer. For more information, see "Define Advanced Settings for a Data Source" (p. 144).

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**TM1_PATH**

This environment variable is specific to the UNIX® operating system. You use this variable to specify the location of the parent folder that contains the SSL certificate.

SSL certificates, distributed by the NGTM1API component, are located in the same folder as the libtm1api.so library within the c10_location\bin\ssl folder.

You can declare this variable in the environment of the operating system.

**Run-time Variables**

*Note:* You should not declare or assign values to any run-time variables.

**DS_AUDIT_ID**

When you execute a build or a JobStream, this variable is set to a unique number that identifies each type of component.

**DS_BUILD_FNAME**

When you execute a build, this variable is set to the name of the build.

**DS_BUILD_NAME**

When you execute a build, this variable is set to the name of the build. If there are non-alphanumeric characters contained in the build name, these are replaced with underscores (_). This allows the variable to be used in file names.

For example, if the build name is ab*cd#, the variable is set to ab_cd_

**DS_DATA_TIMESTAMP**

When you execute a build, this variable is set to the value specified in the Data timestamp box in the Fact Build Properties window or the Dimension Build Properties window.

For more information about timestamping data, see "Set the Date Source for Timestamping Data" (p. 137) or "Set the Date Source for Timestamping Data" (p. 249).

**DS_JOB_AUDIT_ID**

When you execute a JobStream, this variable is set to a unique number that identifies the JobStream nodes.

**DS_JOBSTREAM_FNAME**

When you execute a JobStream, this variable is set to the name of the JobStream.
Chapter 24: Variables

**DS_JOBSTREAM_NAME**

When you execute a JobStream, this variable is set to the name of the JobStream. If there are non-alphanumeric characters contained in the JobStream name, these are replaced with underscores (_). This allows the variable to be used in file names.

For example, if the build name is a*b*cd, the variable is set to a_b_cd

**DS_LOG_NAME**

When you execute a build or a JobStream, this variable is set to the name of the log file.

**DS_RUN_ID**

When you execute a build or a JobStream, this variable is set to a unique number that identifies the build or JobStream.

**DS_RUN_TIMESTAMP**

This variable is set to the current date and time of the system on which you are running IBM® Cognos® Data Manager.

**Debug-Related Variables**

Use debug-related variables to save to the log file trace information that you can use when debugging a build.

**DS_REJECT_NULL_VALUE**

This variable specifies the value to use to represent NULL when writing to the reject file. Defining this variable as an empty string or NULL, results in an empty string representing NULL.

By default, <NULL> is used for fact builds and (null) for dimension builds.

Note: This variable has no effect when rejecting delivery data to tables.

**MEMBER_FREQUENCY**

(Default value: 10000)

This variable determines the frequency of DETAIL messages for member domain processing.

Note: To use this variable, you must also include DETAIL within the TRACE_VALUES specification.

This example uses Bourne shell syntax so that IBM® Cognos® Data Manager writes a DETAIL trace message at every 5,000th domain member it processes.

```bash
MEMBER_FREQUENCY=5000; export $MEMBER_FREQUENCY
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**OTRACE_FREQUENCY**

(Default value: 50000)

This variable specifies the number of output rows to process between DETAIL messages.

Note: To use this variable, you must also include DETAIL within the TRACE_VALUES specification.
This example creates a build variable so that a DETAIL trace message is written every 100,000th output data rows.

```sql
DECLARE OTRACE_FREQUENCY INTEGER 100000
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**REJECT_FILE**

This variable specifies the log file to which data row rejections are logged.

In the following example, all data row rejects are logged to the text file C:\temp\reject.log.

```sql
DECLARE REJECT_FILE CHAR(20) 'C:\temp\reject.log'
```

You can declare this variable in a build or JobStream, or on the command line.

If you are using IBM® Cognos® Data Manager with multilingual or unicode data, you should specify unicode encoding for your reject files. By default, the encoding used is platform dependent. For information, see "Specify the Encoding to Use for Reject Files" (p. 374).

**TRACE_FREQUENCY**

(Default value: 5000)

This variable specifies the number of input rows to be processed between DETAIL and INTERNAL trace messages.

**Note:** To use this variable, you must include DETAIL within the TRACE_VALUES specification.

This example uses Korn shell syntax so that IBM® Cognos® Data Manager writes a trace message every 10,000th input data row.

```sql
TRACE_FREQUENCY=100000; export $TRACE_FREQUENCY
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**TRACE_VALUES**

(Default value: PROGRESS)

This variable determines the events that IBM® Cognos® Data Manager traces during execution of a build. The following table shows the trace values that you can use.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRESS</td>
<td>Shows the overall progress of the application.</td>
</tr>
<tr>
<td>DETAIL</td>
<td>Shows more detailed progress messages.</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Shows internal Data Manager activity messages, including resource usage. For example, memory usage, dimension sizes, and paging information.</td>
</tr>
<tr>
<td>SQL</td>
<td>Shows all the SQL statements used at each stage of execution.</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EXECUTED SQL</td>
<td>Shows the executed SQL for SELECT statements.</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>Shows all the variables used in the build.</td>
</tr>
<tr>
<td>USER</td>
<td>Shows all the application messages written to the log file. You can</td>
</tr>
<tr>
<td></td>
<td>include additional application messages using the LogMsg function.</td>
</tr>
</tbody>
</table>

**Tip:** You can specify combinations of values by separating the values with commas.

The following example declares TRACE_VALUES as a build variable and sets it to include general progress and SQL information within the log file.

```sql
DECLARE TRACE_VALUES CHAR(12) 'PROGRESS,SQL'
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

### Audit-related Variables

Audit-related variables determine the auditing information written to the current catalog when a build is executed. Auditing is only available when you execute builds that reside in IBM® Cognos® Data Manager catalogs.

**AUDIT_VALUES**

Default: no auditing

This variable specifies the type of audit information that IBM® Cognos® Data Manager writes to the current catalog. The following table shows the audit values that you can use.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMING</td>
<td>Records overall timing information. For example, the start, end, and elapsed times for the execution process.</td>
</tr>
<tr>
<td>ACQUIRE</td>
<td>Records information about the source data acquired. For example, the number of data rows that each data source contributes. This information is not available for JobStreams.</td>
</tr>
<tr>
<td>TRANSFORM</td>
<td>Records information about the data processed. For example, the number of data rows received from the data sources. This information is not available for dimension builds or JobStreams.</td>
</tr>
</tbody>
</table>
### Performance-related Variables

These variables relate to the way that IBM® Cognos® Data Manager uses computer memory.

#### DS_BREAK_DIMENSION

This variable is intrinsic to all IBM® Cognos® Data Manager builds. It is available for each data row that the engine processes. If a dimension break occurs as Data Manager processes the row, then this variable contains the name of the breaking dimension. If a break simultaneously occurs in more than one dimension, then this variable contains the name of the most significant. If no dimension break occurs when processing the data row, this variable contains the null value.

For information about dimension breaking, see "Dimension Breaks" (p. 378).

**Note:** This variable is set at run-time, so you should not declare or assign values to it.

The following expression is a calculation for a derivation transformation model element:

```sql
IfNull(ToChar($DS_BREAK_DIMENSION), '<NULL>')
```

If the current data row causes a dimension break, this expression returns the name of the breaking dimension. If not, this expression returns the literal value `'<NULL>'`.

#### DS_LOB_BUF_SIZE

Default buffer size: 8000 bytes

The size of the buffer (in bytes) used for BLOB and CLOB data transfer. This can be used to fine tune BLOB and CLOB performance.

**Note:** CLOB data is held in UTF-16 encoding so a buffer size of 8000 is equivalent to 4000 characters.
**DS_LOB_MAXMEM_SIZE**

Default: 65536 bytes

If a BLOB or CLOB is less than the buffer size then it will be processed in memory otherwise it will be written to a temporary file. This applies to each LOB in a row, so if DS_LOB_BUF_SIZE is 8KB and there are four LOBs in a row then 32KB of memory are allocated.

**HASH_TABLE_SIZE**

Default: 200,000 slots

The hash table is a structure in computer memory that IBM Cognos Data Manager uses to process and transform rows of data. During build execution, Data Manager calculates the maximum number of hash table slots and uses this value if it is smaller than the default or user-specified value. For any build, you can determine the minimum hash table size by including INTERNAL in the TRACE_VALUES specification. Execute the build in check only mode and then search towards the end of the log for an entry like this where <nnnn> gives the minimum hash table size in slots:

```
[INTERNAL - hh:mm:ss] Hash: minimum table size <nnnn>
```

The following example declares HASH_TABLE_SIZE as a build variable that sets the minimum hash table size to 64,000 slots.

```
DECLARE HASH_TABLE_SIZE INTEGER 64000
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**MEMORY_LIMIT**

Default: no limit

This variable places an upper limit (in megabytes) on the amount of memory that IBM Cognos Data Manager may request from the operating system.

Some operating systems terminate any application that requests more memory than the operating system can provide. Therefore, you may want to impose a memory limit to prevent this. You may also want to impose a memory limit to prevent Data Manager requesting memory that other running applications need.

The following example declares MEMORY_LIMIT as a build variable that prevents Data Manager from requesting more than 64MB of memory from the operating system.

```
DECLARE MEMORY_LIMIT INTEGER 64
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**PAGE_SIZE**

Default: 16384 bytes

When IBM Cognos Data Manager reaches the limit of available memory, it creates virtual memory by paging information to disk. This variable determines the maximum page size that Data Manager uses. The maximum page size determines how much paging occurs and how many data
rows Data Manager stores on each page. To conserve memory, Data Manager automatically adjusts the maximum page size downward to accommodate exactly a whole number of data rows.

The following example uses Cshell syntax to declare PAGE_SIZE as an environment variable that sets the maximum page size to 32,768 bytes (32kb).

```
setenv PAGE_SIZE 32768
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**PAGE_TABLE_SIZE**

Default: calculate at run-time

The page table is a table in computer memory that IBM® Cognos® Data Manager uses to track the location of all the pages of memory. This variable specifies the initial page table size in number of page entries. By default, Data Manager determines the initial page table size from the size of the hash table.

The following example uses Microsoft® Windows® operating system syntax to declare PAGE_TABLE_SIZE as an environment variable that sets the initial page table size to 128 page entries.

```
SET PAGE_TABLE_SIZE=128
```

You can declare this variable in the environment of the operating system, in a build or JobStream, or on the command line.

**DBMS-related Variables**

These variables affect the specification of builds and reference structures.

**DS_DBALIAS_FILE**

Default: none

This variable affects only file-based projects. It points to a file that contains database alias definitions. Only one database alias definition file may be active at any one time, so this file must contain a definition for each alias that the current project requires.

For catalog-based projects, you must use the -A parameter to replace the connections that the catalog specifies with the aliases of an alias definition file. For information about using the -A parameter with each IBM® Cognos® Data Manager command, see the IBM® Cognos® Function and Scripting Reference Guide.

The following example uses Microsoft® Windows® operating system syntax so that the aliases of the file c:\cognos\projects\alias.txt are used by default.

```
SET DS_DBALIAS_FILE="c:\cognos\projects\alias.txt"
```

You can declare this variable in the environment of the operating system.

**DS_DB2_LOADER**

This variable specifies the command that invokes the DB2® bulk load utility. By default, IBM® Cognos® Data Manager uses db2cmd /c /w /i db2 under the Microsoft® Windows® operating system
and uses db2 under the UNIX® operating system. To specify another load command, either specify it in this variable or define the CMD property of each DBLOAD delivery. The CMD property takes precedence over the DS_DB2_LOADER variable.

The following example uses Bourne shell syntax to cause Data Manager to use db2 when delivering data using the DB2LOAD delivery module.

```bash
DS_DB2_LOADER='db2'; export DS_DB2_LOADER
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_DBMS_TRIM**

This variable causes IBM® Cognos® Data Manager to right trim white space when reading characters.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>Whitespace is not trimmed</td>
</tr>
<tr>
<td>SPACE</td>
<td>Only the space character is trimmed (default)</td>
</tr>
<tr>
<td>FULL</td>
<td>All whitespace is right trimmed</td>
</tr>
</tbody>
</table>

**DS_DBMS_TRUNCATION**

This variable determines the action IBM® Cognos® Data Manager takes when writing character data to a column that is too small to hold that data.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW</td>
<td>Allows Data Manager to truncate characters as required.</td>
</tr>
<tr>
<td>WARN</td>
<td>Allows Data Manager to truncate characters as required, but issues a warning to inform you.</td>
</tr>
<tr>
<td></td>
<td>You can optionally add a maximum size of the value to include in the warning message:</td>
</tr>
<tr>
<td></td>
<td>- WARN issue a general warning when truncation is detected</td>
</tr>
<tr>
<td></td>
<td>- WARN=255 issue the same warning and include a separate message with up to 255 characters of the value being truncated</td>
</tr>
<tr>
<td>ERROR</td>
<td>Issue an error message and terminate.</td>
</tr>
<tr>
<td></td>
<td>You can optionally add a maximum size of the value to include in the error message:</td>
</tr>
<tr>
<td></td>
<td>- ERROR=255 issue a general error when truncation is detected (default)</td>
</tr>
</tbody>
</table>
**DS_INFORMIX_LOADER**

This variable specifies the command that invokes the Informix bulk load utility. By default, IBM® Cognos® Data Manager uses dbaccess. To specify another bulk load command, either specify it in this variable or define the CMD property of each INXLOAD delivery. The CMD property takes precedence over the DS_INFORMIX_LOADER variable.

The following example uses Microsoft® Windows® operating system syntax to cause Data Manager to use dbaccess when performing bulk loads to Informix databases.

```
SET DS_INFORMIX_LOADER="dbaccess"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_MAX_RECURSION**

This variable specifies the maximum recursive depth of user-defined functions which IBM® Cognos® Data Manager will allow when executing a script. For example, if DS_MAX_RECURSION = 5 then Function1 would succeed but Function2 would fail with error "Maximum recursion of functions has been exceeded". This is to prevent scripts from looping uncontrollably. The default value is 100.

```
Integer Function1(integer count)
If (count < 6) then
    Function1(count+1);
else
    return count;
```

```
Integer Function2(integer count)
If (count < 7) then
    Function1(count+1);
else
    return count;
```

**DS_MSSQLSERVER_BCP**

This variable specifies the command that invokes the Microsoft® SQL Server BCP bulk load utility. By default, IBM® Cognos® Data Manager uses bcp. To specify another load command, either specify it in this variable or define the CMD property of each Microsoft® SQL Server BCP delivery. The CMD property takes precedence over the DS_MSSQLSERVER_BCP variable.

The following example causes Data Manager to use fastload when delivering to Microsoft® SQL Server BCP databases.

```
SET DS_MSSQLSERVER_BCP="bcp"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_ORACLE_LOADER**

This variable specifies the command that invokes the Oracle bulk load utility. By default, IBM® Cognos® Data Manager uses sqlldr. For Data Manager to use the correct bulk loader, either specify
it in this variable or define the CMD property of each ORALOAD delivery. The CMD property takes precedence over the DS_ORACLE_LOADER variable.

For example, if you have both Oracle 9i and Oracle 10 client installed, and you want to be sure that the Oracle 9i version of sqldr is used, set this variable to the full path name, such as

```
SET DS_ORACLE_LOADER=c:\Oracle\Product\9.2\Client_1\bin\sqlldr.exe
```

**Note:** If no value is specified for this variable, and the CMD property of the ORALOAD delivery is not set, the PATH environment variable determines which version to use.

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

### DS_REDBRICK_LOADER

This variable specifies the command that invokes the RedBrick® load utility. By default, IBM® Cognos® Data Manager uses rb_tmu. To specify another load command, either specify it in this variable or define the CMD property of each RBLOAD delivery. The CMD property takes precedence over the DS_REDBRICK_LOADER variable.

The following example uses Microsoft® Windows® operating system syntax to cause Data Manager to use rb_tmu when delivering data to Redbrick databases.

```
SET DS_REDBRICK_LOADER="rb_tmu"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

### DS_RUNENV

**Default:** <DEFAULT>

This variable affects only file-based projects. It specifies the run environment to use when acquiring or delivering data. The active database alias definition file must define all the required aliases for this run environment.

To specify a run environment on the command line, use the -e command-line parameter. For more information, see the IBM® Cognos® Function and Scripting Reference Guide.

**Note:** If you are importing a database alias that is using a run environment other than <DEFAULT>, you must ensure that the DS_RUNENV variable is set to the required value. For more information, see "Database Drivers" (p. 515).

The following example assumes that by default, IBM® Cognos® Data Manager connects to the Oracle database.

```
<DEFAULT> Sales ORACLE user/password@service
DEVELOPMENT Sales ODBC 'DSN=Sales;UID=user;PWD=password'
```

However, you may not want Data Manager to do this while you are developing a build. If you are using the Korn shell you can specify that Data Manager should use the DEVELOPMENT run environment, and so use the ODBC development data source rather than the Oracle production data. You do this by typing

```
DS_RUNENV=DEVELOPMENT; export $DS_RUNENV
```

You can declare this variable in the environment of the operating system.
**DS_TERADATA_FASTLOAD**

This variable specifies the command that invokes the Teradata Fastload bulk load utility. By default, IBM® Cognos® Data Manager uses fastload. To specify another load command, either specify it in this variable or define the CMD property of each Teradata Fastload delivery. The CMD property takes precedence over the DS_TERADATA_FASTLOAD variable.

The following example causes Data Manager to use fastload when delivering to Teradata Fastload databases.

```
SET DS_TERADATA_FASTLOAD="fastload"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_TERADATA_MULTILOAD**

This variable specifies the command that invokes the Teradata Multiload bulk load utility. By default, IBM® Cognos® Data Manager uses mload. To specify another load command, either specify it in this variable or define the CMD property of each Teradata Multiload delivery. The CMD property takes precedence over the DS_TERADATA_MULTILOAD variable.

The following example causes Data Manager to use mload when delivering to Teradata Multiload databases.

```
SET DS_TERADATA_MULTILOAD="mload"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.

**DS_TERADATA_TPUMP**

This variable specifies the command that invokes the Teradata TPump bulk load utility. By default, IBM® Cognos® Data Manager uses tpump under the Microsoft® Windows® operating system and uses tmpump under the UNIX® operating system. To specify another load command, either specify it in this variable or define the CMD property of each Teradata TPump delivery. The CMD property takes precedence over the DS_TERADATA_TPUMP variable.

The following example uses Windows® syntax to cause Data Manager to use tpump when delivering to Teradata TPump databases.

```
SET DS_TERADATA_TPUMP="tpump"
```

You can declare this variable in the environment of the operating system, in the file dm.ini, in a build or JobStream, or on the command line.
Chapter 25: Testing Expressions and Scripts

It is important to be able to adequately test expressions and scripts to ensure they perform as expected during execution. To facilitate this, IBM® Cognos® Data Manager provides a testing utility. The objective of this is to simulate execution conditions as closely as possible, and to allow values to be manipulated so that expressions can be tested using a wide range of data values.

You can test expressions and scripts for

- derivation and derived dimension elements
- user-defined functions
- output filters
- JobStream procedure and condition nodes
- variable definitions

Note: You can test all internal user-defined functions in Data Manager Designer. You can also test external user-defined functions created under the Microsoft® Windows® operating system; those created under the UNIX® operating system cannot be tested from Data Manager Designer.

You test expressions and scripts in the Values for Expression window. The Input values and the Output values boxes have the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
</table>
| Status icon | ✓ indicates a definition for the item has been located. For information, see "Scope of Expressions and Scripts" (p. 342).  
|           | X indicates a definition for the item has not been located, either because it has not yet been defined, or because it is defined at a higher scope than is currently known to Data Manager. For information, see "Scope of Expressions and Scripts" (p. 342).  
|           | ⚠️ indicates a definition has been located, but an error exists within it. To view details of the error, double click the exclamation mark. For information, see "Default Values in Expressions and Scripts" (p. 345). |
| Name      | Lists all the items for which you must specify an initial value.           |
| Type      | If the definition can be located, shows the data type.                     |
| Value     | You enter a value as required.                                             |
The Output values box shows any items to which the expression or script assigns a value.

When you execute a build or JobStream you can use logging to debug expressions and scripts contained within a build or JobStream. For more information, see "Create a Log File for Expression and Script Tracing" (p. 292).

Test an Expression or Script

This section describes how to test an expression or a script.

Steps

1. In the appropriate window, click Test.
   
   Tip: When developing a complex script, you may want to test just a section of the expression or script. You can do this by selecting the relevant section, and then clicking Test.

2. If required, in the Top box, specify the top scope of the expression or script by clicking the Browse button, and selecting the appropriate object that contains the definitions.

   For information, see "Scope of Expressions and Scripts" (p. 342).

3. If required, in the Types column in the Input values box, specify the data type.

   For more information, see "Data Types in Expressions and Scripts" (p. 343).

4. In the Value column in the Input values box, type a value for each item.

   Tip: You can overtype any default values shown.

5. If you want IBM® Cognos® Data Manager to determine the data types and initial values from the definitions, click Set Default Values.

   For more information, see "Default Values in Expressions and Scripts" (p. 345).

6. Click Calculate.

   The result of the calculation is shown in the Result box.

   Note: If you are executing a script, the Result box shows the value from the RETURN statement that caused the script to terminate.

   The Output values box shows any items to which the expression or script assigned a value.
   The final value of each of these items is shown.

7. If the result is incorrect, make the necessary changes, and perform the test again.

8. Click Close.

Scope of Expressions and Scripts

By default, IBM® Cognos® Data Manager only searches for item definitions within the immediate scope of the expression or script. For example, if you are testing a derivation in a fact build, the immediate scope is the fact build in which the derivation is defined. If any fact build variables are used in the derivation, Data Manager finds them.
**Note:** The immediate scope includes any variables defined within the current expression or script as well as variables declared within the operating system environment level.

If a definition is located, a check mark appears next to the item name in the Input values box.

If a definition is not found, a cross appears to indicate that either

- the item has not yet been defined
- the item is defined at a higher scope that is currently unknown to Data Manager

If an item has been defined but its scope is unknown, for testing purposes, you can specify the *top scope* of the expression or script using the Top box.

Suppose you define a variable in a JobStream and reference that variable from an expression defined in a fact build. Because the fact build relies on the inheritance of the variable from the JobStream, the fact build fails if you try to execute it independently of the JobStream. In this instance, the JobStream is the top scope for the expression.

For more information, see "Scope of Variables" (p. 319).

When you specify the top scope, Data Manager attempts to resolve the path between the immediate and top scope so that it can locate the required item definitions.

**Note:** If Data Manager cannot resolve the path, only the immediate scope is used for testing.

---

**Data Types in Expressions and Scripts**

If an item definition is found within the immediate or top scope, its data type is determined from the definition. In this case, the data type appears in the Type column for the appropriate item.

For items that require an input value, if a definition is not found or no data type has been set in the definition, the data type is set to UNKNOWN and the data type is deduced from the test value you specify.

<table>
<thead>
<tr>
<th>Value format</th>
<th>Example test values</th>
<th>Data type deduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characters enclosed in single quotation marks</td>
<td>'10.0'</td>
<td>CHAR</td>
</tr>
<tr>
<td>Digits without a decimal point</td>
<td>100</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Digits with a decimal point</td>
<td>10.0</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Text that represents a date in yyyy-mm-dd hh:mm:ss format</td>
<td>2005-06-22 10:30:15</td>
<td>DATE</td>
</tr>
<tr>
<td>Text that represents a time in hh:mm:ss format</td>
<td>10:30:15.123</td>
<td>TIME</td>
</tr>
<tr>
<td>Text that represents a time in hh:mm:ss format</td>
<td>10:30:15</td>
<td>TIME</td>
</tr>
<tr>
<td>Value format</td>
<td>Example test values</td>
<td>Data type deduction</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Text that represents a date in yyyy-mm-dd hh:mm:ss.fffffffff stzh:tzm format</td>
<td>2005:06:22 10:30:15.123 -8.00</td>
<td>DATE WITH TIME ZONE</td>
</tr>
<tr>
<td>Text that represents a time in hh:mm:ss.fffffffff stzh:tzm format</td>
<td>10:30:15.123 -8.00</td>
<td>TIME WITH TIME ZONE</td>
</tr>
<tr>
<td>Text that represents an interval in sdddddddddd hh:mm:ss</td>
<td>-10 10:30:15.123</td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
<tr>
<td>Text that represents an interval in syyyyyyyy-mm format</td>
<td>1-11</td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td>TRUE or FALSE</td>
<td>TRUE</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>No value given</td>
<td></td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>Text that includes ampersands</td>
<td>'a' &amp; 'b' &amp; 'c' a &amp; b &amp; c</td>
<td>ARRAY</td>
</tr>
<tr>
<td>Any other format</td>
<td>abCdef</td>
<td>CHAR</td>
</tr>
</tbody>
</table>

**Notes**

- When entering a string literal NULL value, you must use single quotation marks, that is, 'NULL'.
- If the precision and scale is greater than the limits allowed for INTEGER or FLOAT, NUMBER is used.
- For quoted character values (for example, '100.12'), the initial character string does not include the quotation marks.
- The expression or script may behave unpredictably if the run-time value is of a different data type to that determined at testing.

If you do not want IBM® Cognos® Data Manager to deduce the data type, you can explicitly set a data type.

Any input values that you have specified, are changed using the new data type. For example, if you specified an input value of 12.345 before setting the data type, the data type is deduced to be FLOAT. If you then set the data type to INTEGER, the input value changes to 12.

If you enter an input value that is invalid for the specified data type, Data Manager changes the value to 0. For example, if you set the data type to DATE, entering ABCD is an invalid value.
Note: You can only set an explicit data type where the data type is initially UNKNOWN.

Steps to Set the Data Type For Items
1. For the appropriate item, click within the Type column, then click the Browse button that appears.
   The Data Type dialog box appears.
2. In the Data type box, click the required data type. For information, see "Data Types" (p. 543).
   If you select CHAR, NUMBER, or BINARY as the data type, the Precision box becomes available.
3. In the Precision box, type the maximum number of characters or numbers permitted for the value.
   Note: The precision can be up to 77 for the NUMBER data type, and for the CHAR data type is limited only by the operating system restrictions.
   If you select NUMBER as the data type, the Scale box becomes available.
4. In the Scale box, type the number of decimal places permitted for the value.
5. Click OK.

Default Values in Expressions and Scripts
An item definition can include an initial value. Where a definition can be located, the initial value is not automatically obtained because you may want to test your expression or script using your own values.

Note: IBM® Cognos® Data Manager searches for definitions within the immediate scope, the top scope, if it is specified, and any environment variables.

Step to Obtain Initial Values
- Click Set Default Values.
  Data Manager searches for definitions of each item listed in the Input values and Output values boxes. For each definition found, an initial value is entered in the Value column, if one has been set in the definition.

If an error exists in a definition, the initial value is not obtained and an exclamation mark appears next to the item name. Although you must resolve the error before execution, for the purposes of testing you can enter a value and continue testing the expression or script.

Note: If an initial value is defined as an expression, obtaining default values may take several minutes because of the processing required to evaluate the expression.
Chapter 26: Multi-Developer Support

Multi-developer support allows developers to share information in a catalog. There are various ways in which you can achieve multi-developer support in IBM® Cognos® Data Manager:

- source code control
- component packages
- backup and restore

The method best suited to you depends upon the number of developers involved in your project and the environment in which you are working.

**Source Code Control**

Storing an IBM® Cognos® Data Manager catalog in a source code control system allows multiple developers concurrent access. A master Data Manager catalog is added to the source code control system, and each developer then creates a personal catalog that is connected to the source code control repository as illustrated here.

![Source Code Control Diagram](image)

*Note:* To use source code control, you must have an external source code control system already set up and you must be connected to it.

Using source code control allows all connected users to view all the contents of the catalog, execute builds and JobStreams, and create components. To maintain existing catalog components, you must first check them out of the source code control system. For more information, see "Check Out Catalog Components" (p. 351).
Guidelines for Planning a Source Code Control Project

If you plan to have multiple developers working on a project, it is important to plan the project at the outset.

When planning a source code control project, you should consider the following:

- Each developer must be aware of the project areas on which they are to work.
- Common components that must be shared by all developers, such as connections and user-defined functions, should be implemented first.
- Once implemented, you can lock common components by checking them out so that no developer can change them.
- Dimensional framework components should be implemented next because reference structures are used in many places in a catalog.
- Complete the development of all fact builds and dimension builds before integrating them into JobStreams.

Enable Source Code Control

Before you can use source code control you must enable it within IBM® Cognos® Data Manager.

Steps

1. Close any open catalogs.
2. From the Tools menu, click Options.
3. Click the Source Code Control tab.
4. Select Enable source code control.
5. Click OK.

Add a Catalog to Source Code Control

You add a catalog to source code control to create a master copy of the catalog which other developers can share.

Note: If you have upgraded from IBM® Cognos® DecisionStream Series 7, and you were previously using source code control, you must add your upgraded catalogs to source code control. For more information, see "Post Upgrade Considerations" (p. 52).

Steps

1. Open the catalog that you want to add to source code control.
2. From the Tools menu, click Source Code Control, Add Catalog to Source Code Control. A login dialog box appears.
3. Log in to your source code control system.
The **Add to Project** dialog box appears.

4. Select the project to which you want to add the catalog.

   **Tip:** If no suitable project exists, create one by typing a project name in the **Project** box.

5. Click **OK**.

   When the catalog has been successfully added to source code control, a message appears informing you of this, and the following files are created:
   - a catalog project file
   - a file for each component in the IBM® Cognos® Data Manager catalog

   Data Manager sets the status of every catalog component to read-only and, in the **Tree** pane, additional icons appear next to each component to indicate their current status under source code control.

   For more information, see "Source Code Control Icons" (p. 557).

---

### Create a Shared Copy of a Catalog from Source Code Control

When a catalog has been added to source code control, a master copy of the catalog is created. Every developer who wants to share that catalog must create a copy of it on their computer from the source code control project.

**Notes**

- Multiple developers can all be connected to the same source code control system. However, it is important that they are not using the same connection to a catalog within IBM® Cognos® Data Manager, otherwise the source code control system may behave incorrectly. Each developer must set up their own personal catalog, and insert a shared copy of the master catalog into it. It is the source code control system that allows multi developer use, not the physical Data Manager catalog.

- When inserting a shared copy of a master catalog into a personal catalog, a developer must log in using their own ID, not a common ID or administrator ID, because user information is stored in the catalog.

**Steps**

1. In Data Manager, create an empty catalog or, if you want to overwrite an existing catalog, open the relevant catalog.

   For information, see "Create an IBM Cognos Data Manager Catalog" (p. 44).

2. From the **Tools** menu, click **Source Code Control, Create Catalog, Create Catalog from Source Code Control**.

   A warning appears informing you that the current catalog will be overwritten.

3. Click **Yes** to continue.

   A login dialog box appears.
4. Log in to your source code control system.
   The Choose Project dialog box appears.

5. Select the project containing the catalog to copy, and then click OK.
   Data Manager overwrites the current catalog with a copy of the catalog from source code control, and sets the status of every catalog component to read-only.

**View the Source Code Control Properties of a Component**

You can view the source code control properties of a specified component. This is useful for verifying status information such as

- the name of the developer who has checked out a component
- when the component was last updated
- the number of changes that have been made to a component

**Steps**

1. Click the component for which you want to check the source code control properties.
2. From the Tools menu, click Source Code Control, Source Code Control Properties.
   A dialog box appears, showing information about the selected component.

**Refresh the Status of Components**

The source code control icons shown in the Tree pane reflect the current status of components in the catalog. When you open a catalog that is connected to a source code control repository, IBM® Cognos® Data Manager automatically updates the icons so that you can see what is currently checked in and checked out. To update the icons you must manually refresh the status.

For more information, see "Source Code Control Icons" (p. 557).

**Note:** Data Manager does not synchronize the catalog when it refreshes the status. For information, see "Obtain the Latest Version of a Catalog" (p. 350).

**Step**

- From the Tools menu, click Source Code Control, Refresh Component Statuses.
  The Tree pane is updated to reflect the status of the master catalog.

**Obtain the Latest Version of a Catalog**

If another developer checks in changes to the master catalog stored in the source code control system, you must synchronize your local catalog to reflect the changes.

**Note:** When you check in or check out components, if necessary you are automatically prompted to synchronize your catalog.

Before synchronizing the catalog, IBM® Cognos® Data Manager compares each component in the catalog with the master catalog and shows a summary of the changes required. This can include...
Creating components in your local catalog that exist in the master catalog

Updating components in your local catalog

Deleting components from your local catalog that no longer exist

When you synchronize a catalog, Data Manager refreshes its status.

**Steps**

1. From the **Tools** menu, click **Source Code Control, Get Latest Version of Catalog**.
   
   The **Results** window appears with a summary of changes required to synchronize your catalog.

2. Click **OK** to continue.

**Check Out Catalog Components**

To amend components in a catalog that is connected to a source code control repository, you must first check out the components.

You can check out individual components, for example a specific dimension build, or you can check out an entire catalog. If you are checking out individual components, IBM® Cognos® Data Manager also forces you to check out any component dependents, such as templates used by a dimension build. However, if the selected component includes dependents that are checked out by another developer, you cannot check it out.

**Note:** When you check out a user-defined folder, only the folder is checked out, not its contents. You must manually check out any required components.

**Steps**

1. In the **Tree** pane, click the component to check out.
   
   **Note:** You cannot check out components that are already checked out by another developer.

2. From the **Tools** menu, click **Source Code Control, Check Component Out**.
   
   If another developer has made changes to the master catalog, a message appears informing you to synchronize the catalog first.
   
   For information, see "**Obtain the Latest Version of a Catalog**" (p. 350). Repeat steps 1 and 2 after synchronizing.
   
   If the selected component has dependents, a list of the dependent components appears. You must also check out these components. Click **OK** to confirm that you want to check them out.
   
   Data Manager checks out the components from source code control, changes their status to read and write, and updates the source code control icons in the **Tree** pane to indicate their current status.

3. Maintain the components as required.
   
   When you have finished maintaining the components, you must check the catalog into source code control. For information, see "**Check a Catalog into Source Code Control**" (p. 352).
Check a Catalog into Source Code Control

You must check your local catalog into your source code control system when

- you create components
- you have finished maintaining components that you previously checked out

The check-in process applies to an entire catalog.

IBM® Cognos® Data Manager compares each component in your local catalog with the master catalog and shows a summary of the changes required. This can include

- adding new components
- updating existing components
- deleting components that no longer exist in your catalog

Steps

1. From the Tools menu, click Source Code Control, Check Catalog In.
   
   If another developer has made changes to the master catalog, a message appears informing you to synchronize the catalog first. For information, see “Obtain the Latest Version of a Catalog” (p. 350).

2. After synchronizing, repeat step 1.
   
   The Check In Catalog dialog box appears.

3. In the Comment box, type a description of the catalog changes you are checking in, and then click OK.
   
   Data Manager updates the master catalog, resets the status of all catalog components to read-only, and refreshes the icons in the Tree pane accordingly.

Undo a Catalog Check Out

If you have checked out components, but not made any changes, or you want to abandon your changes, you can undo the check out.

The undo check-out process applies to an entire catalog.

Step

- From the Tools menu, click Source Code Control, Undo Catalog Check Out.

Open the Source Code Control System from IBM Cognos Data Manager

You can open the source code control system that you are using directly from IBM® Cognos® Data Manager.

Step

- From the Tools menu, click Source Code Control, Run Source Code Control Client.
**Detach a Catalog from Source Code Control**

You can detach your local catalog from the source code control system. This removes all references to the source code control system and provides full read and write access to the catalog components for single users. To make the catalog available to multiple users again, you must recreate a local catalog from the source code control system.

**Steps**

1. From the **Tools** menu, click **Source Code Control, Remove Source Code Control Information**.
   If there are components currently checked out, a message appears recommending that you either check in the catalog or undo the component check out before you detach the catalog from source code control.

2. If you do not want to proceed with detaching the catalog, click **No** to cancel. If you do want to proceed with detaching the catalog, click **Yes**.
   IBM® Cognos® Data Manager removes the source code control icons from the **Tree** pane.

**Create a Standalone Copy of a Catalog from Source Code Control**

You can create a standalone copy of the master catalog directly from the source code control project files. The catalog you create is not linked to source code control. It is a local catalog that is stored only on your computer. You have full read and write access to the catalog components.

You may want to use this, for example, to roll back to a previous version of the master catalog or to restore the catalog from a specific date.

**Note:** To roll back to a specific version or date, the catalog must be labelled in the source code control system.

**Steps**

1. In the source code control system, use the Get Latest Version command to copy the project files for the master catalog onto your hard drive.

2. In IBM® Cognos® Data Manager, create an empty catalog or, if you want to overwrite an existing catalog, open the relevant catalog.

3. From the **Tools** menu, click **Source Code Control, Create Catalog, Create Catalog from Project File**.
   A warning appears informing you that the current catalog will be overwritten.

4. Click **Yes** to continue.
   The **Browse for Folder** dialog box appears.

5. Select the folder containing the catalog project files, then click **OK**.
   Data Manager creates a standalone copy of the catalog, which you can now maintain.
Temporary Files Used By Source Code Control

IBM® Cognos® Data Manager temporarily stores files on your hard drive whenever you work with source code control. By default, temporary files are stored in the directory named Documents and Settings\<username>\Local Settings\Temp\scs. You can change this to a directory of your choice. Temporary files are automatically deleted by Data Manager when the required task is complete.

Steps to Change the Default Directory for Temporary Files

1. From the Tools menu, click Options.
2. Click the Source Code Control tab.
3. In the Default directory for SCCS files box, enter the directory where you want the temporary files to be stored.
4. Click OK.

Component Packages

You can use component packages to
● move components between catalogs
● combine the efforts of several developers
● distribute catalogs or components
● partially backup and restore a catalog

Note: If you want to backup and restore a complete catalog, see "Backup and Restore Catalogs" (p. 359).

Component packages are useful if you work in an environment where concurrent access to catalogs is not feasible.

When you use component packages to copy components between catalogs, the components are saved in a non-editable .pkg file. If you want to copy a catalog component to an editable text file, you should use the CATEXP command to export the catalog, followed by the CATIMP command to import the component into the target catalog. For more information about using these commands, see "Commands" (p. 413).

Create a Component Package

A component package can contain as many catalog components as you require. Catalog components are connections, builds, reference dimensions, reference structures, JobStreams, templates, functions, metadata dimensions, and metadata collections.

Note: Connections are not always portable between computers, so you must assess the connection before you attempt to move it.

When you create a component package, IBM® Cognos® Data Manager lists all the dependent components in the catalog which must also be included in the package. However, when you import
the component package into the target catalog, you can choose whether to include these dependent components, or use existing components in the target catalog.

**Steps**

1. From the File menu, click Create Package.

2. In the Available components pane, select the check box for the components to add to the package.
   
   The selected components and all their dependent components are shown in the Components in package pane.
   
   - When you add a user-defined folder to a package, Data Manager automatically adds all the builds and JobStreams contained in that folder. If you do not want to import the whole folder, clear the check boxes for the unwanted builds or JobStreams.
   
   - If you add a user-defined folder to a package, only the contents of the folder are shown in the Components in package pane. Any builds and JobStreams contained in a user-defined folder are shown as a path string showing the folder name followed by the build or JobStream name. For example, if user-defined Folder1 contains JobStreamA, it appears in the Components in package pane as Folder1\JobStreamA.

   **Tip:** To remove a component from the Components in package pane, clear its check box. Data Manager also removes any dependents of that component.

3. In the Description box, enter a description of the package so that when the package is imported, it can be easily identified.

4. Click OK.
   
   The Package File dialog box appears.

5. In the File name box, type a name for the package file.

6. In the Save as type box, click Package Files (*.pkg).

7. Click Save.

**Import a Component Package**

You can import components from a component package (*.pkg) file to partially restore a catalog or integrate work into a test or production catalog. You can also import components from a catalog backup (*.ctg) file using this method.

**Note:** You cannot import an IBM® Cognos® Data Manager package in IBM® Cognos® Decision-Stream Series 7 or earlier.

Before you import a component package, it is recommended that you create a backup copy of the current catalog. For information, see “Backup an IBM Cognos Data Manager Catalog” (p. 360). To import a component package you use the Import Package wizard.
Review Information Messages

When you select the components to import, each component in the package is compared with the version in the target catalog. For each component that already exists in the target catalog, IBM® Cognos® Data Manager provides messages to help you decide whether you want to import them. There are two types of messages, information messages and warning messages.

A component may have information messages if, for example, an identical component exists in the target catalog, or if there is a different, newer version of the component in the target catalog. If you choose to import a component with an information message, as you progress through the Import Package wizard, Data Manager gives you the option of renaming the component. Information messages are shown with a yellow background.

Warning messages are shown if a component is different from the component with the same name in the target catalog. For example, if a hierarchy uses a different template, or if a template has different attributes. If you choose to import a component with a warning message, as you progress through the Import Package wizard Data Manager forces you to rename it. Warning messages are shown with a red background.

Tip: You can copy all the component names, together with any associated messages, to the clipboard. You can then paste them into another application and print them. To do this, click Copy to Clipboard.

Dependent Components

If you select a component to import that depends on unselected components, the Dependent Components window appears. Dependent components are separated into two groups, those that you must import and those that are optional.

IBM® Cognos® Data Manager provides an information message or a warning message for each of these dependent components.

If you want to remove a component from the import, clear the check box next to the component name. If a component is selected for removal and it has dependent components, the Dependent Components window appears listing all the components that must also be removed.

User-Defined Folders

If a package includes user-defined folders, for each component that belongs in the folder, the folder name appears in the Path column. By default, if you import a component that belongs in a user-defined folder, the folder is also imported.
If you want to import components without their associated user-defined folders, select the **Ignore folders** check box. If a component already exists in the target catalog, IBM® Cognos® Data Manager imports it into its current location in the catalog. If it does not exist in the target catalog, Data Manager imports it into the Builds and JobStreams folder.

### Rename Components

As you progress through the Import Package wizard, any components that you chose to import that had an information message are listed and you are given the option of renaming them. Any components that had a warning message are automatically renamed by IBM® Cognos® Data Manager, although you can change this name to something more appropriate if you require.

For those components for which renaming is optional, you can type a name for each component yourself, or click **Auto** for Data Manager to provide a default name for them all. If you do not want to rename these components, when the package is imported the component in the package overwrites that in the target catalog.

**Note:** User-defined folders do not appear in the list because you cannot rename them.

### Components With Conflicting Paths

If a component belongs to a user-defined folder, say Folder1\Folder2 and the same component already exists in the target catalog in a different user-defined folder, say Folder3\Folder2, IBM® Cognos® Data Manager provides a list of all the possible paths you can use to import the component. Select the required path from the Proposed Catalog Path list.

**Note:** If you select Builds and JobStreams, Data Manager imports the component into the Builds and JobStreams folder, rather than a user-defined folder.
Source Code Control Information

If source code control is being used and there are components you have selected for import that are not currently checked out, IBM® Cognos® Data Manager provides a list of these components. If you choose to progress through the Import Package wizard, Data Manager prompts you to check out the components so that you can complete the package import.

Note: You can obtain more source code control information about a listed component by clicking the Details button.

For more information, see "Source Code Control" (p. 347).

Use the Import Package Wizard

This section describes how to use the Import Package wizard.

Steps

1. Open the target catalog.
2. From the File menu, click Import Package.
   
   A message appears prompting you to backup the current catalog. Click Yes to backup the current catalog.

   The Package File dialog box appears.

3. Click the component package to import, and then click Open.
   
   The Import Package Wizard appears, with a full list of components included in the package.

4. If there are information messages shown for any listed components, ensure you review them before proceeding with the import.
   
   For more information, see "Review Information Messages" (p. 356).

5. For each component you want to import, select the check box next to the component name.
   
   ● To include all listed components, select the check box at the top of the column.
   
   ● To hide unselected components from view, select the Show selected only check box.

   If there are dependent components, the Dependent Components window appears. For more information, see "Dependent Components" (p. 356).

6. If you do not want to import components with their associated user-defined folders, select the Ignore folders check box.

   For more information, see "User-Defined Folders" (p. 356).

7. Click Next.

8. Rename the components as required.

   For more information, see "Rename Components" (p. 357).

9. If you are including user-defined folders, and there is a component with a conflicting path, select the required folder to use for import.
10. Click Finish.

Note: If source code control is being used and there are components that are not checked out, you must click Next to show details of those components and complete the package import. For more information, see "Source Code Control Information" (p. 358).

**Combine the Effort of Several Developers**

The recommended method of combining the efforts of several developers is for the developers to have personal catalogs and to use component packages to transfer components between these catalogs and a master catalog. This method is useful when you do not want to set up a source code control environment. It can also be used when developers are working in a very disconnected way, for example, exchanging components via email.

Combining developer catalogs consists of these main steps:

1. Create the master and developer catalogs.

2. In the master catalog, create any components that are common to all catalogs, and then create a component package that contains all these components.

3. Integrate the catalogs by
   - importing into each developer catalog, all components of the master component package
   - reconfiguring the connections if required, to suit each developer catalog

4. Develop the catalog components as required, ensuring that
   - no two developers modify the same component
   - no common components are modified except for reconfiguration of connections to local or development databases

5. In the developer catalog, create a component package that contains all the components to integrate.

6. Import the required components from the component package into the master catalog, taking care not to import common components.

**Backup and Restore Catalogs**

You can generate a text backup of a complete IBM® Cognos® Data Manager catalog or catalog tables, including all the catalog components. This is useful for backup or change control as the text file can be stored in a file management system. It is also useful if you want to move from one DBMS environment to another.

Note: Audit tables and delivery history are not included in a backup.

If you want to create a partial backup, you can select the components that you require and then save them as a package. For information, see "Component Packages" (p. 354).
Backup an IBM Cognos Data Manager Catalog

You can backup to an existing file or you can create a file.

**Steps**

1. From the File menu, click **Backup Catalog**.
2. Select the location of the existing file or where the new file is to be stored, and then either type a name for the backup file in the **File name** box, or select an existing file.
3. Click **Save**.
4. If you selected an existing file, a message appears stating that the file already exists. Click **Yes** to replace the file.

IBM® Cognos® Data Manager copies the data to the selected file.

Restore an IBM Cognos Data Manager Catalog

You may need to restore a catalog to a previous state using a backup text file. In IBM® Cognos® Data Manager, backup files are usually (*.ctg) files. You can also restore a component package (*.pkg) file using this method.

Before you restore a catalog, it is recommended that you create a backup copy of the current catalog. This is because Data Manager deletes all data from the current catalog before performing the restoration. If you do not create a backup copy, and the restore file is invalid in any way, your current catalog is lost.

For information, see "Backup an IBM Cognos Data Manager Catalog" (p. 360).

**Note:** You cannot restore a Data Manager catalog in IBM® Cognos® DecisionStream Series 7 or earlier.

**Steps**

1. Open the required catalog.
2. Create a backup copy of the catalog.
3. From the File menu, click **Restore Catalog**.

   A warning message appears informing you that restoring the catalog will overwrite the current catalog.
4. Click **Yes**.
5. Locate the backup file that you want to restore.

   **Important:** When you click **Open** in this dialog box, Data Manager deletes all data from the current catalog. If you have not created a backup copy, you may want to click **Cancel** and begin again.
6. Click **Open**.
Data Manager deletes all the data from the current catalog, and populates the catalog with the selected backup file.
Chapter 27: Using Pivots to Rotate Data

Data pivoting is a technique that treats multiple table columns as though they were a single column with multiple values. The term pivot is used because specified table columns rotate through 90 degrees to form rows.

In the following example, two columns have been created, Month and Sales, as a result of pivoting monthly sales figures.

<table>
<thead>
<tr>
<th>Product</th>
<th>January Sales</th>
<th>February Sales</th>
<th>March Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Month</th>
<th>Product</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>A</td>
<td>40</td>
</tr>
<tr>
<td>January</td>
<td>B</td>
<td>10</td>
</tr>
<tr>
<td>February</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>February</td>
<td>B</td>
<td>20</td>
</tr>
<tr>
<td>March</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>March</td>
<td>B</td>
<td>20</td>
</tr>
</tbody>
</table>

Create Pivots

A typical pivot rotates two or more data source columns to two DataStream items. One DataStream item maps to a transformation model element that records from which data source column the value originates. The second DataStream item maps to a transformation model element that contains the values from the data source columns.

Note: Pivoting data usually increases the number of data rows that a DataStream outputs. If you specify a row limit, this affects the number of rows output, not the number of source data rows processed by the DataStream. For information, see "Apply a Maximum Row Limit" (p. 158).

Steps
1. In the fact build for which you want to pivot data, click DataStream.
2. From the Edit menu, click Properties.
3. Click the DataStream Items tab, and then click Add to create a DataStream item.
4. In the Name box, type a name for the DataStream item.
5. Click the Pivot Values tab.
6. Click Import if the pivot values you want to define for the DataStream item already exist as columns or values in a database table.

   A menu appears prompting you to choose the location from which to import pivot values.
7. Click
Chapter 27: Using Pivots to Rotate Data

- **Add Columns from Data Source** or **Add Columns from Table** to import database column names. From the window that appears, click the column names to import as pivot values, and then click **OK**.

- **Add Values from Column** to import the values stored within a database column. From the window that appears, click the column containing the values to import as pivot values, and then click **OK**.

If you do not want to import pivot values, or you want to add further values, you can define them manually by clicking **Add** and then typing a name for the pivot value.

To reorder the list, click the pivot value that you want to move, and then click **Move Up** or **Move Down** as required.

8. When you have created a pivot value for each data source column to be pivoted, click **OK**.

9. Click the **DataStream Items** tab and expand the new DataStream item to display its pivot values.

10. Map each data source column to the appropriate pivot value and to any other DataStream items as required.

    For more information, see "Map Data Source Columns to the DataStream" (p. 152).

11. Click **OK**.

    **Tip:** Before mapping to the transformation model, you can check that the data is pivoted correctly by executing the DataStream.

    For more information, see "Execute a DataStream" (p. 160).

12. Click the transformation model for the fact build.

13. From the **Edit** menu, click **Mapping**.

    The **Transformation Model** window appears.

14. Create the required transformation model elements, and then map the DataStream items to them.

    For more information, see "Map the DataStream to the Transformation Model" (p. 201).

15. Click **OK**.

**Examples**

The following examples illustrate the different ways in which you can pivot data:

- create a single pivot
- create a double pivot
- create multiple pivot values

These examples are located in the DS_Advanced sample catalog.
**Example 1: Create a Single Pivot**

You can work through this example using the Pivot1 fact build in the DS_Advanced catalog. This table records the actual sales and forecast sales data for the products that a company sells.

<table>
<thead>
<tr>
<th>Product</th>
<th>Actual Sales</th>
<th>Forecast Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aloe Relief</td>
<td>598</td>
<td>701</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>1616</td>
<td>1739</td>
</tr>
<tr>
<td>Bear Survival Edge</td>
<td>1184</td>
<td>1118</td>
</tr>
<tr>
<td>Blue Steel Max Putter</td>
<td>118</td>
<td>114</td>
</tr>
<tr>
<td>Blue Steel Putter</td>
<td>352</td>
<td>334</td>
</tr>
</tbody>
</table>

You can pivot the data from the Actual Sales and Forecast Sales columns to two columns, named Scenario and Sales.

**Steps to Create the DataStream Items and Specify Pivot Values**

1. Create a fact build named **Pivot1** and set up a data source that accesses the Pivot1 data.
2. In the Pivot1 fact build, click the DataStream.
3. From the **Edit** menu, click **Properties**.
4. Click the **DataStream Items** tab, and then click **Add**.
5. In the **Name** box, type **Scenario**.
6. Click the **Pivot Values** tab.
7. Type the pivot values **ACTUAL** and **FORECAST**.
   These values correspond to the Actual Sales and Forecast Sales data source columns respectively.
8. Click **OK**.
9. Click the **DataStream Items** tab, and then click **Add**.
10. In the **Name** box, type **Sales**.
11. Click **OK**.
   Leave the **DataStream Properties** window open so that you can map the data source columns.

**Steps to Map the Data Source Columns**

1. In the **DataStream Properties** window, click the **DataStream Items** tab.
2. Drag the Product data source column from the left pane to the white space in the **DataStream Items** pane to create and map and DataStream item for the column.
3. In the DataStream Items pane, expand Scenario and then drag the following pivot values to the required data source column in the left pane.

<table>
<thead>
<tr>
<th>Pivot value</th>
<th>Data Source column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario:ACTUAL</td>
<td>ActualSales</td>
</tr>
<tr>
<td>Scenario:FORECAST</td>
<td>ForecastSales</td>
</tr>
</tbody>
</table>

4. Holding down the Ctrl key, drag the following DataStream items from the DataStream Items pane to the required data source column in the left pane.

<table>
<thead>
<tr>
<th>DataStream item</th>
<th>Data Source column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>ActualSales</td>
</tr>
<tr>
<td>Sales</td>
<td>ForecastSales</td>
</tr>
</tbody>
</table>

The DataStream Properties window should look like this.

5. Click OK.

**Steps to Execute the DataStream to Test the Pivoted Data**

1. In the Pivot1 fact build, click the DataStream.
2. From the Actions menu, click Execute.

The Execute DataStream window appears with the results of the pivoted data.
The Scenario column records from which data source column each value in the Sales column originates. Each row from the source table becomes two rows in the DataStream.

**Steps to Map the DataStream Items to the Transformation Model**

1. Click **Transformation Model**.
2. From the **Edit** menu, click **Mapping**.
   
   The **Transformation Model** window appears.
3. In the left pane, click the Product, Scenario, and Sales DataStream items.
4. Drag the selected items from the left pane to the white space in the **Transformation Model** pane.
5. When prompted, click **Create New as Attribute**.
   
   The selected DataStream items are mapped to the new transformation model elements.
   
   The **Transformation Model** window should look like this.
6. Click **OK**.
7. For each listed element, select the **Add to Fact Deliveries** check box.
8. Click **OK**.

**Example 2: Create a Double Pivot**

You can work through this example using the Pivot2 fact build in the DS_Advanced catalog.

You can pivot more than one data source within a DataStream. In this example, there are four data source columns named Actual Sales, Actual Revenue, Forecast Sales, and Forecast Revenue and they pivot to the single column named Amount. The intersection of the Scenario and Measure columns records which data source column contained each value.

<table>
<thead>
<tr>
<th>Product</th>
<th>Actual Sales</th>
<th>Actual Revenue</th>
<th>Forecast Sales</th>
<th>Forecast Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aloe Relief</td>
<td>598</td>
<td>2912</td>
<td>701</td>
<td>3407</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>1616</td>
<td>60525</td>
<td>1739</td>
<td>66992</td>
</tr>
<tr>
<td>Bear Survival Edge</td>
<td>1184</td>
<td>94238</td>
<td>1118</td>
<td>90601</td>
</tr>
</tbody>
</table>

To define this double pivot, you must create three DataStream items, Scenario, Measure, and Amount.
The Scenario DataStream item has two pivot values, and records whether each value in the Amount column is actual data or forecast data.

The Measure DataStream item also has two pivot values and records whether each value in the Amount column relates to sales or revenue.

On the DataStream Items tab of the DataStream Properties window, the pivot looks like this.

Simultaneously performing two pivots results in this table.

<table>
<thead>
<tr>
<th>Product</th>
<th>Scenario</th>
<th>Measure</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aloe Relief</td>
<td>ACTUAL</td>
<td>SALES</td>
<td>598</td>
</tr>
<tr>
<td>Aloe Relief</td>
<td>FORECAST</td>
<td>SALES</td>
<td>701</td>
</tr>
<tr>
<td>Aloe Relief</td>
<td>ACTUAL</td>
<td>REVENUE</td>
<td>2912</td>
</tr>
<tr>
<td>Aloe Relief</td>
<td>FORECAST</td>
<td>REVENUE</td>
<td>3407</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>ACTUAL</td>
<td>SALES</td>
<td>1616</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>FORECAST</td>
<td>SALES</td>
<td>1739</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>ACTUAL</td>
<td>REVENUE</td>
<td>60525</td>
</tr>
<tr>
<td>Bear Edge</td>
<td>FORECAST</td>
<td>REVENUE</td>
<td>66992</td>
</tr>
<tr>
<td>Bear Survival Edge</td>
<td>ACTUAL</td>
<td>SALES</td>
<td>1184</td>
</tr>
<tr>
<td>Bear Survival Edge</td>
<td>FORECAST</td>
<td>SALES</td>
<td>1118</td>
</tr>
</tbody>
</table>
Example 3: Create Multiple Pivot Values

You can work through this example using the Pivot3 fact build in the DS_Advanced catalog.

No limit exists on the number of pivot values that you can define for a DataStream item. However, when you define a data source pivot, you must declare all the values that can exist in the corresponding DataStream item.

For example, the following table shows sales by month for each product.

<table>
<thead>
<tr>
<th>Product</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Dome</td>
<td>2.0</td>
<td>1.76</td>
<td>4.17</td>
<td>5.24</td>
<td>3.53</td>
<td>4.81</td>
<td>4.29</td>
<td>3.40</td>
<td>3.96</td>
<td>9.38</td>
<td>2.81</td>
<td>2.68</td>
</tr>
</tbody>
</table>

To pivot this information to a table that has one column for the name of the month and another to record the sales value, you must create three DataStream items, Product, Month, and Sales.

The Month DataStream item has twelve pivot values, one for each month.

On the DataStream Items tab of the DataStream Properties window, the pivot looks like this.

You must then map the DataStream items to three transformation model elements.

The Transformation Model window, should look like this.
This table shows the result of pivoting the original data to create Month and Sales columns.

<table>
<thead>
<tr>
<th>Product</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Dome</td>
<td>Jan</td>
<td>2.20</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Feb</td>
<td>1.76</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Mar</td>
<td>4.17</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Apr</td>
<td>5.24</td>
</tr>
<tr>
<td>Star Dome</td>
<td>May</td>
<td>3.53</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Jun</td>
<td>4.81</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Jul</td>
<td>4.29</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Aug</td>
<td>3.40</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Sep</td>
<td>3.96</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Oct</td>
<td>9.38</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Nov</td>
<td>2.81</td>
</tr>
<tr>
<td>Star Dome</td>
<td>Dec</td>
<td>2.68</td>
</tr>
</tbody>
</table>
IBM® Cognos® Data Manager can reject data that it is acquiring or delivering. You control how Data Manager handles this rejected data.

You can specify that rejected source data is saved to a text file and a SQLTXT definition file, or a relational table. You can also choose to send rejected fact data to a fact delivery where it can be saved.

Rejected Source Data

During data acquisition, IBM® Cognos® Data Manager can reject a row of source data if either of these is true:

● the data row duplicates the combination of dimension values of a data row acquired earlier during data acquisition.

● the value of a dimension element does not match a member of the corresponding hierarchy.

Reject Duplicate Source Data

Duplicate data source rows have identical dimension elements. The first data row is accepted and any subsequent duplicate data rows are rejected.

You can save the rejected data rows, edit them, and then execute the data rows again. You can treat this file as a data source using IBM® Cognos® Data Manager SQLTXT Designer.

Note: You can also accept or merge duplicate source data rows. For information see "Handle Duplicate Data" (p. 138).

Steps

1. Click the required fact build.

2. From the Edit menu, click Properties.

3. Click the Input tab.

4. Select Reject records with duplicate keys.

5. If you want to change the default name and location of the reject file, in the File name box, enter the full directory path and file name that you require.

   Tip: Click the browse button to open the Select Reject File dialog box where you select the required file.

6. Click OK.
The Visualization pane is refreshed to show graphically that duplicate records are to be rejected.

7. Manually edit the data in the reject file to eliminate the problem that caused the data to be rejected, and then execute the build again. You can do this either by editing the reject file directly, or by using SQLTerm.

You can use the Data Manager SQLTXT DBMS driver to further process the rejected data.

Reject Unmatched Members

A dimension element is rejected in the transaction data if it is associated with a hierarchy or lookup, but its ID cannot be found in the corresponding reference structure. You can stop this automatic rejection in one of two ways. You can

- map the dimension element to the hierarchy or lookup
  For information, see "Associate a Dimension or Derived Dimension Element with a Reference Item" (p. 171).

- specify that you want to include the unmatched member
  For information, see "Accept Source Data Containing Unmatched Dimension Values" (p. 179).

Save Rejected Source Data

When IBM® Cognos® Data Manager rejects records during build execution, the rejected data can be saved to

- file
  Data Manager creates a simple tab delimited text file. The first time a build that creates reject records is executed, Data Manager creates a text file. If the build has been executed previously, the existing data in the text file is replaced.
  By default, the text file is named <build_name>.rej and is stored in the Data Manager Data directory.
  Saving reject records to file is the default.

- table
  This table can reside in any writeable Data Manager connection.

If you do not want to save rejected records, click Don’t write reject records.

Steps to Save Rejected Data to a File

1. Click the required fact build.

2. From the Edit menu, click Properties.

3. Click the Input tab.

4. In the Reject Record Handling box, click Write reject records to file.
5. If you want to change the default name and location of the reject file, in the **File name** box, enter the full directory path and file name.

   **Tip:** Click the browse button to open the **Select Reject File** dialog box where you select the required file.

6. Click **OK**.

**Steps to Save Rejected Data to a Table**

1. Click the required fact build.

2. From the **Edit** menu, click **Properties**.

3. Click the **Input** tab.

4. In the **Reject Record Handling** box, click **Write reject records to table**.

5. In the **Connection** box, click the connection that contains the table where the rejected records are to be saved.

   **Tip:** To create a connection, click **New**. For information on creating a connection, see "Create a Database Connection" (p. 63).

6. In the **Table name** box, enter the name of the table to which you want the rejected data to be saved.

   **Tip:** You can also click the **Browse for table** button and then click the required table from the list in the **Select Table** dialog box.

7. Click **OK**.

**Rejected Delivery Data**

During data delivery, a row of fact data can be rejected if either of these is true

- the data row does not match any level filter
- the data row evaluates to FALSE in an output filter

Primarily, filters are used to restrict the fact data being delivered. However, they can also be used to reject unwanted data.

**Use a Level Filter to Reject Data**

You can use level filters to configure a delivery to accept only specific dimension and hierarchy levels. Data is rejected unless its dimension values match at least one of its level filters.

For more information, see "Use a Level Filter to Deliver Specific Dimensions and Hierarchy Levels" (p. 219).
Use an Output Filter to Reject Data

An output filter rejects all rows for which the expression evaluates to FALSE. Each delivery may have no more than one output filter.

You can also use an output filter together with a derivation to reject data. For example, you could create a derivation that calculates profit margin. If the profit margin is less than 50%, IBM® Cognos® Data Manager sets the field to "<50%", if it is over 50% to ">50%", and if the margin is negative, to "Negative Margin". You can then create an output filter that has an expression that evaluates "Negative Margin" to FALSE and so rejects the data.

For more information, see "Use an Output Filter to Deliver Specific Data Rows" (p. 220), and "Derivations" (p. 155).

Save Data Rejected During Delivery

To save rejected data on delivery, create a fact delivery to send the rejected data to a suitable database in any supported DBMS.

Steps

1. Determine an expression that evaluates to TRUE only for the rejected data. If necessary, you can create a derivation transformation model element against which to apply the filter expression.

2. Create a fact delivery to save the unwanted data. This fact delivery must be the topmost fact delivery in the Build tree.

3. Click the delivery, and from the Edit menu, click Properties.

4. Create an output filter that uses the expression you determined in step 1.

   See "Use an Output Filter to Deliver Specific Data Rows" (p. 220).

5. To ensure that subsequent deliveries are not offered the same data rows, click the General tab and then select the Exclusive check box.

6. Execute the build.

7. To prepare the rejected data for reprocessing, edit it in the DBMS to which it was delivered or in SQLTerm.

Specify the Encoding to Use for Reject Files

If you are using IBM® Cognos® Data Manager with multilingual or unicode data, you should specify unicode encoding for your reject files. By default, the encoding used is platform dependent.

You can specify the encoding using one of these methods:

- enter a command specifying the encoding to use in the Execute Build or Execute JobStream dialog box
- set an environment variable
- specify the encoding in the dm.ini file
For information on using the last two methods, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Steps to Specify the Encoding to use From the Execute Dialog Box

1. Select the build or JobStream for which you want to specify the encoding.
2. From the Actions menu, click Execute.
3. In the Additional options box, use the following syntax to specify the file name (or variable) and the encoding to use:
   
   `<filename|variable>!<encoding>`

   For example
   
   `-VREJECT_FILE={$DS_BUILD_FNAME}.rej!UTF-16`
Chapter 29: Managing Memory

A fact build that merges or aggregates high volumes of data, and contains a significant number of objects, such as attributes, transformation model elements, and so on, may use a large amount of memory.

When performing aggregation, each accepted (input) row from the DataStream may contribute to many output rows for potential delivery. The maximum number of output rows is the product of the number of output levels of each hierarchy upon which aggregation occurs. For example, if a build uses seven dimensions, and each dimension uses a hierarchy with five output levels, then the maximum number of output rows is $5 \times 5 \times 5 \times 5 \times 5 \times 5 \times 5$, or 78,125 per input row.

To better manage memory, you can limit the memory IBM® Cognos® Data Manager uses during fact build execution by using dimension breaking or by setting a memory limit.

Using dimension breaking allows Data Manager to clear the memory of all data associated with dimension values that will not reoccur in the input stream. Setting a memory limit within Data Manager prevents an operating system from terminating an application that requests more memory than the operating system can provide. It also ensures that Data Manager uses its own paging algorithm rather than relying on the operating system to provide this function.

Data Manager allocates memory for several purposes.

Structure Data

IBM® Cognos® Data Manager stores all the reference structure data, and caches it at the start of build execution. When built, this structure is static and is not subject to application paging.

You can reduce the memory used by caching reference data attributes in a temporary file rather than memory. You do this from the Lookup Properties dialog box by selecting the Use file caching for attributes box.

Dimension Domains

There are two domain types, dynamic and reference. When using a dynamic domain, new members are added as their values are detected in the transactional input stream. When using a reference domain, all the members from the associated reference structure, irrespective of whether they are referenced by the transactional input stream, are loaded into memory.

For more information, see "Dynamic and Reference Domains" (p. 179).

The Hash Table

The hash table is an internal IBM® Cognos® Data Manager structure that is used to process and transform rows of data. Data Manager normally uses hashing to track memory required for aggregation and to merge duplicate rows.
The Page Pool

The page pool contains fixed-size data pages in which IBM® Cognos® Data Manager stores aggregated data and the associated hash table for indexing the data.

The Page Table

The page table provides IBM® Cognos® Data Manager with location and content details for all the data and hash pages currently in use.

Dimension Breaks

During data acquisition, partially built aggregates and all data that may contribute to future aggregations are stored. Breaking is the process of tracking changes in ordered dimensions to reduce memory usage. When a change is detected in a sorted input stream, all aggregates can be cleared from memory, releasing the memory that the data occupied.

For example, the following hierarchy contains year, quarter, and month levels.

```
Year
  Q1
    Jan Feb Mar
  Q2
    Apr May Jun
```

When all the records for March have been processed, the memory is cleared of the March records, the hierarchy is checked, and all the records for Q1 are flushed. Processing continues until all the records for June have been processed, when memory is cleared of all the records for Q2 and Year.

In another example, suppose the acquired data is ordered on a City dimension, and data rows that have a city value of London are being processed. The first row to arrive with a city value other than London implies that all the data rows that have the value London have been received.

You can perform breaking on a single dimension or, more usually, on a combination of dimensions. **Note:** You cannot perform breaking on a derived dimension or data aggregated using aggregate rules.

Breaking requires that the data is ordered consistently across all data sources that are being merged. When data is sourced from more than one data source and the databases are not using the same collation sequence, problems can occur. You must therefore ensure that a compatible collation sequence is used to order the data for each data source. For more information see, "Distributed Sorting" (p. 64).

To perform breaking you must
- determine the dimensions on which to break
- set the dimension breaks
- specify that sorting is to be performed
specify when to break

Determine the Dimensions on which to Break
Determining the dimensions on which to break requires a mixture of analysis, experience, and the evaluation of different options.

A dimension that is suitable for breaking
- is not aggregated, or is aggregated through few levels, with relatively few aggregated members generated
- has a large domain that can be broken into smaller segments
- is a balanced hierarchy, having an even distribution of members within parents

Set Dimension Breaks
When you have determined the dimensions on which to break, you are ready to set the dimension breaks and specify their behavior.

Steps
1. Click the required fact build.
2. From the Edit menu, click Properties.
3. Click the Breaks tab.
4. Move the dimensions that you want to use for breaking from the Dimensions pane to the Break On pane.
   Tip: In addition to using the arrow buttons, you can also double-click a dimension to move it to the Break On pane.
5. Ensure that the order of the dimensions in the Break On pane is the order in which breaking is to be performed. You can change this order using the up and down arrow buttons.
6. Click OK.

Specify that Sorting is to be Performed
Breaking can only function correctly if each data source is sorted by the same set of dimensions, in the same sequence, and is of the equivalent character sets. Sorting is often performed in the database, for example, indexing often orders the data.

However, you can specify that sorting is performed by adding an ORDER BY clause to the source SQL or by selecting the Force Sort on Break Dimensions check box in the Fact Build Properties window.

An error occurs if the database does not support ORDER BY clauses, or if the SELECT statement is made invalid by adding an ORDER BY clause. For example, if there is already an ORDER BY clause or a FOR READ (db2) clause.
Chapter 29: Managing Memory

Note: Some data sources such as SQLTXT and Published FM Packages cannot have the sort applied and an error will occur. In these cases, you must manually add the sort to the data source or ensure that the data rows are pre-sorted.

You specify that sorting is to be performed after you set the required dimension breaks.

Steps
1. Click the required fact build.
2. From the Edit menu, click Properties.
3. Click the Breaks tab.
4. Select the Force Sort on Break Dimensions check box.
5. Click OK.

Specify When to Break
Data should not necessarily be flushed on every break because identifying the completed row takes time.

You specify when breaking is to occur after you set the required dimension breaks.

Specifying when breaking is to occur can greatly improve break performance, especially when merging or aggregating on dimension elements with very high distinct cardinality. Using this setting, 60% is recommended, can reduce the number of breaks that occur and improve overall performance because memory is utilized more efficiently.

Steps
1. Click the required fact build.
2. From the Edit menu, click Properties.
3. Click the Breaks tab.
4. Select the Perform break processing every check box, and then
   - select Breaks to specify a break every n dimension changes
   - select Percent to specify that the break is to be based on the percentage of the hash table that is used
5. In the box to the left of the option buttons, type a number to indicate the number of dimension changes or the required percentage (as a number between 1 and 100).
6. Click OK.

Processing Data Sources in Series or in Parallel
By default, data sources are read in series unless breaking is used, in which case they are read in parallel.
When data sources are read in parallel, IBM® Cognos® Data Manager orders the data sources using the order of the dimensions that you specified on the Breaks tab of the Fact Build Properties window. If there are multiple data sources, processing in parallel forces the ordering to apply to all the dimensions. This affects performance, because Data Manager compares the dimension values from the current row of each data source, to ascertain which is the next in order. However, if you know that there is no ordered overlap between the data from all the data sources, then you can avoid this performance overhead by executing the data sources in series. For example, if a FiscalPeriod dimension was first in the break on list and data source one contained data for periods 1995 to 1999 and data source two contained data for periods 2000 onwards, you could use execute in series. If you execute in series and there are overlaps in the data, then you will have duplicate data.

When data sources are processed in series, the order in which they appear in the Tree pane is used. You can change this order if you require. For information, see "Data Source Processing Order" (p. 159).

**Steps to Process Data Sources in Series When Breaking is Used**

1. Click the DataStream.
2. From the Edit menu, click Properties.
3. Click the Input tab.
4. Select the Execute data sources in series check box.
5. Click OK.

**Memory Limits**

You may want to limit the amount of memory that IBM® Cognos® Data Manager requests from the operating system for these reasons:

- Some operating systems terminate any application that requests more memory than the operating system can provide.
- If Data Manager must share resources with other applications running on the computer, you may want to prevent Data Manager from requesting memory that the other applications require.

**Tips**

- To obtain useful information for determining optimum memory settings, execute the build in check only mode.
- To obtain this information, you must specify that Internal details are included in the log file. For information, see "Specify Log File Details" (p. 283).

**Maximum Memory Usage**

You can determine the maximum amount of memory that IBM® Cognos® Data Manager requested when executing a build by inspecting the log file. Search for the last entry of this format:

```
[INTERNAL - hh:mm:ss]Mem(M): x/Peak=y(details)
```
The value of y (the peak value) gives the maximum amount of memory in MB that Data Manager requested.

If Data Manager reaches the limit of available memory, it creates virtual memory by paging information to disk. You can determine whether this happens by inspecting the log file. Search for entries of this format:

\[\text{INTERNAL - hh:mm:ss} \text{Paging: n1 pages, n2 in pool, n3 fault(s)}\]

The presence of one or more such lines indicates that Data Manager has exhausted available real memory. The value of n3 indicates the number of times it occurred.

For information about how to specify the maximum amount of memory to be used, see "Set Memory Limits" (p. 383).

**Maximum Page Size**

The page size determines how much paging will occur, and how many data rows IBM® Cognos® Data Manager stores on each page. The default page size is 16 KB. Data Manager adjusts this value (or any specified value) downwards to accommodate a whole number of data rows.

You can determine the page size that Data Manager used for the build by searching the log for an entry of this format:

\[\text{INTERNAL - hh:mm:ss} \text{Paging: page size nn}\]

This entry appears near the top of the entries for the execution and expresses the page size in bytes.

For information about specifying the maximum page size, see "Set Memory Limits" (p. 383).

**Initial Size of the Page Table**

By default, IBM® Cognos® Data Manager determines the initial size of the page table from the size of the hash table.

For information about changing this value, see "Set Memory Limits" (p. 383).

**Initial Size of the Hash Table**

The hash table is an internal IBM® Cognos® Data Manager structure that is used to process and transform rows of data. Data Manager normally uses hashing to track memory required for aggregation and to merge duplicate rows.
At several times during execution, Data Manager shows statistics about the hash table. The following message indicates that 118 pages are used for the hash slots, 135173 of the 150000 slots of the page are in use, with none marked as reusable.

```
[INTERNAL - 11:14:05]Hash: 118 page(s), 135173 of 150000 slots in use (90%), 0 reusables, avg 2.56 hits
```

The 'avg hits' value is the average number of probes (hits) into the hash table to find slots being searched for. The counters used to compute the value are reset each time it is reported. Hit rates higher than three may indicate that a hash table should be increased in size. When a build completes, the log reports the minimum hash table size that the build requires. In the following example, *nnn* is the table size in slots:

```
[INTERNAL - 11:14:05]Hash: minimum table size nnn
```

Allocating a larger hash table at the start of the build reduces the overhead associated with resizing the hash table during build execution.

By default, the initial size of the hash table is 200,000 slots. However, Data Manager precalculates the maximum possible number of hash table slots and uses the precalculated value when this is smaller than the specified number of slots.

As a build progresses, the hash table may become full which causes Data Manager to extend the hash table. Data Manager resizes the table to 150% of the previous size. You can determine whether the hash table was restructured or resized by searching the log. Entries of this format indicate that resizing occurred:

```
[INTERNAL - hh:mm:ss]Hash: table full, restructuring
[INTERNAL - hh:mm:ss]Hash: resizing table from xx to yy
[INTERNAL - hh:mm:ss]Hash: table restructuring complete
```

Where possible, you should determine the minimum hash table size from the log, and specify this value on the Memory tab of the Fact Build Properties window. If you need to specify a lower value, then use a value that will result in the minimum hash table size after resizing. You can calculate suitable values by multiplying the minimum hash table size by \((2/3)n\) where \(n\) is an integer.

For example, if the log indicates that the minimum hash table is 10,000 slots, then, rounding up to the nearest integer:

\[
10,000 \times \frac{2}{3} = 6,666.67 \\
10,000 \times \left(\frac{2}{3}\right)^2 = 4,444.44 \\
10,000 \times \left(\frac{2}{3}\right)^3 = 2,962.96
\]

In this example, 6670, 4450, and 2963 make good starting values.

### Set Memory Limits

This section describes how to set memory limits.

#### Steps

1. Click the required fact build.

2. From the Edit menu, click Properties.

3. Click the Memory tab.

4. In the Working memory limit (MB) box, type a suitable value to specify the maximum amount of memory to be used.
Chapter 29: Managing Memory

For more information, see "Maximum Memory Usage" (p. 381).

5. In the Max page size (bytes) box, type a suitable value to specify the maximum page size.
   For more information, see "Maximum Page Size" (p. 382).

6. In the Initial page table size (slots) box, type a suitable value to specify the initial size of the page table.
   For more information, see "Initial Size of the Page Table" (p. 382).

7. In the Initial hash table size (slots) box, type a suitable value to specify the initial size of the hash table.
   For more information, see "Initial Size of the Hash Table" (p. 382).

8. Click OK.
Chapter 30: IBM Cognos Data Manager SQLTXT Designer

You use IBM® Cognos® Data Manager SQLTXT Designer to configure the SQLTXT definition files used within IBM® Cognos® Data Manager to access text files as though they are in an SQL database.

Data Manager also supports direct access to data using named pipes. For more information, see "Use Named Pipes to Access Data" (p. 405).

You can generate test data through the MAKEROWS SQLTXT database.

You access SQLTXT databases through the SQLTXT connection type.

SQLTXT Definition Files

A SQLTXT definition file (.def) defines a virtual database of text file tables. It associates each table with a text file in the database directory. It also specifies the structure of the tables.

The SQLTXT definition file contains a number of entries. The first line of each entry consists of the table name enclosed in square brackets. Following this are definition lines that define the contents of the associated text file. Each definition line starts with a keyword and includes a number of parameters. White space separates the keyword and parameters.

You can include comments within the SQLTXT definition file. These commence with two forward slashes like this //, and continue to the end of the current line. Blank lines are ignored.

A SQLTXT definition file must be stored using a unicode encoding, such as UTF-16, if any table or column name contains characters not in the current code page. For example, when using Japanese table names in a Microsoft® Windows® operating system-1252 environment.

Steps to Create a SQLTXT Definition File

1. In IBM® Cognos® Data Manager SQLTXT Designer, from the File menu, click New.

2. From the File menu, click Save As.

3. In the File name box, type a name for the file.
   
   Note: The definition file must have the extension .def

4. Click Save.

   When you have created a definition file, you must add tables and columns to it. You can do this from Data Manager SQLTXT Designer using the Import wizard, or by adding a table and columns manually.

   For more information, see "Import a Table Definition using the Import Wizard" (p. 397), or "Define a Table Manually" (p. 402).
SQLTXT Text File Format

In addition to a definition file, a SQLTXT database requires a text file for each SQL table being accessed.

Text files can exist in a variety of character sets and contain any of these types of rows:

- Header rows provide metadata about the data rows in the text file and are required by some applications to provide names for the data columns in the table. Header rows are optional in a SQLTXT database because the definition file provides this information.
- Data rows.
- Trailer rows provide optional information. However, they limit the range of operations that IBM® Cognos® Data Manager can perform on the file. Trailer rows prevent Data Manager from appending data to the file, effectively limiting interaction to SELECT and DROP statements.

Blank data rows are ignored but blank header or trailer rows are not.

Wildcards in SQLTXT File Names

If the text file associated with a SQLTXT table has a wildcard in the file name, rows are taken from all the files that match the wildcard.

This is useful where
- different source systems have identically formatted but differently named files in a directory
  Using wildcards means that these can be treated collectively.
- files have names that contain dates, perhaps where a file arrives every day
  These files can be referred to with a single logical name.

SQLTXT Column Names

Column names cannot begin with a number, and should not contain characters other than letters and numbers. If a column name contains other characters they are removed. If this results in duplicate column names, the duplicate column name is appended with a sequential number.

SQLTXT Record Types

SQLTXT can read and write

- delimited records with either delimited or fixed width columns
- nondelimited records

SQLTXT Record Delimiters

When you are defining delimited records, you must specify the delimiter used to separate the individual records in the file. The following table describes each of the delimiters:
<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Description</th>
</tr>
</thead>
</table>
| NL        | New line. This value is dependent on the platform on which IBM® Cognos® Data Manager is running, and the locale of the input file. The supported options are  
  - for an ASCII file on a Microsoft® Windows® operating system platform, the byte sequence is 13 10  
  - for an ASCII file on a UNIX® operating system platform, the byte sequence is 10  
  - for an EBCDIC file on any platform, the byte sequence is 21 |
| {CR}      | Carriage return. The byte sequence is ASCII 13. |
| {LF}      | Line-feed character. The byte sequence is ASCII 10. |
| {CRLF}    | Carriage return and line feed combination. The byte sequence is ASCII 13 10. |
| Other     | Another delimiter is used. In the Other box, type the delimiter values.  
  **Note:** You must specify delimiter values using byte sequences. Each byte must be introduced with a backslash, followed by the byte value specified as a 3 digit decimal value. For example, to use the carriage return delimiter, type 013\010. |

**Note:** The options in { } are defined by byte sequences. These options are translated into the selected character set for the file. For example, if the file is ASCII, and {LF} is selected, byte sequence 10 is the line delimiter. If EBCDIC is chosen, the ASCII byte sequence is translated as follows:  
ASCII 10 becomes EBCDIC 13  
ASCII 13 becomes EBCDIC 13  
EBCDIC NL becomes ASCII 21

### SQLTXT Column Delimiters

When you are defining delimited records with delimited columns, you must specify the delimiter used to separate the columns. The following table describes each of the delimiters:

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>Columns are tab-delimited.</td>
</tr>
<tr>
<td>Comma</td>
<td>Columns are comma-delimited.</td>
</tr>
</tbody>
</table>
Columns are delimited by at least two characters. These can be space, tab, or Enter. You cannot enter a single space.

Columns are delimited by one or more space characters.

Another delimiter is used. In the Other box, type the delimiter code.

### Quotation Characters in SQLTXT Delimited Records

In delimited records with delimited columns, a value may be enclosed within quotation characters, for example, "text". The use of quotation characters is optional. However, by using quotation characters, you can include values that contain the inter-column delimiter. For example the text "31, Acacia Avenue" must be enclosed in quotation characters in a comma-delimited file.

If you want to include a quotation character in a value that is enclosed in quotation characters, you must escape the included quotation character with another. For example, if the quotation character is a quotation mark, the value 'te"xt' must appear as 'te" "xt' if the string in which it appears is enclosed in quotation marks.

When defining delimited records, you must specify whether columns are enclosed in quotation characters. The following table describes each of the quotation characters:

<table>
<thead>
<tr>
<th>Quotation character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Columns are enclosed by double quotation marks.</td>
</tr>
<tr>
<td>Single</td>
<td>Columns are enclosed by single quotations marks.</td>
</tr>
<tr>
<td>None</td>
<td>Columns are not enclosed or a closing quotation character is missing for one or more fields.</td>
</tr>
<tr>
<td>Other</td>
<td>Columns are enclosed by another character. In the Other box, type the character.</td>
</tr>
</tbody>
</table>

SQLTXT removes quotation characters during the import process.

**Note:** Quoted fields must start and end with the same quotation character. If a quotation character is missing, IBM® Cognos® Data Manager continues to search the record, ignoring the column delimiters.

### Special Characters as Delimiters

To use a nonprinting or special character as a delimiter, type a backslash, followed by the decimal value. For example, to use ESC as a delimiter, type \27, and to use AB, type \065\066.
SQLTXT Data Types

SQLTXT supports the following data types for SQLTXT columns:

- BINARY
- BLOB
- BOOLEAN
- CHAR
- CLOB
- DATE
- DATE WITH TIMEZONE
- FLOAT
- INTEGER
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- NUMBER
- TIME
- TIME WITH TIMEZONE

**BINARY**

BINARY can be used in delimited and fixed width columns.

This data type interprets binary values as hexadecimal string values.

When setting up a column type of BINARY, you must specify the length of the column. In delimited columns, the length equates to the maximum number of bytes permitted for the column. In fixed width columns, the length equates to the exact number of bytes of data. Binary values are held in hexadecimal format in the data file.

**Note:** The length of a BINARY column can be 0 to 8192. If you specify 0 as the length, IBM Cognos Data Manager sets the length to 8192 by default.

**BLOB**

BLOB can be used only in delimited columns.

This data type can be used to represent large binary objects. The BLOB itself is held in a file and the data file contains a reference to this BLOB file in the format "<< filename >>".

When setting up a column type of BLOB, you must specify the maximum length.

**Note:** There is no limit to the size of a BLOB column (other than available disk space). If you specify 0 as the length, Data Manager sets the length to 8192 by default.
**BOOLEAN**

BOOLEAN can be used in delimited and fixed width columns.

This data type interprets the Boolean values TRUE and FALSE as string values.

When setting up a fixed width column type of BOOLEAN, you must specify the length of the column. The length equates to the exact number of bytes of data. For example, the value TRUE consists of four bytes.

DataManager interprets the value as T, t and 1 as TRUE, and F, f and 0 as FALSE. All other values are interpreted as TRUE.

**Note:** In fixed width columns, the length of a BOOLEAN column can be 0 to unlimited. If you specify 0 as the length, DataManager sets the length to 1 by default.

**CHAR**

CHAR can be used in delimited and fixed width columns.

This data type interprets letters and numbers as string values.

When setting up a column type of CHAR, you must specify the length of the column. In delimited columns, the length equates to the maximum number of bytes of data permitted for the column. In fixed width columns, the length equates to the exact number of bytes of data. For example, the value MYVALUE consists of seven bytes.

**Note:** The length of a CHAR column can be 0 to unlimited. If you specify 0 as the length, DataManager sets the length to 30 by default.

**CLOB**

CLOB can be used only in delimited columns.

This data type can be used to represent large string objects. The CLOB itself is held in a file in UTF-16 encoding and the data file contains a reference to this CLOB file in the format "<< filename >>".

When setting up a column type of CLOB, you must specify the maximum length.

**Note:** There is no limit to the size of a CLOB column (other than available disk space). If you specify 0 as the length, DataManager sets the length to 8192 by default.

**DATE**

DATE can be used in delimited and fixed width columns.

This data type interprets dates as string values.

When setting up a column type of DATE, you must specify the format of the date. In delimited and fixed width columns, the format must equate to the format of the date in text file.

**Note:** The format can be any valid date format. If no format is given, DataManager sets the format to syyyy-mm-dd hh:mm:ss.fffffffff by default.

For more information, see "Date String Formats in SQLTXT Files" (p. 394).
DATE WITH TIME ZONE
DATE WITH TIME ZONE can be used in delimited and fixed width columns.
This data type interprets dates with a time zone as string values.
When setting up a column type of DATE WITH TIME ZONE, you must specify the date format.
In delimited and fixed width columns, the format must equate to the format of the date in the data file.

Note: The format can be any valid date format followed by a valid time zone format. If no format is given, Data Manager sets the format to syyyy-mm-dd hh:mm:ss.fffffffff stzh:tzm by default.
For more information, see "Date String Formats in SQLTXT Files" (p. 394).

FLOAT
FLOAT can be used in delimited and fixed width columns.
This data type interprets floating point numbers as string values.
When setting up a fixed width column type of FLOAT, you must specify the length of the column.
The length equates to the exact number of bytes of data. For example, the value 4.2000 consists of six bytes.

Note: In fixed width columns, the length of a FLOAT column can be 0 to 30. If you specify 0 as the length, Data Manager sets the length to 30 by default.

INTEGER
INTEGER can be used in delimited and fixed width columns.
This data type interprets integer values as string values.
When setting up a fixed width column type of INTEGER, you must specify the length of the column.
The length equates to the exact number of bytes of data. For example, the value 12345 consists of five bytes of data.

Note: In fixed width columns, the length of an INTEGER column can be 0 to 19. If you specify 0 as the length, Data Manager sets the length to 19 by default.

INTERVAL DAY TO SECOND
INTERVAL DAY TO SECOND can be used in delimited and fixed width columns.
This data type interprets day to second intervals as string values.
When setting up a column type of INTERVAL DAY TO SECOND, you must specify the interval format.
In delimited and fixed width columns, the format must equate to the format of the interval in the data file.

Note: The format can be any valid day to second interval format. If no format is given, Data Manager sets the format to sdddddddddd hh:mm:ss.fffffffff by default.
For more information, see "Date String Formats in SQLTXT Files" (p. 394).

INTERVAL YEAR TO MONTH
INTERVAL YEAR TO MONTH can be used in delimited and fixed width columns.
This data type interprets year to month intervals as string values.

When setting up a column type of INTERVAL YEAR TO MONTH, you must specify the interval format. In delimited and in fixed width columns, the format must equate to the format of the interval in the data file.

**Note:** The format can be any valid year to month interval format. If no format is given, Data Manager sets the format to syyyyyyyyy-mm by default.

For more information, see "Date String Formats in SQLTXT Files" (p. 394).

**NUMBER**

NUMBER can be used in delimited and fixed width columns.

This data type interprets precise numbers as string values.

When setting up a column type of NUMBER, you must specify the length and scale of the column. In delimited columns, the length equates to maximum number of bytes of data permitted for the column. In fixed width columns, the length equates to the exact number of bytes of data. The scale represents the number of digits after the decimal point.

For example, the value 12345.67890 consists of 11 bytes of data, and has a scale of five.

**Notes**

- The length of a NUMBER column can be 1 to 77 and the scale can be 0 to 77. The length must be greater than the scale.
- The precision (number of digits) is assumed to be the length.

**TIME**

TIME can be used in delimited and fixed width columns.

This data type interprets times as string values.

When setting up a column type of TIME, you must specify the time format. In delimited and in fixed width columns, the format must equate to the format of the time in the data file.

**Note:** The format can be any valid time format followed by a valid time zone format. If no format is given, Data Manager sets the format to hh:mi:ssffffff stzh:tzm by default.

For more information, see "Date String Formats in SQLTXT Files" (p. 394).

**TIME WITH TIMEZONE**

TIME WITH TIME ZONE can be used in delimited and fixed width columns.

This data type interprets times with a time zone as string values.

When setting up a column type of TIME WITH TIME ZONE, you must specify the date format. In delimited and in fixed width columns, the format must equate to the format of the time in the data file.

**Note:** The format can be any valid time format followed by a valid time zone format. If no format is given, Data Manager sets the format to hh:mi:ssffffff stzh:tzm by default.

For more information, see "Date String Formats in SQLTXT Files" (p. 394).
PACKED

PACKED can only be used in fixed width columns.

This data type interprets decimal numbers with two digits per byte. For example, the value 123.45 consists of five digits and is represented as follows:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>3 4</td>
<td>5 +</td>
</tr>
</tbody>
</table>

The last byte only ever contains one digit, the remainder of the byte being filled with the sign (positive or negative).

When setting up a column type of PACKED, you must specify the length and scale of the column. The length represents the exact number of bytes of data. The scale represents the number of digits after the decimal point.

For example, the value 123.45 consists of three bytes of data (as shown above), and has a scale of two.

Notes
- The scale must be less than or equal to \((\text{length} \times 2)-1\)
- Because the last byte can only contain one digit, the precision (number of digits) is calculated as \((\text{length} \times 2)-1\)
- This data type does not support null values. Data Manager replaces null values with zero when writing data of this type.
- You can only read packed data from an EBCDIC file, not an ASCII file.

ZONED

ZONED can only be used in fixed width columns.

This data type interprets decimal numbers with one digit per byte. The final byte also contains the sign (positive or negative).

For example, the value 1234.56 consists of six digits and therefore six bytes.

When setting up a column type of ZONED, you must specify the length and scale of the column. The length represents the exact number of bytes of data. The scale represents the number of digits after the decimal point.

For example, the value 1234.56 consists of six bytes of data, and has a scale of two.

Notes
- The length of a ZONED column can be 1 to 77 and the scale can be 0 to 77. The length must be greater than the scale.
- The precision (number of digits) is assumed to be the length.
This data type does not support null values. Data Manager replaces null values with zero when writing data of this type.

You can read zoned data from an EBCDIC or ASCII file.

### Date String Formats in SQLTXT Files

IBM® Cognos® Data Manager requires that date strings containing text months (for example, Jan, Feb), are in English. If you attempt to import another language, an error appears.

### Error Checking in SQLTXT

The Action on row error feature allows you to specify the action that IBM® Cognos® Data Manager should take if format errors are detected in data rows when reading files in a table during a SELECT operation.

**Note:** Only those columns specified in the SELECT statement have their data validated against the column data type specification. Any other columns included in the SQLTXT definition file are ignored.

The different actions are described below.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Use when</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Immediately abort the current SQLTXT operation and issue an error message.</td>
<td>You want to be sure that the file being read has absolute integrity.</td>
</tr>
<tr>
<td>Warn</td>
<td>Ignore the data row, and continue processing the file. Issue any warnings about this row when processing is complete. A summary warning message, which includes the number of data rows ignored, is always issued if errors are detected. Warnings about individual data rows are issued according to the limit specified. For more information, see &quot;Define a Table Manually&quot; (p. 402).</td>
<td>You want to process the file, but require warning for any potential problems.</td>
</tr>
<tr>
<td>Ignore</td>
<td>Ignore the data row and continue processing the file. No warnings are issued.</td>
<td>You are completely certain of the integrity of the file, or you want to process it irrespective of any errors.</td>
</tr>
</tbody>
</table>

You should take into consideration the following points when deciding which action to use.
### Consideration

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too many columns (delimited files only)</td>
<td>Error detected unless Ignore extra columns is selected. For more information, see &quot;Define a Table Manually&quot; (p. 402).</td>
</tr>
<tr>
<td>Too few columns (delimited files only)</td>
<td>Error detected.</td>
</tr>
<tr>
<td>Value longer than column definition (CHAR only)</td>
<td>Error detected unless Allow truncation is selected. For more information, see &quot;Define a Table Manually&quot; (p. 402).</td>
</tr>
<tr>
<td>Invalid or incorrectly formatted data</td>
<td>Error detected.</td>
</tr>
<tr>
<td>Null values in non nullable fields</td>
<td>Error detected.</td>
</tr>
</tbody>
</table>

For information on using Action on row error, see "Define a Table Manually" (p. 402).

### Null Values in SQLTXT

If a field in a file contains a null value, the way in which this value is handled depends upon the data type and nullability setting of the column in the table definition.

For information about each of the data types, see "SQLTXT Data Types" (p. 389). For information about nullability settings, see "Add Columns to a Table" (p. 403).

<table>
<thead>
<tr>
<th>Data type</th>
<th>Outcome</th>
</tr>
</thead>
</table>
| CHAR                           | For a field containing a value that is empty or contains all spaces (after being trimmed if Remove trailing whitespace from fields is selected):
|                                | • if the column is not nullable, the value is unchanged
|                                | • if the column is nullable, the value is null                                                                                         |
| PACKED and ZONED               | These data types cannot represent null values.                                                                                         |
| All other data types           | If a field is empty or contains all spaces, the value is null.                                                                         |
| DATE                           | For a date field that contains the value 0000-00-00 00:00:00, or a variant of this (depending on the format selected):
|                                | • if the column is not nullable, an error occurs
|                                | • if the column is nullable, the value is null                                                                                         |
### Data type | Outcome
---|---
**TIME** | For a time field that contains the value 00:00:00, or a variant of this (depending on the format selected):
- if the column is not nullable, an error occurs
- if the column is nullable, the value is null

**DATE WITH TIME ZONE** | For a date with time zone field that contains the value 0000-00-00 00:00:00 00:00, or a variant of this (depending on the format selected):
- if the column is not nullable, an error occurs
- if the column is nullable, the value is null

**TIME WITH TIME ZONE** | For a time with time zone field that contains the value 00:00:00 00:00, or a variant of this (depending on the format selected):
- if the column is not nullable, an error occurs
- if the column is nullable, the value is null

**INTERVAL DAY TO SECOND** | For a day to second interval field that contains the value 00:00:00, or a variant of this (depending on the format selected):
- if the column is not nullable, an error occurs
- if the column is nullable, the value is null

**INTERVAL YEAR TO MONTH** | For a year to month interval field that contains the value 0-0, or a variant of this (depending on the format selected):
- if the column is not nullable, an error occurs
- if the column is nullable, the value is null

---

**Change the Font in IBM Cognos Data Manager SQLTXT Designer**

You can change the font used in IBM® Cognos® Data Manager SQLTXT Designer.

**Steps**

1. From the **Tools** menu, click **Options**.

2. Click the appropriate **Browse** button for the setting that you want to change
   - **Application font** applies to the **Tree** pane, and all window and dialog boxes
   - **Fixed width Font** applies to places where a fixed width font is used

The **Select Font** dialog box appears.
3. From the drop down lists, select the appropriate font.
   Tip: If you are changing fixed width fonts, you can restrict the list of fonts to fixed width fonts only by selecting the Fixed width only check box.

4. Click OK to close the Select Font dialog box.
   Tip: To return to the default fonts, click Reset.

5. Click OK.

Open a SQLTXT Database to Access Text Files

You access text files using a SQLTXT database that you define. You must open a database within IBM® Cognos® Data Manager SQLTXT Designer before you can work with its contents. If there is a database open, a message appears informing you that IBM® Cognos® Data Manager will disconnect from the current database.

Steps
1. From the File menu, click Open.
   The Open SQLTXT Definition File dialog box appears.

2. Locate the required database.
   Tip: If you must first create a database, from the File menu, click New.

3. Click Open.

Import a Table Definition using the Import Wizard

IBM® Cognos® Data Manager SQLTXT Designer can read and write
- nondelimited records
- delimited records with either delimited columns, or fixed width columns

You can define a table using the Import wizard, or manually by entering table and column definitions. For information about the manual method, see "Define a Table Manually" (p. 402).

Import Delimited Records with Delimited Columns

In the following example, each column is delimited by a tab character (indicated by a black vertical bar), and each record is delimited by a carriage return (indicated by the vertical gray line on the right).
Steps

1. Open the appropriate SQLTXT database.

2. From the Edit menu, click Import Table Data.

   The Import SQLTXT Table Data File dialog box appears.

3. Select the text file to import, and click Open.

   Tip: You can change the font used in the Import wizard by clicking the Font button.

4. Click Delimited records.

5. In the Record delimiter box, click the delimiter used to separate individual records in the file.

   For information on delimiter types, see "SQLTXT Record Delimiters" (p. 386).

6. In the Number of header rows box, enter the number of header rows in the file.

   IBM® Cognos® Data Manager SQLTXT Designer uses these header rows to name the columns.

7. In the Decimal symbol box, type a single character that is used for the decimal point in numeric values. For example, the value 123.45 uses a period (.) as the decimal point.

8. In the Character set box, enter the encoding used in the text file.

   Tip: You can click an encoding type from the list, or type the code page into the box.

9. In the Digit grouping symbol box, type the character that is used for the thousands separator in numeric values. For example, the value 12,345 uses a comma as the thousands separator.

   You can only specify a single character.

10. In the Preview rows box, enter the number of rows that you want to view in the wizard.

    Tip: Click Refresh Preview to reflect your selections.

11. Click Next.

12. Select Delimited columns.

13. In the Delimiter type box, click the delimiter used to separate the columns in the table.

    For information on column delimiter types, see "SQLTXT Column Delimiters" (p. 387).

14. In the Quote character box, click the quotation character used to enclose individual columns within a record.

    For information on quote characters, see "Quotation Characters in SQLTXT Delimited Records" (p. 388).
15. Click Next.

IBM® Cognos® Data Manager automatically names and attempts to specify the data type of each column. You can override these selections.

16. To rename a column, click the column name and type a new name.

17. To change a data type, double-click the appropriate column, and in the Data Type dialog box, click a different data type.

   If you select NUMBER, you must also specify the number of digits expected after the decimal point in the Scale box.

   If you select a date or time, you must also specify the format in the Format box.

   For information about each of the data types, see "SQLTXT Data Types" (p. 389).

18. Click OK to close the Data type dialog box.

19. Click Process All Records to ensure that all record lengths and data types are correct, and then click Close.

20. Click Finish.

   The wizard adds the table to the database.

   You can now customize the definition file as required. For information, see "Define a Table Manually" (p. 402).

**Import Delimited Records With Fixed Width Columns**

In the following example, each column is of a fixed width, and each record is delimited by a carriage return (indicated by the vertical gray line on the right).

**Steps**

1. Open the appropriate SQLTXT database.

2. From the Edit menu, click Import Table Data.

   The Import SQLTXT Table Data File dialog box appears.

3. Select the text file to import, and click Open.

   Tip: You can change the font used in the Import wizard by clicking the Font button.

4. Click Delimited records.

5. In the Record delimiter box, click the delimiter used to separate individual records in the file.

   For information on delimiter types, see "SQLTXT Record Delimiters" (p. 386).

6. In the Number of header rows box, enter the number of header rows in the file.

   IBM® Cognos® Data Manager SQLTXT Designer uses these header rows to name the columns.

7. In the Decimal symbol box, type a single character that is used for the decimal point in numeric values. For example, the value 123.45 uses a period (.) as the decimal point.
8. In the Character set box, click the encoding used in the text file.

   Tip: You can click an encoding type from the list, or type the code page into the box.

9. In the Digit grouping symbol box, type the character that is used for the thousands separator in numeric values. For example, the value 12,345 uses a comma (,) as the thousands separator.

   You can only specify a single character.

10. In the Preview rows box, enter the number of rows that you want to view in the wizard.

    Tip: Click Refresh Preview to reflect your selections.

11. Click Next.

    Click on the first character in each column to add a delimiter to separate the columns.

    IBM® Cognos® Data Manager adds a vertical bar to indicate the position of the delimiter.

    Tip: You can move a delimiter by dragging it to a new position. To delete a delimiter, click it.

12. Select Fixed width columns.

13. Click on the first character in each column to add a delimiter to separate the columns.

14. Click Next.

    Data Manager automatically names and attempts to specify the data type of each column. You can override these selections.

15. To rename a column, click the column name and type a new name.

16. To change a data type, double-click the appropriate column, and in the Data type dialog box, click a different data type.

    If you select NUMBER, you must also specify the number of digits expected after the decimal point in the Scale box.

    If you select a date or time, you must also specify the format in the Format box.

    For information about each of the data types, see "SQLTXT Data Types" (p. 389).

17. Click OK to close the Data type dialog box.

18. Click Process All Records to ensure that all record lengths and data types are correct. Click Close.

19. Click Finish.

    The wizard adds the table to the database.

    You can now customize the definition file as required. For information, see "Define a Table Manually" (p. 402).
Import Nondelimited Records

The following example shows a nondelimited record which is a single string that does not have delimiters or fixed columns.

```
1996015DFO5  161996015C5D01  35  19
10 22930199601WVF041C0  8763199601WVF863
416996018920E82  2364199602M920  49
3 80 371199601HX02  28  175199601KGF
1114199601BX04  57  1193199601H4095  71
```

**Steps**

1. Open the appropriate SQLTXT database.
2. From the **Edit** menu, click **Import Table Data**.
   
   The **Import SQLTXT Table Data File** dialog box appears.
3. Select the text file to import, and click **Open**.
   
   **Tip:** You can change the font used in the Import wizard by clicking the **Font** button.
4. Click **Non-delimited records**.
5. In the **Record length** box, type the number of characters in the record.
6. In the **Number of header rows** box, enter the number of header rows in the file.
7. In the **Decimal symbol** box, type a single character that is used for the decimal point in numeric values. For example, the value 123.45 uses a period (.) as the decimal point.
8. In the **Character set** box, click the encoding used in the text file.
   
   **Tip:** You can click an encoding type from the list, or type the code page into the box.
9. In the **Digit grouping symbol** box, type the character that is used for the thousands separator in numeric values. For example, the value 12,345 uses a comma (,) as the thousands separator.
   
   You can only specify a single character.
10. In the **Preview rows** box, enter the number of rows that you want to view in the wizard.
    
    **Tip:** Click **Refresh Preview** to reflect your selections.
11. Click **Next**.
12. Click on the first character in each column to add a delimiter to separate the columns.

   **IBM® Cognos® Data Manager** adds a vertical bar to indicate the position of the delimiter.

   **Tip:** You can move a delimiter by dragging it to a new position. To delete a delimiter, click it.
13. Click **Next**.
Data Manager automatically names and attempts to specify the data type of each column. You can override these selections.

14. To rename a column, click the column name and type a new name.

15. To change a data type, double-click the appropriate column, and in the **Data type** dialog box, click a different data type.
   
   If you select NUMBER, you must also specify the number of digits expected after the decimal point in the **Scale** box.

   If you select a date or time, you must also specify the format in the **Format** box.

   For information about each of the data types, see "SQLTXT Data Types" (p. 389).

16. Click **OK** to close the **Data type** dialog box.

17. Click **Process All Records** if you want to ensure that all record lengths and data types are correct. Click **Close**.

18. Click **Finish**.

   The wizard adds the table to the database.

   You can now customize the definition file as required. For information, see "Define a Table Manually" (p. 402).

### Define a Table Manually

Instead of using the Import wizard, you can define a table manually by adding an empty table, selecting the file that holds the data, and then adding the required columns.

#### Steps

1. Open the appropriate SQLTXT database.

2. Click the **Database** icon.

3. From the **Edit** menu, click **Insert Table**.

   An empty table appears in the tree.

4. Click the new table and, from the **Edit** menu, click **Rename**.

5. Type the new name for the table.

6. In the **Data file** box, enter the name of the text file.

7. In the **Header lines** box, type the number of header rows in the file.

8. In the **Trailer lines** box, type the number of trailer lines in the file.

   SQLTXT ignores these rows when it accesses data in the file.

9. In the **Character set** box, enter the encoding used in the text file.

   **Tip:** You can click an encoding type from the list, or type the code page into the box.
10. In the **Action on row error** box, click the action to apply when faulty data is read.
   For information, see "Error Checking in SQLTXT" (p. 394).

11. If you selected **Warn** in the **Action on row error** box, specify the maximum number of warnings to report in **Limit** box.
   **Tip:** For unlimited warnings, set the limit to -1.

12. Select or clear the **Remove trailing whitespace from fields** check box as appropriate.

13. Specify whether columns in the input text file are of fixed length or delimited by clicking either **Fixed length records** or **Delimited records**.

14. In the **Record delimiter** box, click the delimiter used to separate individual records in the file.
   For information on the delimiter types, see "SQLTXT Record Delimiters" (p. 386).

15. If you selected **Delimited records**, in the **Column delimiter** box, specify the delimiter used to separate the columns in the table.
   For information on the delimiter types, see "SQLTXT Column Delimiters" (p. 387).

16. If you selected **Delimited records**, in the **Quote character** box, click the quotation character used to enclose individual columns within a record.
   For information on quote characters, see "Quotation Characters in SQLTXT Delimited Records" (p. 388).

17. If you selected **Delimited records**, select the **Ignore extra columns** check box to ignore any excess fields in a single record.
   **Note:** SQLTXT does not perform any error checks on the excess columns.

18. If you selected **Delimited records**, select the **Allow truncation** check box to truncate data in a column that exceeds the specified column length.

19. In the **Decimal symbol** box, type a single character that is used for the decimal point in numeric values. For example, the value 123.45 uses a period (.) as the decimal point.

20. In the **Digit grouping symbol** box, type the character that is used for the thousands separator in numeric values. For example, the value 12,345 uses a comma (,) as the thousands separator.
   You can only specify a single character.

---

**Add Columns to a Table**

This section describes how to add columns to a table.

**Steps**

1. Click the table to which you want to add a column.

2. From the **Edit** menu, click **Insert Column**.
   A new column appears beneath the selected table.
3. In the **Data type** box, click the data type for the column.
   
   If you have specified that columns are delimited and you select
   - BINARY, BLOB, CHAR, or CLOB, enter the maximum column width found in the text file in the **Length** box
   - a date or time, enter the format of the date in the **Format** box
   
   If you have specified that columns are of fixed width and you select
   - NUMBER, PACKED or ZONED, enter the required decimal value in the **Length.Scale** box
   - a date or time, enter the format of the date in the **Format** box
   - any other data type, enter the required value in the **Length** box

   For information about each of the data types, see "SQLTXT Data Types" (p. 389).

4. If the column can accept null values, select the **Nullable** check box.

5. If the column forms part of the primary key of the table, select the **Key** check box.

### Create a Similar Table

If you want to create a table that is similar to an existing table, duplicate the table and then make the required changes. This is often quicker and simpler than creating a completely new table.

**Steps**

1. Click the table to duplicate.

2. From the **Edit** menu, click **Duplicate Table**.

   IBM® Cognos® Data Manager SQLTXT Designer duplicates the table and assigns a unique name to it. The columns in the original table are also duplicated, but the original column names are retained.

### Edit a Column or a Table

You can edit an existing column or table.

**Steps**

1. Click the table or column that you want to edit.

2. In the right pane, edit the values as required.

### Delete a Column or a Table

You can delete a column or table that is no longer required.

**Steps**

1. Click the table or column that you want to delete.
2. From the Edit menu, click Delete.

3. In the message box, click Yes.

**Use Named Pipes to Access Data**
You can read data from, and write data to, pipes using SQLTXT. When reading data, IBM® Cognos® Data Manager continues to read from the pipe until it is closed, or an end-of-file marker (EOF) is read.

**Define a Pipe using IBM Cognos Data Manager SQLTXT Designer**
To define a pipe in IBM® Cognos® Data Manager SQLTXT Designer, you must manually create a SQLTXT definition file (.def). For more information, see "Define a Table Manually" (p. 402).

The definition file must include the name of the pipe in the Data file box. The syntax for pipes in SQLTXT is

"[PIPE!]/<filename>"

Although it is preferable to always include the pipe syntax, SQLTXT uses named pipes by default in the following circumstances:

- In the Microsoft® Windows® operating system, if the specified file name takes the form
  - 
  - 
  - "//./pipe/<name>" for a local pipe
  - "//<machine_name>/pipe/<name>" for a network pipe

- In the UNIX® operating system, if the specified file name already exists, and it is a pipe (FIFO) device.

It is also possible to specify a pipe, using the pipe syntax, where the file name does not include any directory information. For example:

"PIPE!test"

In Windows®, SQLTXT assumes that this is a named pipe on a local machine. Using the example above, this would be

"PIPE!//./pipe/test"

In UNIX®, SQLTXT assumes that this is a named pipe located in the same directory as the SQLTXT definition file.

**Notes**
- Header lines are supported when reading pipes, and ignored when writing pipes.
- Trailer lines are not supported, and cause a run-time error if specified. Therefore, you should ensure that the value in the Trailer lines box is always set to zero.
Example SQLTXT Definition Files for Pipes

The following is an example SQLTXT definition file for a Microsoft® Windows® operating system pipe.

[Input]
FILENAME '//./pipe/input'
VERSION 8
CHARSET 'GB2312'
HEADERLINES 4
TRAILERLINES 0
ROWERROR ERROR
QUOTECHAR DOUBLE
DECIMALSEP '.
THOUSANDSEP NONE
LINEDELMITER CR
RECORD TAB
COLUMN 'col1' CHAR 3 'NOT NULL' 'NOT KEY'
COLUMN 'col2' CHAR 3 'NOT NULL' 'NOT KEY'

The following is an example SQLTXT definition file for a UNIX® operating system pipe.

FILENAME 'pipe!/output'
VERSION 8
CHARSET 'ASCII'
HEADERLINES 1
TRAILERLINES 0
ROWERROR ERROR
QUOTECHAR DOUBLE
DECIMALSEP '
THOUSANDSEP ','
LINEDELMITER CR
RECORD TAB
COLUMN 'C1' INTEGER 10 'NOT NULL' 'NOT KEY'
COLUMN 'C2' CHAR 33 'NOT NULL' 'NOT KEY'

Create an External Pipe

IBM® Cognos® Data Manager can read data from pipes that have been created by external applications. When you create an external pipe, there are certain restrictions that you should be aware of. Data Manager creates named pipes using the CreateNamedPipe() Microsoft® Windows® operating system API function with the following parameter settings:

- PIPE_ACCESS_DUPLEX is used for the dwOpenMode parameter. If the client end uses the SetNamedPipeHandleState() Windows® API function on the pipe handle, it requires GENERIC_WRITE access to the named pipe.

  Note: When Data Manager is the client, it will use the SetNamedPipeHandleState() function if the pipe handle is not in PIPE_READMODE_BYTE state, therefore external programs creating pipes for Data Manager should use PIPE_ACCESS_DUPLEX.
PIPE_TYPE_MESSAGE, PIPE_READMODE_BYTE, and PIPE_WAIT are used for the dwPipeMode parameter.

Data Manager writes an explicit EOF to the pipe when it has sent all the data. This is done by flushing the pipe, writing a zero byte message and then flushing the pipe again before disconnecting, as shown by the following C language code sample:

```c
FlushFileBuffers( hPipe );
WriteFile( hPipe, (LPVOID)NULL, (DWORD)0, &dwWritten, (LPOVERLAPPED)NULL);
FlushFileBuffers( hPipe );
DisconnectNamedPipe( hPipe );
```

If Data Manager is acting as a client end of a named pipe (using SELECT to obtain data from it), it detects an EOF either by a successful call to ReadFile() when a zero byte read is returned, or when ReadFile() returns an error and the last error code is ERROR_NO_DATA.

**Test Pipe Access**

If you are reading data from a pipe, you can test the pipe access using SQLTerm or by executing the DataStream used to read the data.

For information on using SQLTerm, see "SQLTerm and SQL Helper" (p. 55). For information on executing a DataStream, see "Execute a DataStream" (p. 160).

When you execute the SQL query or DataStream, if the pipe cannot be found, there is a short delay while IBM® Cognos® Data Manager waits for a pipe to be created. During this time it is not possible to interrupt the execution. If no pipe is created, Data Manager informs you after the delay period times out.

**Note:** By default, the delay timeout period is 30 seconds. You can change this value by setting the DS_PIPE_CREATE_WAIT variable.

**SQL Support for SQLTXT Files**

A SQLTXT database supports a limited level of SQL. It is possible to select, filter, and deliver data, to and from SQLTXT files.

Where possible, SQLTXT supports the functions and operators that are available in IBM® Cognos® Data Manager. Some functions, such as member functions, are meaningless outside the context of a Data Manager build and SQLTXT does not support them.

For information about Data Manager functions and operators, see the IBM® Cognos® Function and Scripting Reference Guide.

**Expressions in VALUES Clauses**

SQLTXT does not support expressions within VALUES clauses of INSERT statements, such as the following:

```sql
INSERT INTO progress(timestamp, message)
VALUES (datetimenow(), 'Starting')
```
However, you can use a dummy table, which has a single data row, to avoid using a VALUES clause.

**Steps to Use a Dummy Table**

1. Create a dummy table, for example
   
   ```sql
   CREATE TABLE dummy (dummy CHAR(1));
   ```

2. Into the dummy table, insert a single data row, for example
   
   ```sql
   INSERT INTO dummy VALUES ('1');
   ```

3. Insert the required data into the table; for example
   
   ```sql
   INSERT INTO progress(timestamp, message)
   SELECT datetimenow(), 'Starting'
   FROM dummy;
   ```

**Examples**

These are examples of SQL statements supported by IBM® Cognos® Data Manager SQLTXT Designer.

**SELECT Statements**

- **SELECT DISTINCT * FROM sales;**

  This example retrieves all data from the table named sales and removes any duplicate data rows.

- **SELECT prod_cd, unit_price*units_sold FROM sales WHERE prod_cd LIKE 'aa%';**

  This example retrieves prod_cd and calculates the corresponding product of unit_price and units_sold for all data rows in the table named sales that have a prod_cd value starting with aa.

- **SELECT DISTINCT * FROM product WHERE unit_price BETWEEN 1.0 AND 2.5;**

  This example removes all duplicate data rows, retrieves all data from the table named product where the unit_price value is between 1.0 and 2.5 inclusive.

- **SELECT DISTINCT prod_cd, unit_price FROM sales WHERE sales.prod_cd IN ('S01', 'S03', 'S05');**

  This example removes all duplicate data rows, retrieves the prod_cd and unit_price from all data rows in the table named sales where prod_cd is one of S01, S02, or S03.
**INSERT Statements**

- INSERT INTO product (prod_cd, name, unit_price)
  VALUES ('p001', 'Widget', 2.25);

  This example inserts into the table named product a single data row having the value called prod_cd equal to 'p001', the value named name value equal to 'Widget', and the value named unit_price equal to 2.25. Any other fields contain null.

- INSERT INTO sales_copy SELECT * FROM sales;

  This example copies all rows from the table named sales to the table named sales_copy.

**DELETE Statements**

- DELETE FROM sales WHERE prod_cd='p001';

  This example deletes all rows from the table named sales where prod_cd is 'p001'.

- DELETE FROM sales;

  This example deletes all rows from the table named sales.

**CREATE TABLE Statements**

- CREATE TABLE product (
  prod_cd CHAR(3) 'NOT NULL' 'KEY',
  name CHAR(16) 'NULL' 'NOT KEY',
  unit_price FLOAT 'NULL' 'NOT KEY');

  This example creates a table named product that has a column named prod_cd of type CHAR with length 3 that is the primary key of the table, a column named name of type CHAR with length 16, and column named unit_price of type FLOAT.

**DROP TABLE Statements**

- DROP TABLE product;

  This example removes the table named product from the database and deletes the associated text file.

**Special Columns**

ROWNUM, FILEROWNUM, and FILENAME are special, read-only columns automatically added to SQLTXT data tables. ROWNUM returns the ordinal number of the row within the returned data; FILEROWNUM returns the line number of the row within the entire table; FILENAME returns the name of the file containing the current row.

For example, this is a Sales table.

<table>
<thead>
<tr>
<th>Period</th>
<th>Location</th>
<th>Product</th>
<th>Sales</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>200001</td>
<td>SD</td>
<td>F05</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
This SQL statement can be run against the Sales table:

```
SELECT ROWNUM, FILEROWNUM, Location, Product
FROM Sales
WHERE Product LIKE 'K%'
```

<table>
<thead>
<tr>
<th>ROWNUM</th>
<th>FILEROWNUM</th>
<th>Location</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>SD</td>
<td>K05</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>SD</td>
<td>K04</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>SD</td>
<td>K01</td>
</tr>
</tbody>
</table>

**Generate Test Data**

MAKEROWS is a virtual SQLTXT database that IBM® Cognos® Data Manager provides to generate test data. It does not have a corresponding definition file.

To generate test data, write a special type of SELECT statement with the following where `<expression>` is a literal value or a function, and `<number_of_rows>` is the number of rows to generate.

```
SELECT [DISTINCT] <expression> {,<expression>} FROM <number_of_rows>;  
```

Because the generated rows may contain duplicates, the number of returned rows might be less than this number if you use the DISTINCT option.

**Connect to the MAKEROWS Database**

You can connect to the MAKEROWS database from IBM® Cognos® Data Manager Designer, from the Data Manager language, and from SQLTerm.

**Steps to Connect in the Designer**

1. Use the SQLTXT DBMS driver to create a connection to the MAKEROWS database.

---

### Table: Sales Data

<table>
<thead>
<tr>
<th>Period</th>
<th>Location</th>
<th>Product</th>
<th>Sales</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>200001</td>
<td>SC</td>
<td>S01</td>
<td>35</td>
<td>191</td>
</tr>
<tr>
<td>200001</td>
<td>SD</td>
<td>S03</td>
<td>71</td>
<td>538</td>
</tr>
<tr>
<td>200001</td>
<td>SD</td>
<td>F02</td>
<td>1</td>
<td>289</td>
</tr>
<tr>
<td>200001</td>
<td>SD</td>
<td>S01</td>
<td>17</td>
<td>95</td>
</tr>
</tbody>
</table>
2. Replace the name of the definition file with MAKEROWS.

**Step to Connect in the Command Line**

- Create an alias for the MAKEROWS database by using this syntax where <runenv> is the run environment context for the alias, and <aliasname> is your name for the alias.

  `<runenv> <aliasname> SQLTXT 'MAKEROWS'`

**Step to Connect in SQLTerm**

- Use the following command:

  `CONNECT SQLTXT MAKEROWS`

**Examples of Generating Rows of Test Data**

These examples generate rows of test data:

- `SELECT 'sample' FROM 1000;`
  
  This example generates 1,000 rows of data, each containing a single column having the text value, 'sample'.

- `SELECT RAND(1,1000,1), 'sample' FROM 1000;`
  
  This example generates 1,000 rows of data, each containing two columns. The first column contains a random number between 1 and 1,000. The second column contains the text value, 'sample'.

- `SELECT DISTINCT RAND(1,10000,1), 'sample' FROM 1000;`
  
  This is the same as the previous example except that IBM® Cognos® Data Manager removes duplicate data rows from the output. Thus, the number of output data rows might be less than 1,000.
# Chapter 31: Commands

The following table briefly describes all the IBM® Cognos® Data Manager commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>databuild</td>
<td>Executes the fact build that you specify in command line parameters, lists available delivery modules, or lists delivery module properties.</td>
</tr>
<tr>
<td>dimbuild</td>
<td>Executes the dimension build that you specify in command line parameters.</td>
</tr>
<tr>
<td>rundsjob</td>
<td>Executes a JobStream.</td>
</tr>
<tr>
<td>dsremote</td>
<td>Executes a fact build, dimension build, or JobStream that is located on a remote computer.</td>
</tr>
<tr>
<td>catimp</td>
<td>Imports a component from a text file to a Data Manager catalog.</td>
</tr>
<tr>
<td>catexp</td>
<td>Exports a component from a Data Manager catalog to a text file.</td>
</tr>
<tr>
<td>catlist</td>
<td>Lists the contents of a Data Manager catalog.</td>
</tr>
<tr>
<td>catbackup</td>
<td>Performs a backup of a Data Manager catalog to a text file.</td>
</tr>
<tr>
<td>catrestore</td>
<td>Restores a backup of a Data Manager catalog from a text file.</td>
</tr>
<tr>
<td>catupgrade</td>
<td>Upgrades a Data Manager catalog.</td>
</tr>
<tr>
<td>catmanage</td>
<td>Manages a Data Manager catalog.</td>
</tr>
<tr>
<td>catdoc</td>
<td>This command documents the contents of a catalog.</td>
</tr>
<tr>
<td>showref</td>
<td>Lists the members of a reference structure.</td>
</tr>
<tr>
<td>dstofm</td>
<td>Exports a metadata collection to an XML file. This file can then be used to produce a model of your data mart or data warehouse in IBM® Cognos® Framework Manager, without the need to recreate the complete model.</td>
</tr>
<tr>
<td>sqlterm</td>
<td>Starts the SQLTerm application.</td>
</tr>
</tbody>
</table>
Command | Description
--- | ---
catpkgtoxml | Converts a Data Manager package file to a data movement specification XML file.

tip: When using parameters that contain embedded spaces, you may need to enclose them in double quotation marks. For example, "dsn=global sales".

databuild

This command executes a fact build that you specify in command line parameters, lists available delivery modules, or lists the properties for the specified delivery module.

Execute a Fact Build

Syntax
databuild [<catdb_driver> <catdb_items>] <databuild_name or uuid> [options]

When you run databuild with this syntax, it executes the named fact build that resides in the named catalog or package (specified using the option -p).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver&gt;</td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td>&lt;catdb_items&gt;</td>
<td>Specifies the connection string to pass to the database driver.</td>
</tr>
<tr>
<td>&lt;databuild_name&gt;</td>
<td>Specifies the name of the fact build to execute within the catalog or package.</td>
</tr>
<tr>
<td>&lt;uuid&gt;</td>
<td>Specifies the universally unique identifier for the component to execute.</td>
</tr>
<tr>
<td>&lt;options&gt;</td>
<td>For information, see &quot;databuild Options&quot; (p. 416).</td>
</tr>
</tbody>
</table>

Example
databuild ODBC "dsn=salescat;uid=sales;pwd=dm123" US_Sales

This command runs databuild to execute the fact build named US_Sales. To retrieve the build definition, databuild connects to the ODBC data source named SalesCat as user Sales, with password dm123.
databuild -p"c:\data files\sales.pkg" US_Sales
This example runs databuild to execute a fact build named US_Sales from the package named sales.pkg.

**List All Delivery Modules**

**Syntax**

databuild -D

When you run databuild with this syntax, it returns a list of all the delivery modules to which you have access.

**Example**

databuild -D

Available Delivery Modules are:

```
FACT Delivery
-------------
DB2LOAD  -  DB2 LOAD
ESSDIRECT -  Essbase Direct Feed
ESSFILE  -  Essbase File Feed
INXLOAD  -  Informix LOAD
SSLLOAD  -  Microsoft SQL Server BCP Command
ORALOAD  -  ORACLE SQL*Loader
RBLOAD   -  Red Brick Loader (TMU)
TABLE    -  Relational Table
BCPLOAD  -  SQL Server Bulk Copy via API
FASTLOAD -  Teradata Fastload
MULTILOAD -  Teradata Multiload
TPUMP    -  Teradata TPump
TEXT     -  Text File
```

DIMENSION Delivery

```
STAR -  Relational Table
```

For more information on delivery modules, see "Fact Delivery Module Properties" (p. 455).

**List the Properties of a Delivery Module**

**Syntax**

databuild -D<module>

When you run databuild with this syntax, it returns a list of all the properties for the delivery module that you specify on the command line.
Specifies the name of the delivery module for which you want to list the properties.

For more information on delivery modules, see "Fact Delivery Module Properties" (p. 455).

data build Options

Syntax
<options> ::= 
{-V<name>[=<value>]}[-X[F][D]

Symbol | Description
--- | ---
<module> | Specifies the name of the delivery module for which you want to list the properties.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e</td>
<td>Denotes the run environment for the fact build.</td>
</tr>
<tr>
<td>&lt;runenv&gt;</td>
<td>Specifies the run environment to apply to identify the correct database alias.</td>
</tr>
<tr>
<td>-p</td>
<td>Denotes that the fact build is executed from a package.</td>
</tr>
<tr>
<td>&lt;package_file&gt;</td>
<td>Specifies the name and path of the package file.</td>
</tr>
<tr>
<td>Note:</td>
<td>If you also specify catalog details, then logging and audit information is stored in the catalog, even if the catalog has no connection with the package.</td>
</tr>
<tr>
<td>-A</td>
<td>Denotes the text file that contains the database alias definitions to use for the fact build.</td>
</tr>
<tr>
<td>&lt;dbalias_file&gt;</td>
<td>Specifies the name and path of the file that contains the database alias definitions to use for the fact build.</td>
</tr>
<tr>
<td>-C</td>
<td>Executes the fact build in check only mode.</td>
</tr>
<tr>
<td>-D</td>
<td>Lists the available delivery modules, or the properties of the specified delivery module.</td>
</tr>
<tr>
<td>&lt;module_name&gt;</td>
<td>Specifies the name of the delivery module for which you want to list the properties.</td>
</tr>
<tr>
<td>-L</td>
<td>Logs progress to the specified file. You can specify the file name using the &lt;logfile&gt; argument.</td>
</tr>
</tbody>
</table>
Specifies the full path and name of the text file to which IBM® Cognos® Data Manager should log progress.

-O  Executes the build in object creation mode.

-P  Specifies that at the end of a build, a keystroke is required before terminating.

-T  Denotes the path and name for expression and script tracing log files.

<tracefile_root>  Specifies the path and name to use for the expression and script tracing log files.

For more information, see "Create a Log File for Expression and Script Tracing" (p. 292).

-V  Denotes that the parameter is a named variable and value pair.

<name>  Specifies the variable to use during item substitution.

<value>  Specifies the value that the corresponding variable is to use.

-X  Specifies the delivery type:
  - F delivers data using fact deliveries
  - D delivers data using dimension deliveries

**dimbuild**

Executes the dimension build that you specify in the command line parameters.

**Syntax**

dimbuild [<catdb_driver> <catdb_items>] <dimbuild_name> [options]

When you run dimbuild with this syntax, it executes the named dimension build that resides in the named catalog or package (specified using the option -p).
**Example**

```plaintext
dimbuilt ODBC "DSN=SalesCat;UID=Sales;PWD=ds123" Period
```

This command runs dimbuild to execute the dimension build named Period. To retrieve the dimension build definition, dimbuild connects to the ODBC data source named SalesCat as user Sales, with password ds123.

```plaintext
dimbuilt -p"c:\data files\sales.pkg" Period
```

This example runs dimbuild to execute a dimension build named Period from the package named sales.pkg.

**dimbuilt Options**

**Syntax**

```plaintext
<options> ::= [-e<runenv>] [-p<package_file>] [-A<dbalias_file>] [-C]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e</td>
<td>Denotes the run environment for the dimension build.</td>
</tr>
<tr>
<td>&lt;runenv&gt;</td>
<td>Specifies the run environment to apply to identify the correct database alias.</td>
</tr>
<tr>
<td>-p</td>
<td>Denotes that the dimension build is executed from a package.</td>
</tr>
<tr>
<td>&lt;package_file&gt;</td>
<td>Specifies the name and path of the package file.</td>
</tr>
</tbody>
</table>

**Note:** If you also specify catalog details, then logging and audit information is stored in the catalog, even if the catalog has no connection with the package.
### rundsjob

This command executes a JobStream.

**Syntax**

```
rundsjob [<catdb_driver> <catdb_items>] <jobstream_name or uuid> [options]
```

When you run rundsjob with this syntax, it executes the named JobStream that resides in the named catalog or package file (specified using the option `-p`).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;catdb_driver&gt;</code></td>
<td>Specifies the database driver to use to connect to the catalog containing the JobStream.</td>
</tr>
</tbody>
</table>
Chapter 31: Commands

**Example**

```bash
rundsjob ODBC "DSN=Sales;UID=admin" example_jobstream
-R
```

This example runs `rundsjob` to restart the JobStream named `example_jobstream`. The IBM® Cognos® Data Manager catalog resides in the ODBC data source named Sales, to which Data Manager connects as user admin with no password.

```bash
rundsjob -p"c:\data files\sample.pkg" example_jobstream
```

This example runs `rundsjob` to execute a JobStream named `example_jobstream` from the package named `sample.pkg`.

### rundsjob Options

**Syntax**

```bash
<options> ::= 
[-T<tracefile_root>] [-V<name>[=<value>]]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e</td>
<td>Denotes the run environment for the JobStream.</td>
</tr>
<tr>
<td>&lt;runenv&gt;</td>
<td>Specifies the run environment to apply to identify the correct database alias.</td>
</tr>
<tr>
<td>-p</td>
<td>Denotes that the JobStream is executed from a package.</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;package_file&gt;</td>
<td>Specifies the name and path of the package file.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you also specify catalog details, then logging and audit information is stored in the catalog, even if the catalog has no connection with the package.</td>
</tr>
<tr>
<td>-A</td>
<td>Denotes the text file that contains the database alias definitions to use for the JobStream.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot combine this option with -D (execute build nodes as individual data movement tasks).</td>
</tr>
<tr>
<td>&lt;dbalias_file&gt;</td>
<td>Specifies the name and path of the text file that contains database alias definitions to use for the build.</td>
</tr>
<tr>
<td>-C</td>
<td>Executes the JobStream in check only mode.</td>
</tr>
<tr>
<td>-D</td>
<td>Executes fact build and dimension build nodes within the JobStream as individual data movement tasks. For more information, see &quot;Configure a JobStream Execution to Use IBM Cognos Business Intelligence Load Balancing&quot; (p. 281).</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> You cannot combine this option with -A (local external alias files).</td>
</tr>
<tr>
<td>-L</td>
<td>Logs progress to the specified file. You can specify the file name using the &lt;logfile&gt; argument.</td>
</tr>
<tr>
<td>&lt;logfile&gt;</td>
<td>Specifies the full path and name of the text file to which IBM Cognos Data Manager should log progress.</td>
</tr>
<tr>
<td>-M</td>
<td>By default, user messages can only be included in the log file for the JobStream node that is being executed. Using this flag, you can include user messages for a JobStream node in the log file of the master JobStream.</td>
</tr>
<tr>
<td></td>
<td>You must also specify USER values in the TRACE_VALUES variable for both the JobStream node and the master JobStream. For information, see &quot;TRACE_VALUES&quot; (p. 331).</td>
</tr>
<tr>
<td>-N</td>
<td>Restricts the number of processes that a JobStream can execute concurrently.</td>
</tr>
</tbody>
</table>
### dsremote

This command executes a fact build, dimension build, or JobStream that is located on a remote computer.

To use this command, you must have the IBM® Cognos® Data Manager Network Services server installed and configured on the remote Data Manager server.

For more information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

**Syntax**

```
dsremote [-n] [-P] <host> <command> {<command_arg>}
```

- `<command>` ::= rundsjob | databuild | dimbuild

### Symbol | Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;maxprocesses&gt;</code></td>
<td>Specifies the number of processes that a JobStream can execute concurrently. Any subsequent master JobStream nodes are suspended until the process count for the currently executing JobStream falls below the <code>&lt;maxprocesses&gt;</code> value.</td>
</tr>
<tr>
<td><code>-P</code></td>
<td>Specifies that at the end of a JobStream, a keystroke is required before terminating.</td>
</tr>
<tr>
<td><code>-O</code></td>
<td>Executes builds in the JobStream in object creation mode.</td>
</tr>
<tr>
<td><code>-R</code></td>
<td>Restarts executing the JobStream at the point of a previous interruption. The JobStream restarts at the earliest node that did not successfully complete.</td>
</tr>
<tr>
<td><code>-T</code></td>
<td>Denotes the path and name for expression and script tracing log files.</td>
</tr>
<tr>
<td><code>&lt;tracefile_root&gt;</code></td>
<td>Specifies the path and name to use for the expression and script tracing log files. For more information, see &quot;Create a Log File for Expression and Script Tracing&quot; (p. 292).</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Denotes that the parameter is a named variable and value pair.</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td>Specifies the name of the variable to use during item substitution.</td>
</tr>
<tr>
<td><code>&lt;value&gt;</code></td>
<td>Specifies the value that the corresponding named variable takes.</td>
</tr>
</tbody>
</table>
**Symbol** | **Description**
---|---
-n | Specifies that the client should not wait for the output of the remote command, but submit the command and then exit.
-P | Specifies that when remote execution is complete, a keystroke is required before terminating.
-host> | The computer on which remote execution is to take place.
<command> | Specifies the type of component that dsremote is to execute:
  - rundsjob executes a JobStream
  - databuild executes a data build
  - dimbuild executes a dimension build
<command_arg> | Specifies the arguments for the type of component that you specified in <command>.

For more information, see "databuild" (p. 414), "dimbuild" (p. 417), or "rundsjob" (p. 419).

**Example**

dremote Europe dimbuild ODBC DSN=DS_Tutorial Product

This command runs dsremote to execute the dimension build named Product on a remote computer named Europe. To retrieve the dimension build definition, databuild connects to the ODBC data source named DS_Tutorial that is defined on the remote computer.

**catlist**

This command lists the contents of an IBM® Cognos® Data Manager catalog. By default, catlist lists all database aliases, fact builds, dimension builds, and reference structures. However, you can list only one type of catalog component (or a single catalog component) by using the appropriate option.

**Syntax**

```plaintext```
catlist <catdb_driver_name> <catdb_driver_items>  
[[<component_type>]]  [[<component_namecomponent_uuid>]]  
-o<filename>] [-I[C][N][U]] [-P] [-V<name>[=value>]]
```

```
<component_type>::=  
A | B | D | F | H | J | K | L | M | S | T
```

User Guide 423
**Symbol** | **Description**
--- | ---
<catdb_driver_name> | Specifies the database driver to use to connect to the catalog.
<catdb_driver_items> | Specifies the connection string to pass to the database driver. For example, to connect as user DBA, with password PASS, to a catalog in an ODBC data source named Sample, you would specify "DSN=Sample;UID=DBA;PWD=PASS"
<component_type> | Specifies the type of component that catlist is to list:
  - A specifies a database alias
  - B specifies a fact build
  - D specifies a reference dimension
  - F specifies a user-defined function
  - H specifies a hierarchy
  - J specifies a JobStream
  - K specifies a metadata conformed dimension
  - L specifies a lookup
  - M specifies a metadata collection
  - S specifies a dimension build
  - T specifies a template
<component_name> | Specifies the name of the individual component to list.
<component_uuid> | Specifies the universally unique identifier for the component to list.
-o | Denotes that the parameter is the name of a text file to which Data Manager should write the output.
<filename> | Specifies the name and path of the text file to which Data Manager should write the output.
-I | Shows additional information about the specified component:
  - C shows the business name
  - N shows the description
  - U shows the universally unique identifier
This command exports a catalog item to an IBM® Cognos® Data Manager file-based specification.

Syntax

catexp <catdb_driver_name> <catdb_driver_items> <export_filename> <component_type> [ <component_nameuuid> ] [-a] [-L<logfile>] [-P] [-V<name>=[<value>]]

<component_type> ::= A | B | D | F | H | J | K | L | M | S | T

This command exports a catalog item to an IBM® Cognos® Data Manager file-based specification.

Syntax

catexp <catdb_driver_name> <catdb_driver_items> <export_filename> <component_type> [ <component_nameuuid> ] [-a] [-L<logfile>] [-P] [-V<name>=[<value>]]

<component_type> ::= A | B | D | F | H | J | K | L | M | S | T

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver_name&gt;</td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td>&lt;catdb_driver_items&gt;</td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog.</td>
</tr>
<tr>
<td></td>
<td>For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify</td>
</tr>
<tr>
<td></td>
<td>&quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td>&lt;export_filename&gt;</td>
<td>Specifies the path and name of the file to which you want to export the catalog component.</td>
</tr>
<tr>
<td>Symbol or option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| <component_type> | Specifies the type of component to export:  
  - A specifies a database alias  
  - B specifies a fact build  
  - D specifies a reference dimension  
  - F specifies a user-defined function  
  - H specifies a hierarchy  
  - J specifies a JobStream  
  - K specifies a metadata conformed dimension  
  - L specifies a lookup  
  - M specifies a metadata collection  
  - S specifies a dimension build  
  - T specifies a template |
| <component_name> | The name of the individual component to export. |
| <uuid>          | Specifies the universally unique identifier for the component to export. |
| -a              | Appends the exported component to the named file. |
| -L              | Logs progress to the specified file. You can specify the file name using the <logfile> argument or you can use the following syntax:  
  - Microsoft® Windows® operating system syntax  
    -L %<variable_name>\<file_name>  
  - UNIX® operating system syntax  
    -L $<variable_name>/<file_name> |
| <logfile>       | Specifies the name and path of the text file to which Data Manager should log progress. |
| -P              | Specifies that at the end of the export, a keystroke is required before terminating. |
| -V              | Denotes that the parameter is a named variable and value pair. |
| <name>          | Specifies the name of a variable. |
| <value>         | Specifies the value that the associated variable takes. |
Notes

- If you export an object to a file in which that object already exists, the old and new definitions may conflict to produce an invalid definition. To avoid this, edit the target text file to remove the redundant definition before exporting the new definition to the file.

- Data Manager cannot import from definition files that contain more than one fact build or dimension build. To avoid this, always specify the component name or universally unique identifier when you export a fact build or dimension build.

**catimp**

This command imports a component from a text file to an IBM® Cognos® Data Manager catalog.

**Syntax**

```
```

```
<filetype> ::= B | D | F | J | K | M | R | S
```

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver_name&gt;</td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td>&lt;catdb_driver_items&gt;</td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog. For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td>&lt;import_filename&gt;</td>
<td>Specifies the path and name of the text file that defines the component.</td>
</tr>
<tr>
<td>Symbol or option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>&lt;filetype&gt;</td>
<td>Specifies the type of component to import:</td>
</tr>
<tr>
<td></td>
<td>• B specifies a fact build</td>
</tr>
<tr>
<td></td>
<td>• D specifies a database alias</td>
</tr>
<tr>
<td></td>
<td>• F specifies a user-defined function</td>
</tr>
<tr>
<td></td>
<td>• J specifies a JobStream</td>
</tr>
<tr>
<td></td>
<td>• K specifies a metadata conformed dimension</td>
</tr>
<tr>
<td></td>
<td>• M specifies a metadata collection</td>
</tr>
<tr>
<td></td>
<td>• R specifies a reference structure</td>
</tr>
<tr>
<td></td>
<td>• S specifies a dimension build</td>
</tr>
</tbody>
</table>

| <component_name> | Specifies the name of the component to import. This is mandatory for JobStreams, fact builds, dimension builds, metadata conformed dimensions, and metadata collections. It should not be used with reference structures, database aliases, or functions. |

| -d               | Denotes that the parameter is the name of the default database alias that Data Manager should use where none is otherwise specified. |

| <default_dbalias> | Specifies the name of the default database alias to use where none is otherwise specified. |

| -o               | Overwrites any catalog components that have the same type and name as an imported component. |

| -I               | Causes Data Manager to ignore any parse errors. |

| -v               | Denotes that the parameter is the version number stored against a component. |

| <version>        | Specifies the version number stored against a component. Version numbers follow this format:  |
|                  | x.y.z |

| -r               | Denotes that the parameter is the revision number to be stored against a component. |
### Symbol or option | Description
---|---
<revision> | Specifies the revision number to be stored against a component. Revision numbers follow this format:  
\[ x.y.z \]  
The revision number is incremented each time the component is modified and saved within Data Manager Designer.  
**Note:** The revision number cannot be older than the version number.
-u | Indicates an upgrade procedure, in other words, the component being imported is from IBM® Cognos® DecisionStream Series 7.
-L | Logs progress to the specified file. You can specify the file name using the <logfile> argument or you can use the following syntax:  
**Microsoft® Windows®** operating system syntax  
\[-L %<variable_name>%\<file_name>\]  
**UNIX®** operating system syntax  
\[-L $<variable_name>/<file_name>$\]  
logfile | Specifies the name and path of the text file to which Data Manager should log progress.
-P | Specifies that at the end of the import, a keystroke is required before terminating.
-V | Denotes that the parameter is a named variable and value pair.
-name | Specifies the name of a variable.
-value | Specifies the value that the associated variable takes.

### Example

```
catimp ODBC "DSN=SalesCat" SalesBld.bld B US_Sales  
"US Sales" -o
```

This command runs catimp to import the build named US_Sales from the text file named Sales.bld, into the catalog that resides in the ODBC data source named SalesCat, overwriting any build with the same name. In the catalog, catimp gives the build US_Sales the caption US Sales. catimp connects to the named data source as the default user, with no password.
Order of Import

Because IBM Cognos Data Manager validates objects that reside in a catalog, you must import text definitions in dependency order. For example, do not import a build until the catalog contains all the aliases and hierarchies that it uses. Normally, you should import text definitions in this order:

1. Aliases.
2. User-defined functions.
3. Reference dimensions.
4. Templates.
5. Reference structures.
6. Fact builds and dimension builds.
7. JobStreams and metadata dimensions.
8. Metadata collections.

Notes

- If you import an object from a file that defines the object more than once, the resulting definition may be invalid. To avoid this, edit the file to remove the redundant definition before importing from the file.

- Data Manager cannot import from definition files that contain more than one fact build, or dimension build. If a file contains more than one of these components, edit it to remove all but one component before importing from the file.

catbackup

This command performs a backup of an IBM Cognos Data Manager catalog to a text file.

Syntax

```
catbackup <catdb_driver_name> <catdb_driver_items> <filename>
[-L<logfile>] [-P] {-V<name>[=<value>]}  
```

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;catdb_driver_name&gt;</code></td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td><code>&lt;catdb_driver_items&gt;</code></td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog. For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
</tbody>
</table>
### Symbol or option | Description
---|---
<filename> | Specifies the path and name of the text file to which you want to backup the catalog.
-L | Logs progress to the specified file. You can specify the file name using the <logfile> argument or you can use the following syntax:
  
  **Microsoft® Windows® operating system syntax**
  
  `-L %<variable_name>%\<file_name>`
  
  **UNIX® operating system syntax**
  
  `-L $<variable_name>/\<file_name>`

<logfile> | Specifies the name and path of the text file to which Data Manager should log progress.
-P | Specifies that at the end of the backup, a keystroke is required before terminating.
-V | Denotes that the parameter is a named variable and value pair.
<name> | Specifies the name of a variable.
<value> | Specifies the value that the associated variable takes.

---

**catrestore**

This command restores a backup of an IBM® Cognos® Data Manager catalog from a text file.

**Syntax**

catrestore <catdb_driver_name> <catdb_driver_items> <filename>

[[-L<logfile>] [P] [-V<name>[=<value>]]]

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver_name&gt;</td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
</tbody>
</table>
| <catdb_driver_items> | Specifies the connection string to pass to the database driver to connect to the catalog.  
For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify "DSN=Sample;UID=DBA;PWD=PASS" |
| <filename> | Specifies the path and name of the text file that defines the catalog backup. |
### Symbol or option | Description
--- | ---
-L | Logs progress to the specified file. You can specify the file name using the `<logfile>` argument or you can use the following syntax:

Microsoft® Windows® operating system syntax
- L %<variable_name>%\<file_name>

UNIX® operating system syntax
- L $<variable_name>\<file_name>

-<logfile> | Specifies the name and path of the text file to which Data Manager should log progress.

-P | Specifies that at the end of the restore, a keystroke is required before terminating.

-V | Denotes that the parameter is a named variable and value pair.

<name> | Specifies the name of a variable.

<value> | Specifies the value that the associated variable takes.

---

**catupgrade**

This command upgrades an IBM® Cognos® Data Manager catalog.

**Syntax**

```bash
catupgrade <catdb_driver_name> <catdb_driver_items>
[-c] [-d] [-h] [-l<migrate_level>] [-s<migrate_since_timestamp>]
[-L<logfile>] [-P] {-V<name>[=<value>]
```

**Symbol or option | Description**
--- | ---

<catdb_driver_name> | Specifies the database driver to use to connect to the catalog.
<br>
<catdb_driver_items> | Specifies the connection string to pass to the database driver to connect to the catalog.

For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify "DSN=Sample;UID=DBA;PWD=PASS"

-c | Specifies that the catalog tables should be created, if necessary.

-d | Specifies that the old catalog tables are dropped after a successful upgrade.
<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Specifies that the run history be migrated.</td>
</tr>
<tr>
<td>-l</td>
<td>Denotes that the parameter is the level of run history to migrate.</td>
</tr>
<tr>
<td>&lt;migrate_level&gt;</td>
<td>Specifies the level of run history to migrate:</td>
</tr>
<tr>
<td></td>
<td>• 1 the minimum.</td>
</tr>
<tr>
<td></td>
<td>• 2 core run details (default).</td>
</tr>
<tr>
<td></td>
<td>• 3 full run details and audit trail.</td>
</tr>
<tr>
<td>-s</td>
<td>Denotes that the parameter is the date/time from which to migrate the run history.</td>
</tr>
<tr>
<td>&lt;migrate_since_timestamp&gt;</td>
<td>Specifies the date/time from which to migrate run history. You specify the date/time using this format YYYY-MM-DD [HH:MI:SS]. By default Data Manager migrates all the available run history.</td>
</tr>
<tr>
<td>-L</td>
<td>Logs progress to the specified file. You can specify the file name using the &lt;logfile&gt; argument or you can use the following syntax:</td>
</tr>
<tr>
<td></td>
<td>Microsoft® Windows® operating system syntax</td>
</tr>
<tr>
<td></td>
<td>-L %&lt;variable_name&gt;%&lt;file_name&gt;</td>
</tr>
<tr>
<td></td>
<td>UNIX® operating system syntax</td>
</tr>
<tr>
<td></td>
<td>-L $&lt;variable_name&gt;/&lt;file_name&gt;</td>
</tr>
<tr>
<td>&lt;logfile&gt;</td>
<td>Specifies the name and path of the text file to which Data Manager should log progress.</td>
</tr>
<tr>
<td>-P</td>
<td>Specifies that at the end of the upgrade, a keystroke is required before terminating.</td>
</tr>
<tr>
<td>-V</td>
<td>Denotes that the parameter is a named variable and value pair.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Specifies the name of a variable.</td>
</tr>
<tr>
<td>&lt;value&gt;</td>
<td>Specifies the value that the associated variable takes.</td>
</tr>
</tbody>
</table>

**catmanage**

This command allows you to manage an IBM® Cognos® Data Manager catalog.

**Syntax**

```
catmanage <catdb_driver_name> <catdb_driver_items> {email to}
```
Chapter 31: Commands

[-C] [-P] [-L<logfile>] [-M<email_profile>] [-s<email_subject>]
[<catalog_options>]
<options> ::=  
- c | - i | - g | - d | - k

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver_name&gt;</td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td>&lt;catdb_driver_items&gt;</td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog. For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td>emailto</td>
<td>Specifies the name of the email recipient.</td>
</tr>
<tr>
<td>-C</td>
<td>Connects to the catalog.</td>
</tr>
<tr>
<td>-P</td>
<td>Specifies that a keystroke is required before terminating.</td>
</tr>
<tr>
<td>-L</td>
<td>Logs progress to the specified file. You can specify the file name using the &lt;logfile&gt; argument or you can use the following syntax: Microsoft® Windows® operating system syntax -L %&lt;variable_name&gt;%&lt;file_name&gt; UNIX® operating system syntax -L $&lt;variable_name&gt;/&lt;file_name&gt;</td>
</tr>
<tr>
<td>&lt;logfile&gt;</td>
<td>Specifies the name and path of the text file to which Data Manager should log progress.</td>
</tr>
<tr>
<td>-M</td>
<td>Specifies that the value is an email profile.</td>
</tr>
<tr>
<td>&lt;email_profile&gt;</td>
<td>Specifies the email profile.</td>
</tr>
<tr>
<td>-s</td>
<td>Specifies that the value is an email subject.</td>
</tr>
<tr>
<td>&lt;email_subject&gt;</td>
<td>Specifies the email subject.</td>
</tr>
</tbody>
</table>
Select from the following options:

- `-c` creates the catalog tables.
- `-i` creates indexes for the catalog tables.
- `-g` grants all permissions to all users for all tables in the schema. However, this may not apply to all databases.
- `-d` deletes the contents from all the catalog tables in the schema.
- `-k` drops all the catalog tables, removing the catalog from the database.

**catdoc**

This command documents the contents of a catalog.

**Syntax**

```
catdoc <catdb_driver_name> <catdb_driver_items> <filename> [-T<file type>] [-L<logfile>] [-P]{-V<name>[=<value>]]
```

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;catdb_driver_name&gt;</code></td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td><code>&lt;catdb_driver_items&gt;</code></td>
<td>Specifies the connection string to pass to the database driver. For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td><code>&lt;filename&gt;</code></td>
<td>Specifies the path and name of the text file that defines the catalog.</td>
</tr>
<tr>
<td><code>-T</code></td>
<td>Denotes the path and name for tracing the log files.</td>
</tr>
<tr>
<td><code>&lt;file type&gt;</code></td>
<td>Specifies the log file type.</td>
</tr>
</tbody>
</table>
| `-L`                  | Logs the progress to the specified file. You can specify the file name using the `<logfile>` argument or you can use the following syntax: Microsoft® Windows® operating system syntax
  `-L`%
  `<variable_name>`\`<file_name>`
  UNIX® operating system syntax
  `-L` ${<variable_name>}/`<file_name>` |
This command shows details of the members of a reference structure defined in a catalog.

**Syntax**

```
showref <catdb_driver> <catdb_items> <refdim_name|uuid> <refdata_name|uuid> [options]
```

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;logfile&gt;</code></td>
<td>Specifies the name and path of the text file to which IBM® Cognos® Data Manager should log progress.</td>
</tr>
<tr>
<td><code>-P</code></td>
<td>Specifies that when documenting is complete, a keystroke is required before terminating.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Denotes that the parameter is a named variable and value pair.</td>
</tr>
<tr>
<td><code>&lt;name&gt;</code></td>
<td>Specifies the name of a variable.</td>
</tr>
<tr>
<td><code>&lt;value&gt;</code></td>
<td>Specifies the value that the associated variable takes.</td>
</tr>
</tbody>
</table>

**showref**

This command shows details of the members of a reference structure defined in a catalog.

**Syntax**

```
showref <catdb_driver> <catdb_items> <refdim_name|uuid> <refdata_name|uuid> [options]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;catdb_driver&gt;</code></td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td><code>&lt;catdb_items&gt;</code></td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog. For example, to connect as user DBA, with password PASS, to a catalog that resides in the ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td><code>&lt;refdim_name&gt;</code></td>
<td>Specifies the reference dimension that contains the reference structure.</td>
</tr>
<tr>
<td><code>&lt;uuid&gt;</code></td>
<td>Specifies the universally unique identifier for the reference dimension that contains the reference structure.</td>
</tr>
<tr>
<td><code>&lt;refdata_name&gt;</code></td>
<td>Specifies the name of the reference structure.</td>
</tr>
<tr>
<td><code>&lt;uuid&gt;</code></td>
<td>Specifies the universally unique identifier for the reference structure.</td>
</tr>
<tr>
<td><code>&lt;options&gt;</code></td>
<td>For information, see &quot;showref Options&quot; (p. 437).</td>
</tr>
</tbody>
</table>
Example

showref ODBC "DSN=Example Catalog" Fiscal -b
Year -- {Year} 4 members [0 leaf nodes]
    Quarter -- {Quarter} 13 members [1 leaf node]
    Period -- {Period} 36 members [36 leaf nodes]

This example runs showref to produce a brief description of the Fiscal hierarchy defined in the catalog that resides in the ODBC data source named Example Catalog.

showref Options

Syntax
<options> ::= [-e<runenv>] [-A<dbalias_file>]
[-b] [-h] [-l] [-m<mindepth>] [-0[<output_file>]] [-x<maxdepth>]
[-F] [-M] [-P] {-V<name>[=<value>]}
Specifies the name and path of the text file to which Data Manager should write the reference structure listing.

Denotes that the parameter gives the maximum depth (in hierarchy levels) for the listing.

Specifies the maximum depth (in number of hierarchy levels) to list.

Removes any unused foster parents from the reference structure listing.

Reports all messages when caching the reference data item. This overrides any reporting limit set on the Features tab of the reference structure Properties window.

Specifies that a keystroke is required before terminating.

Denotes that the parameter is a named variable and value pair.

Specifies the name of the variable to use during item substitution.

Specifies the value that the corresponding named variable takes.

dstofm

This command exports a conformed model description to an XML file. This file can then be used to produce a model of your target data mart or data warehouse in IBM® Cognos® Framework Manager, without the need to recreate the complete model.

Syntax

dstofm <catdb_driver> <catdb_items> <output_file> {<collection_name>} [-D<description>] [-N<name>] [-P] [-U] {-V<name>[=<value>]}
<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;description&gt;</td>
<td>Specifies the description for the exported metadata file.</td>
</tr>
<tr>
<td>-N</td>
<td>Denotes the model name for the exported metadata file.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Specifies the model name for the metadata export.</td>
</tr>
<tr>
<td>-P</td>
<td>Specifies that at the end of a build, a keystroke is required before terminating.</td>
</tr>
<tr>
<td>-U</td>
<td>Specifies that unreferenced metadata dimensions are to be included.</td>
</tr>
<tr>
<td>-V</td>
<td>Denotes that the parameter is a named variable and value pair.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Specifies the variable to use during item substitution.</td>
</tr>
<tr>
<td>&lt;value&gt;</td>
<td>Specifies the value that the corresponding variable is to use.</td>
</tr>
</tbody>
</table>

**Example**

dstofm ODBC "DSN=Example Catalog" ExampleExport.xml -N ExampleModel

This example runs dstofm to export the example conformed model description that resides in the Example Catalog, to the metadata export file named ExampleExport.xml.

**sqlterm**

This command starts the SQLTerm application (the IBM® Cognos® Data Manager terminal for SQL databases). Using SQLTerm, you can

- connect to and query SQL databases
- obtain information about database objects, DBMS drivers, and data sources
- run SQL scripts

You can work with SQLTerm in either of these modes:

- from the command line to execute an SQL script
- in the SQLTerm shell

**Syntax**

```
sqlterm [<driver_name> <item_string>]
[-d<dbalias>] [-e<runenv>] [-A<dbalias_file>]
[-f<sql_file>] [-o<output_file>] [-w<num_width>]
[-C] [-E] [-P] {-V<name>[=<value>]}  
```
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;driver_name&gt;</td>
<td>Specifies the name of the DBMS driver to use to connect to the data.</td>
</tr>
<tr>
<td>&lt;item_string&gt;</td>
<td>Specifies the connection string to pass to the DBMS driver to establish</td>
</tr>
<tr>
<td></td>
<td>the data connection.</td>
</tr>
<tr>
<td>-d</td>
<td>Denotes the default database alias.</td>
</tr>
<tr>
<td>&lt;dbalias&gt;</td>
<td>Specifies the name of the alias that defines the database to which Data</td>
</tr>
<tr>
<td></td>
<td>Manager should connect.</td>
</tr>
<tr>
<td></td>
<td>For more information, see &quot;Use an Alias to Connect to a Database&quot; (p. 68).</td>
</tr>
<tr>
<td>-e</td>
<td>Denotes that the parameter is the name of a Data Manager run environment</td>
</tr>
<tr>
<td>&lt;runenv&gt;</td>
<td>Specifies the run environment context to use to determine the database</td>
</tr>
<tr>
<td></td>
<td>to which each alias refers.</td>
</tr>
<tr>
<td>-A</td>
<td>Denotes the text file that contains the database alias.</td>
</tr>
<tr>
<td>&lt;dbalias_file&gt;</td>
<td>Specifies the name and path of the file that defines database connection</td>
</tr>
<tr>
<td></td>
<td>parameters for each alias and run environment.</td>
</tr>
<tr>
<td>-f</td>
<td>Denotes that the parameter is the name of a script file.</td>
</tr>
<tr>
<td>&lt;sql_file&gt;</td>
<td>Specifies the path and name of the script file that SQLTerm should</td>
</tr>
<tr>
<td></td>
<td>execute. When you choose this option, SQLTerm executes the script and then</td>
</tr>
<tr>
<td></td>
<td>terminates.</td>
</tr>
<tr>
<td>-o</td>
<td>Denotes that the parameter is the name of a text file to which Data</td>
</tr>
<tr>
<td></td>
<td>Manager should write the output.</td>
</tr>
<tr>
<td>&lt;output_file&gt;</td>
<td>Specifies the name and path of the text file to which Data Manager</td>
</tr>
<tr>
<td></td>
<td>should write the output.</td>
</tr>
<tr>
<td>-w</td>
<td>Denotes that the parameter is the width at which SQLTerm should wrap its</td>
</tr>
<tr>
<td>&lt;num_width&gt;</td>
<td>Specifies the width in number of characters at which SQLTerm should wrap</td>
</tr>
<tr>
<td></td>
<td>its output.</td>
</tr>
<tr>
<td>-C</td>
<td>Specifies that Cognos® SQL is to be the default.</td>
</tr>
</tbody>
</table>
### Symbol | Description
--- | ---
-E | Causes SQLTerm to echo SQL statements to the console immediately before it executes them.
-P | Specifies that a keystroke is required before terminating.
-V | Denotes that the parameter is a variable name and value pair.
<name> | Specifies the name of the variable.
<value> | Specifies the value that the named variable takes.

### Example
- `sqlterm ODBC "DSN=Sales;UID=user;PWD=pass"`
  
  This example runs SQLTerm to automatically connect to the ODBC data source named Sales as a user named user with the password pass.

- `sqlterm -fmyscript.sql`
  
  This example runs SQLTerm to execute the script named myscript.sql. SQLTerm terminates when it has completed processing the script.

- `sqlterm -dSales -eDEVELOP -Astdalias.def`
  
  This example runs SQLTerm to connect to the database listed in the file named stdalias.def as alias Sales for the run environment context DEVELOP.

For information on SQL scripts, see "Work with SQL Scripts" (p. 451).

### Working in the SQLTerm Shell

SQLTerm makes available the following commands:
- `!` (system command)
- `COGNOS SQL`
- `CONNECT`
- `DBALIAS`
- `DISCONNECT`
- `DESCRIBE`
- `DIFF`
- `DRIVERS`
- `ECHO`
- `EXIT`
! (System Command)

Use this to issue an operating system command in the SQLTerm shell.

**Syntax**

```
!<syscmd>
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;syscmd&gt;</td>
<td>Specifies the operating system command to execute.</td>
</tr>
</tbody>
</table>

**Example**

```
!dir
```

This example executes the Microsoft® Windows® operating system dir command to list the contents of the current directory.

**Cognos SQL**

Use this command to use Cognos® SQL rather than native SQL.
**Syntax**
Cognos SQL

**CONNECT**
Use this command to directly connect to a data source. SQLTerm terminates any existing connection before establishing the connection to the required data source.

**Syntax**
CONNECT <driver> <items>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;driver&gt;</td>
<td>Specifies the name of the DBMS driver to use to connect to the data.</td>
</tr>
<tr>
<td>&lt;items&gt;</td>
<td>Specifies the connection string that to pass to the DBMS driver to establish the connection.</td>
</tr>
</tbody>
</table>

**Example**
CONNECT ODBC DSN=Sales;UID=user;PWD=pass;

This example instructs SQLTerm to connect to the ODBC data source named Sales as a user named user with password pass.

**DBALIAS**
Use this command to connect to a data source using a database alias. SQLTerm terminates any existing connection before establishing the connection to the required data source.

**Syntax**
DBALIAS <dbalias>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dbalias&gt;</td>
<td>Specifies the name of the alias to which SQLTerm should connect.</td>
</tr>
</tbody>
</table>

To use this command, you must pass the name of a valid database alias definition file to SQLTerm. You can either do this using the -A command line option, or by setting the DS_DBALIAS_FILE environment variable. If you require a run environment context other than <DEFAULT>, you must also pass the run environment name by using the -e command line option or by setting the DS_RUNENV environment variable.

**Example**
DBALIAS Sales

This example instructs SQLTerm to connect to the database to which the alias Sales refers.

**DISCONNECT**
Use this command to terminate an existing connection to a database.
Syntax
DISCONNECT

DESCRIBE
On its own, the DESCRIBE command lists the tables in the current database.
If you supply a valid table name, SQLTerm details each column in the table. Where possible, these are the column name, its data type and length, whether the column can accept null values, and whether the column forms part of the primary key of the table.

Syntax
DESCRIBE [<table_name>]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;table_name&gt;</td>
<td>Specifies the name of a table in the currently connected database.</td>
</tr>
</tbody>
</table>

Example
This example lists the tables in the currently connected database:
```
: DESCRIBE
Table Name
-----------------------------------------------
ConversionRate  
Country        
EuroConversion 
OrderDetail    
OrderHeader    
OrderMethod    
Product        
ProductForecast
ProductLine    
ProductType    
ReturnReason   
ReturnedItem   
SalesBranch    
SalesStaff     
SalesTarget    
15 tables
```

DIFF
This command lists the differences between two tables that you specify. Both tables must reside in the database to which sqlterm is connected.

Syntax
DIFF <tab 1> <tab 2>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;table_1&gt;</td>
<td>Specifies the first of the two tables to compare.</td>
</tr>
<tr>
<td>&lt;table_2&gt;</td>
<td>Specifies the second of the two tables to compare.</td>
</tr>
</tbody>
</table>

**DRIVERS**

This command lists the DBMS drivers for which IBM® Cognos® Data Manager is configured. Use the ALL argument to obtain a list of all the drivers known to Data Manager.

**Syntax**

```
DRIVERS [ALL]
```

**ECHO**

ECHO ON performs the same function as the -E command line argument. It causes SQLTerm to echo SQL statements to the console immediately before it executes them. ECHO OFF prevents SQLTerm from echoing SQL statements to the console. On its own, ECHO returns its status (either ON or OFF).

This command is particularly useful if you use redirection to provide the standard input for SQLTerm. ECHO is OFF by default when SQLTerm takes input from the standard input. When you type an SQL statement at the keyboard, it appears on screen as you type and does not need to be echoed. However, input that is redirected from a text file does not appear on screen. Setting ECHO to ON allows you to see the statements that SQLTerm executes.

**Syntax**

```
ECHO [ON | OFF]
```

**EXIT**

This command causes SQLTerm to terminate.

**Syntax**

```
EXIT
```

Synonyms for this command are QUIT, EX, and \q.

**FETCHSIZE**

This command sets or shows the fetch row count used to query the database.

**Syntax**

```
FETCHSIZE <N>
```
This setting controls the number of rows that are queried at once by using the bulk fetch facility for your database. You can use this to tune large extractions from a source database. This setting has no effect for databases that do not support bulk fetch operations. The default fetch row size is 50.

## GO

This command causes SQLTerm to execute an SQL statement.

Synonyms for the GO command are ;, \p\g, and \g. For example, SELECT * FROM product;

### Syntax

<statement>GO

### Example

SELECT DISTINCT product_id, product_name FROM productsGO

This example instructs SQLTerm to execute the SQL statement, SELECT DISTINCT product_id, product_name FROM product.

## HELP

This command returns a list of available SQLTerm commands and a brief description of each.

### Syntax

HELP

## ISOLEVEL

This command sets or shows the database isolation level defined for the database transaction you are using to query the data source.

### Syntax

ISOLEVEL <N>
This setting can be used where you specifically want to control the transaction isolation level for a query transaction. This setting has no effect for database systems that do not support transaction isolation levels. The default value is determined by the source database system.

The values you can set are
- 0 (DBMS default)
- 1 (Read uncommitted)
- 2 (Read committed)
- 3 (Cursor stability)
- 4 (Repeatable read)
- 5 (Phantom protection)
- 6 (Serializable)

### NUMBER

Use this command to set or show the width at which SQLTerm displays columns of numeric data.

**Syntax**

```
NUMBER [<width>]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;width&gt;</td>
<td>Specifies the width in number of characters, at which SQLTerm should show columns of numeric data.</td>
</tr>
</tbody>
</table>

### OBJECTS

Use this command to list objects in the database to which SQLTerm is currently connected.

**Syntax**

```
OBJECTS [T][V][S][*] [<schema>]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>Lists tables.</td>
</tr>
<tr>
<td>V</td>
<td>Lists views.</td>
</tr>
<tr>
<td>S</td>
<td>Lists schemas.</td>
</tr>
<tr>
<td>*</td>
<td>Lists all objects in the database. (default)</td>
</tr>
</tbody>
</table>
NATIVE SQL

Use this command to specify that native SQL is to be used rather than Cognos® SQL.

Syntax

NATIVE SQL

OUTPUT

Use this command to determine whether SQLTerm should show the data that results from SQL SELECT statements.

Syntax

OUTPUT [ON | OFF]

On its own, this command returns the OUTPUT setting (either ON or OFF). OUTPUT ON causes SQLTerm to show data; OUTPUT OFF causes SQLTerm to suppress data display. This command is useful if you want to obtain a row count for a large data table.

Example

: DBALIAS 'Sales'
Connected to ODBC
: OUTPUT OFF
OUTPUT is OFF
: SELECT * FROM ds_sales;
10529 rows selected
:

PROGRESS

This command shows select progress every n rows.

Syntax

PROGRESS <N>

PROMPT

This command shows a line of text.

Syntax

PROMPT <text>
QUIT

This command causes SQLTerm to terminate.

Syntax
QUIT

SCHEMA

This command lists the schemas and user names from the current connection.

Syntax
SCHEMA

SCREEN

This command sets the width in number of characters, at which SQLTerm wraps its output. On its own, it returns the current width.

Syntax
SCREEN [<width>]

SCRIPT

This command causes SQLTerm to execute an SQL script.

Syntax
SCRIPT <filename>

For information about working with SQL scripts, see "Work with SQL Scripts" (p. 451).

SOURCES

This command causes SQLTerm to return a list of available data sources.

SOURCES lists data sources that are available using the specified driver. If you specify the optional ALL argument, SOURCES lists all data sources that are available using any DBMS driver.

Syntax
SOURCES [<driver>ALL]

<table>
<thead>
<tr>
<th>Symbol or option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;driver&gt;</td>
<td>Specifies the DBMS driver for which SOURCES should list available data sources. By default this is the current driver.</td>
</tr>
</tbody>
</table>
### Statement

This command indicates the type of the next SQL statement.

**Syntax**

```
STATEMENT [SELECT|DML|DDL]
```

### Statistics

This command determines whether SQLTerm returns timing statistics after executing each SQL statement.

On its own, STATISTICS returns its status (either ON or OFF). STATISTICS ON causes SQLTerm to return timing statistics after it executes each SQL statement. STATISTICS OFF turns this off.

This command is particularly useful in determining SQL statement efficiency.

**Syntax**

```
STATISTICS [ON|OFF]
```

### Time

This command shows the system date and time.

**Syntax**

```
TIME
```

### Types

This command shows information about the current connection. For ODBC, it also shows information about the data type conversion.

**Syntax**

```
TYPES
```
**Work with SQL Scripts**

You can create SQL scripts in any text editor. In addition to SQL statements, SQL scripts can contain any valid SQLTerm commands. You can run them on the command line by using the `-f` argument. You can also run them from the SQLTerm shell by using the `SCRIPT` command.

The following script connects to an ODBC data source, returns the number of data rows for a data table, disconnects from the data source, and terminates SQLTerm.

```sql
CONNECT ODBC 'DSN=GOSales'
SELECT COUNT(*) FROM SalesStaff;
DISCONNECT
EXIT
```

If the script is named example.sql then the following example runs it from the command line:

`sqlterm -f example.sql`

If the script is named example.sql, then this example runs it from the SQLTerm shell:

`SCRIPT example.sql`

**Redirection**

Most operating systems support redirection. It allows you to specify that a program should take its input from a text file, rather than from the keyboard. You can also use redirection to send the output to a text file, instead of the computer monitor. On Microsoft® Windows® and UNIX® operating systems, the syntax for this is:

```
sqlterm [<command_options>] [< <scriptname>] [> <outputfile>]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;command_options&gt;</code></td>
<td>Specifies the command line options for SQLTerm. For more information, see &quot;sqlterm&quot; (p. 439).</td>
</tr>
<tr>
<td><code>&lt;scriptname&gt;</code></td>
<td>Specifies the name and path of the file from which the operating system should supply input to SQLTerm.</td>
</tr>
<tr>
<td><code>&lt;outputfile&gt;</code></td>
<td>Specifies the name and path of the text file to which the operating system should send output from SQLTerm.</td>
</tr>
</tbody>
</table>

**Example**

`sqlterm -E < sample.sql > results.log`

This example runs SQLTerm with ECHO on, taking its input from the file named sample.sql, and saving the output to results.log.

**Note:** By default, ECHO is OFF when you use redirection. This means that SQL statements do not appear in the output stream. If you want the SQL statements to appear in the output stream, you should use the `-E` command line option. Alternatively, you can issue an ECHO ON command in the script.
catpkgtoxml

This command converts an IBM® Cognos® Data Manager package file to a data movement specification XML file.

Syntax

catpkgtoxml <pkgfile> <xmlfile> [defaultRunnableName] [-d|-f|-j] [-P] {-V<name>[=<value>]}
based objects, you must import the definitions in dependency order. For more information, see "Order of Import" (p. 430).

You can also work directly with package files without requiring that they be imported into a host catalog before executing a build or JobStream. The databuild, dimbuild, and rundsjob commands have command line options to allow a build or uuid to be specified and the package treated as the complete store of components to be executed. This can be important when executing builds and JobStreams from other applications when it is not possible to have the required relational database in place to hold the catalog components.
Appendix A: Fact Delivery Module Properties

In IBM® Cognos® Data Manager you use delivery modules to deliver the transformed data to the specified data repositories. Data Manager organizes the delivery modules into these groups:

- fact delivery modules that deliver data produced by a fact build
- dimension delivery modules that deliver data describing a single dimension

For more information on setting up delivery modules, see "Delivering Fact Data Using a Fact Build" (p. 207) and "Delivering Dimension Data Using a Fact Build" (p. 227).

Each fact delivery module has different module properties, and may also have element properties that you can set.

These are the fact delivery modules:

- DB2 LOAD
- Essbase Direct Feed
- Essbase File Feed
- Informix LOAD
- Microsoft SQL Server BCP Command
- Netezza NZLoad
- ORACLE SQL*Loader
- Red Brick Loader (TMU)
- Relational Table
- SQL Server Bulk Copy via API
- Sybase ASE BCP Command
- Sybase IQ LOAD
- TM1 Turbo Integrator
- Teradata Fastload
- Teradata Multiload
- Teradata TPump
- Text File

Note: Your Data Manager license may not include all the delivery modules described. Please contact your Data Manager software vendor to extend your license for additional delivery modules.
Appendix A: Fact Delivery Module Properties

**DB2 LOAD Delivery Module**

The DB2® LOAD delivery module delivers data directly to IBM® DB2 databases using the DB2 bulk loader.

**Note:** The bulk loader command for DB2® is designed for the DB2® UDB loader for Microsoft® Windows® and UNIX® operating systems. To deliver output for bulk loading into DB2/400 or DB2/MVS, in the Operation box, select Create Loader Files.

**Module Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader</strong> to deliver the text data and control files, load the data</td>
</tr>
<tr>
<td></td>
<td>into the database, and clean up the control files and any other intermediate</td>
</tr>
<tr>
<td></td>
<td>files. (default)</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader (leave files)</strong> to deliver the text data and control files,</td>
</tr>
<tr>
<td></td>
<td>load the data into the database, but leave the control files and any other</td>
</tr>
<tr>
<td></td>
<td>intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>● <strong>Create Loader Files</strong> to deliver the text data and control files, but</td>
</tr>
<tr>
<td></td>
<td>not invoke the loader to load the data into the database. You should use</td>
</tr>
<tr>
<td></td>
<td>this option to deliver output for bulk loading into DB2/400 or DB2/MVS.</td>
</tr>
<tr>
<td></td>
<td>The control file generated uses options specific to DB2® UDB for</td>
</tr>
<tr>
<td></td>
<td>Microsoft® Windows® and UNIX® operating systems and generates a machine</td>
</tr>
<tr>
<td></td>
<td>portable text data file.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager</td>
</tr>
<tr>
<td></td>
<td>loads data into its Data directory.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If Create Loader Files is selected in the Operation box, specify the prefix</td>
</tr>
<tr>
<td></td>
<td>to use to generate the file names. If this box is empty, a default value</td>
</tr>
<tr>
<td></td>
<td>is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the loader generates.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>No row count check</td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td>Log file success message</td>
<td>If No row count check is not selected, this property is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td>Use named pipes</td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
</tbody>
</table>
| Load mode                | Specifies one of these load modes:  
  • Insert inserts the data in the target table. (default)  
  • Replace deletes the contents of the target table but not the indexes before delivering the data. This is equivalent to a DELETE FROM statement. |
| Data file encoding       | Specifies the data file encoding to use.                                                                                                                                                                   |
| Load from client file    | Specifies that data should be loaded from a client file that is not stored on the DB2® server.                                                                                                                  |
| Savecount                | An integer that specifies the number of rows at which the loader should establish a consistency point. This allows a failed load to be restarted from the last consistency point. The default value is zero, which means that no consistency points are established, unless necessary. |
| Warningcount             | An integer that specifies the maximum number of warnings that can occur before the loader stops. The default is zero which means that the loader does not stop.                                                      |
**Property** | **Description**
--- | ---
Additional load options | Specifies a string of extra load options to append to the command.

**DB2 CLP command** | Specifies the path and command for the DB2 loader process.

*Note:* You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

---

**Element Properties**

For this delivery module, the element properties are the same for all element types.

**Property** | **Description**
--- | ---
Value if null | Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.

Perform escape processing | Specifies whether to use the escape character (\) before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.

Floating point scale | Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.

---

**Essbase Direct Feed Delivery Module**

The Essbase Direct Feed delivery module delivers data directly to an Oracle Essbase server.

If the Essbase server stores the data in an IBM® DB2® database, this module gives automatic support for DB2® OLAP.

---

**Module Properties**

**Property** | **Description**
--- | ---
Server login | Specifies the Essbase login details. This value is mandatory.
### Property Description

<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application: database</strong></td>
<td>Specifies the Essbase application and database to which you want to connect. This value is mandatory.</td>
</tr>
<tr>
<td><strong>Rows per update</strong></td>
<td>Specifies the number of rows to be written to Essbase before changes become permanent.</td>
</tr>
<tr>
<td><strong>Clear existing data</strong></td>
<td>Specifies whether to delete any existing data from the Essbase cube before starting the load process.</td>
</tr>
<tr>
<td><strong>Calculate after load</strong></td>
<td>Specifies the Essbase calculation to run on completion of the load process.</td>
</tr>
<tr>
<td><strong>Update outline</strong></td>
<td>Specifies whether to update the Essbase outline throughout the load process.</td>
</tr>
</tbody>
</table>

### Element Properties

The element properties available depend on the type of element being defined. Note that attribute elements are not available for this delivery module.

<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Member name expression</strong></td>
<td>An expression that specifies the member attribute to use as the Essbase member name. By default, IBM Cognos Data Manager uses ID. This value is an expression, for example, <code>concat(CAPTION, '(', ID, ')')</code> This value is mandatory.</td>
</tr>
<tr>
<td><strong>Member alias expression</strong></td>
<td>An expression that specifies the member attribute to use as the Essbase member alias.</td>
</tr>
<tr>
<td><strong>Dimension name</strong></td>
<td>Specifies the name of the Essbase dimension that corresponds to the Data Manager dimensions.</td>
</tr>
<tr>
<td><strong>Full outline update</strong></td>
<td>Specifies whether to completely rebuild the Essbase outline for a dimension.</td>
</tr>
<tr>
<td><strong>Member name</strong></td>
<td>Specifies the name of the member in the measures dimension.</td>
</tr>
<tr>
<td><strong>Floating point scale</strong></td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>
Essbase File Feed Delivery Module

The Essbase File Feed delivery module delivers data to flat files that may then be loaded into Oracle Essbase.

If the Essbase server stores the data in an IBM® DB2® database, this module gives automatic support for DB2 OLAP.

Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data file</td>
<td>Specifies the name of the file to hold the data for loading into Essbase. This value is mandatory.</td>
</tr>
<tr>
<td>Generate dimension build files</td>
<td>Specifies whether to generate Essbase dimension files in addition to the fact data.</td>
</tr>
</tbody>
</table>

Element Properties

The element properties that are available depend on the type element being defined. Note that attribute elements are not available for this delivery module.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member name expression</td>
<td>An expression that specifies the member attribute to use as the Essbase member name. By default, IBM® Cognos® Data Manager uses ID. This value is mandatory.</td>
</tr>
<tr>
<td>Member alias expression</td>
<td>An expression that specifies the member attribute to use as the Essbase member alias.</td>
</tr>
<tr>
<td>Dimension name</td>
<td>Specifies the name of the Essbase dimension that corresponds to the Data Manager dimensions.</td>
</tr>
<tr>
<td>Member name</td>
<td>Specifies the name of the member in the measures dimension.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

Informix LOAD Delivery Module

The Informix® LOAD delivery module delivers data directly to an Informix® database using the Informix® DBACCESS command.
## Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader</strong> to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader (leave files)</strong> to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Create Loader Files</strong> to deliver the text data and control files, but not invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create Loader Files</strong> is selected in the <strong>Operation</strong> box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td><strong>Log file success message</strong></td>
<td>If <strong>No row count check</strong> is not selected, this property is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td><strong>Use named pipes</strong></td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
</tbody>
</table>
### Load mode

Specifies one of these load modes:

- **Append** appends the data to the target table. (default)
- **Truncate** deletes the contents of the target table together with indexes, before data delivery. This is equivalent to a DELETE FROM statement but is usually faster.
- **Replace** deletes the contents of the target table but not the indexes before data delivery. This is equivalent to a DELETE FROM statement.

### Informix 'dbaccess' command

Specifies the path and the command for the Informix® loader process.

**Note:** You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

---

### Element Properties

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value if null</strong></td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td><strong>Perform escape processing</strong></td>
<td>Specifies whether to use the escape character () before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td><strong>Floating point scale</strong></td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

---

### Microsoft SQL Server BCP Command Delivery Module

The Microsoft® SQL Server BCP Command delivery module delivers data directly to Microsoft® SQL Server databases using the Microsoft® SQL Server BCP bulk load utility.
The BCP command utility is a component of the Microsoft® SQL Server client installation. It provides users with an intermediate load file that is passed to the BCP utility.

**Note:** To load to Microsoft® SQL Server without using an intermediate load file use the SQL Server Bulk Copy via API delivery module.

## Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>- <strong>Run Loader</strong> to deliver the text data and control files, load</td>
</tr>
<tr>
<td></td>
<td>the data into the database and clean up the control files and any other</td>
</tr>
<tr>
<td></td>
<td>intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Run Loader (leave files]</strong> to deliver the text data and control</td>
</tr>
<tr>
<td></td>
<td>files, load the data into the database but leave the control files and</td>
</tr>
<tr>
<td></td>
<td>any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Create Loader Files</strong> to deliver the text data and control files, but</td>
</tr>
<tr>
<td></td>
<td>not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data</td>
</tr>
<tr>
<td></td>
<td>Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> To view the Data directory, from the <strong>Tools</strong> menu, click</td>
</tr>
<tr>
<td></td>
<td><strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create Loader Files</strong> is selected in the <strong>Operation</strong> box, specify</td>
</tr>
<tr>
<td></td>
<td>the prefix to use to generate the file names. If this box is empty, a</td>
</tr>
<tr>
<td></td>
<td>default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data.</td>
</tr>
<tr>
<td></td>
<td>Clear this check box to prevent an error if the target DBMS does not</td>
</tr>
<tr>
<td></td>
<td>report successful delivery in the expected format. Select this check box</td>
</tr>
<tr>
<td></td>
<td>to use a newer version of the DBMS where the vendor changed the format of</td>
</tr>
<tr>
<td></td>
<td>the completion message.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Log file success message</td>
<td>If No row count check is not selected, this property is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td>Record delimiter</td>
<td>Specifies the character sequence to use to separate records. Specify a character string or use one of these: Tab, Comma, Space, None, NL (new line), CRLF (carriage return and line feed combination) on a Microsoft® Windows® operating system, or CR (carriage return) on a UNIX® operating system. NL is the default.</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
</tbody>
</table>
| Load mode                            | Specifies one of these load modes:  
  - **Append** appends the data to the target table. (default)  
  - **Truncate** deletes the contents of the target table, and drops all indexes, before delivering the data. This is equivalent to a DELETE FROM statement, but is usually faster.  
  - **Replace** deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.  
  - **Insert** inserts the data in the target table.                                                                                                                                                                                                                                                                               |
| Data file encoding                   | Specifies the data file encoding to use.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Treat empty string as null value     | Specifies whether empty string values should be read as null values.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| SQL Server BCP command               | Specifies the path and the command for the loader process. **Note:** You must specify the exact path name and command, including any double quotation marks required. For more information, see "Specify Executables to Deliver Data" (p. 511).                                                                                                                                                                                                                       |
| Additional command line options      | Allows additional BCP command line options to be specified.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
Element Properties

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

Netezza NZLoad Delivery Module

The Netezza NZLoad delivery module delivers data directly to a Netezza database using the NZLoad utility.

Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>• Run Loader to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>• Run Loader(leave files) to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>• Create Loader Files to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td>File directory</td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td>Tip: To view the Data directory, from the Tools menu, click Browse Data Files.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Filename prefix</td>
<td>If Create Loader Files is selected in the Operation box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td>Dump log file</td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td>No row count check</td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td>Log file success message</td>
<td>If No row count check is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td>Use named pipes</td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Record delimiter</td>
<td>Specifies the character sequence to use to separate records. Specify a character string or use one of these: Tab, Comma, Space, None, NL (new line), CRLF (carriage return and line feed combination) on a Microsoft® Windows® operating system, or CR (carriage return) on a UNIX® operating system. NL is the default.</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
</tbody>
</table>
### Property Description

**Load mode**
- Specifies one of these load modes:
  - **Append** appends the data to the target table. (default)
  - **Truncate** deletes the contents of the target table and indexes, before delivering the data. This is usually faster than a DELETE FROM statement.
  - **Replace** deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.
  - **Insert** inserts the data in the target table.

For more information, see the bulk loader documentation.

**Host**
- Specifies the server name for the database connection.

**Database**
- Specifies the database name.

**Data file encoding**
- Specifies the data file encoding to use.

**Use quotes**
- Specifies whether to use quotation marks to enclose data values.
  - By default, no quotation marks are used.

**Quote value**
- Specifies the type of quotation character to use for enclosing data values.

**Null value indicator**
- Specifies the value to represent null data values. By default, this value is a blank string.

**Additional command line options**
- Allows additional Netezza command line options to be specified.

**Loader command**
- Specifies the path and the command for the loader process.
  - **Note:** You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

### Element Properties

For this delivery module, the element properties are the same for all element types.
Appendix A: Fact Delivery Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Perform escape processing</td>
<td>Specifies whether to use the escape character () before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

**ORACLE SQL*Loader Delivery Module**

The ORACLE SQL*Loader delivery module delivers data directly to an Oracle database using the Oracle Bulk Load utility.

**Module Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>- Run Loader to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>- Run Loader (leave files) to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>- Create Loader Files to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td>File directory</td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> To view the Data directory, from the Tools menu, click Browse Data Files.</td>
</tr>
<tr>
<td><strong>Property</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Filename prefix</td>
<td>If Create Loader Files is selected in the Operation box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td>Dump log file</td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td>No row count check</td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td>Log file success message</td>
<td>If No row count check is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as sub-strings in each line of the log file or output file.</td>
</tr>
<tr>
<td>Use named pipes</td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Record delimiter</td>
<td>Specifies the character sequence to use to separate records. Specify a character string or use one of these: Tab, Comma, Space, None, NL (new line), CRLF (carriage return and line feed combination) on a Microsoft® Windows® operating system, or CR (carriage return) on a UNIX® operating system. NL is the default.</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
</tbody>
</table>
### Load mode

Specifies one of these load modes:

- **Append** appends the data to the target table. (default)
- **Truncate** deletes the contents of the target table and indexes, before delivering the data. This is usually faster than a DELETE FROM statement.
- **Replace** deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.
- **Insert** inserts the data in the target table.

For more information, see the bulk loader documentation.

### Data file encoding

Specifies the data file encoding to use.

### Control file encoding

Specifies the control file encoding to use.

### Direct load

Specifies whether Data Manager should use the Oracle DIRECT path instead of the Oracle CONVENTIONAL path for the load.

### Unrecoverable (direct load only)

In conjunction with the Direct load property, specifies whether to invoke the Oracle Unrecoverable option. This option can reduce load time but, in case of load failure, may result in corruption of the target table.

### CLOB data inline

Specifies whether to treat Oracle CLOB (Character Large Object) data as character data. By default, CLOB data is written to temporary files which are referenced from the loader data file, whereas character data is written directly into the data file.

Writing CLOB data directly into the data file allows faster loading.

**Notes**

- All CLOB data be must less than 8000 characters in length.
- The CLOB data must not contain any special characters that are likely to conflict with load control characters, such as new line characters.
### Red Brick Loader (TMU) Delivery Module

The Red Brick® Loader (TMU) delivery module delivers data directly to an IBM® Red Brick® database using the Red Brick® Bulk Load utility.

---

**Element Properties**

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>
### Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Specifies whether IBM® Cognos® Data Manager should&lt;br&gt;● <strong>Run Loader</strong> to deliver the text data and control files, load the data into the database, and clean the control files and any other intermediate files. (default)&lt;br&gt;● <strong>Run Loader(leave files)</strong> to deliver the text data and control files, load the data into the database, but leave the control files and any other intermediate files so that you can view them.&lt;br&gt;● <strong>Create Loader Files</strong> to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td>File directory</td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.&lt;br&gt;Tip: To view the Data directory, from the Tools menu, click Browse Data Files.</td>
</tr>
<tr>
<td>Filename prefix</td>
<td>If <strong>Create Loader Files</strong> is selected in the Operation box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td>Dump log file</td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td>Use named pipes</td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>TMU &lt;database&gt;</td>
<td>Specifies the database for the TMU connection.</td>
</tr>
</tbody>
</table>
**Load mode**
Specifies one of these load modes:
- **Append** mode appends the data to the target table. (default)
- **Replace** mode deletes the contents of the target table but not the indexes, before delivering data. This is equivalent to a DELETE FROM statement.
- **Insert** mode inserts the data in the target table.
- **Modify** mode updates rows where the delivered rows have the same key as a row that exists in the target table. When there is no match, Data Manager appends the row to the target table.
- **Update** mode updates rows where delivered rows have the same key as a row that exists in the target table. When there is no match, Data Manager discards the rows that do not correspond to an existing row in the target table.

For more information, see the bulk loader documentation.

**Optimized load**
Specifies whether to perform a Red Brick® optimized load.

**Red Brick TMU command**
Specifies the command to run the TMU process.

**Note:** You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

---

**Element Properties**
For this delivery module, the element properties are the same for all element types.

**Value if null**
Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.

**Floating point scale**
Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.
Relational Table Delivery Module

The Relational Table delivery module delivers fact build data to relational tables. These tables can then be accessed by client-server analysis applications, or can form the basis of standard reporting systems.

Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh type</td>
<td>Specifies the method by which rows should be applied to the output table:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Append</strong> appends the data to the target table. (default)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Truncate</strong> deletes the contents of the target table, together with all indexes, before delivering data. This is equivalent to a DELETE FROM statement but is usually faster.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Replace</strong> deletes the contents of the target table but not the indexes, before delivering data. This is equivalent to a DELETE FROM statement.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Update/Insert</strong> updates or inserts data in the target table. The value selected in the Update detection method box determines the way this is performed.</td>
</tr>
<tr>
<td></td>
<td>For more information, see &quot;Update/Insert Delivery Method&quot; (p. 476)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Update</strong> updates rows where delivery records have the same key as a row that exists in the target table. When there is no match, IBM® Cognos® Data Manager discards the records that do not correspond to an existing row in the target table.</td>
</tr>
<tr>
<td></td>
<td>For more information, see &quot;Update Delivery Method&quot; (p. 483)</td>
</tr>
<tr>
<td>Incoming record keys are unique</td>
<td>If you are using the Update/Insert or Update refresh type, selecting this option indicates that there is a unique key for every record contained in the delivery data.</td>
</tr>
<tr>
<td></td>
<td>For more information, see &quot;Unique Incoming Record Keys for Update/Insert&quot; (p. 479).</td>
</tr>
</tbody>
</table>
If you are using the Update/Insert or Update refresh type, the way in which Data Manager updates or inserts data in the target table depends upon the update detection method you choose:

- **Update** (default)
- **Select**
- **Select/Compare**
- **Key Cache**

For more information, see "Update Detection Method for Update/Insert" (p. 477) and "Update Detection Method for Update" (p. 484).

**Key Cache max memory (MB)**

If you are using the Update/Insert or Update refresh type, selecting this option specifies the maximum size of the key cache and its location. Enter

- a negative value to store the key cache in memory
- zero to store the key cache on disk
- a positive value <n> to indicate that memory is to be used if the calculated size is less than or equal to <n>, otherwise use disk

**Note:** By default, the key cache is stored in memory.

**Key cache SELECT restriction**

If you are using the Update/Insert or Update refresh type, selecting this option specifies the WHERE clause to restrict the number of key value combinations cached.

By default, Data Manager reads the key value combinations for all rows in the target table. For very large tables this can be problematic, so you can specify a restriction to limit the values read. For example, you could enter a date range or a key range.

**Key cache Cognos SQL restriction**

If you are using the Update/Insert or Update refresh type, and have entered a restriction clause in the Key cache SELECT restriction box, selecting this check box specifies that the restriction is Cognos® SQL.
Appendix A: Fact Delivery Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit interval</td>
<td>Specifies (in terms of number of rows updated/inserted), the interval between commits of data in the target table. By default, Data Manager does not issue a COMMIT instruction until it has delivered all the data rows. That is, it executes the entire delivery as a single transaction. You must decide whether this represents an unacceptably large or long transaction.</td>
</tr>
<tr>
<td>Share database connection</td>
<td>If a fact build includes more than one relational table delivery that delivers to the same database connection, then selecting this option specifies that each delivery should share the database connection and transactions.</td>
</tr>
<tr>
<td></td>
<td>Sharing a database connection may be required to reduce the number of connections to a database, or to deliver related tables where a database constraint exists between those tables.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The lowest specified commit interval on the deliveries is used for the transactions on the shared database connection.</td>
</tr>
</tbody>
</table>

**Update/Insert Delivery Method**

The Update/Insert delivery method allows you to control the way in which existing fact records are updated and new fact records are inserted in the target table. This provides more efficient delivery of fact records where it is not known if a row with the same key exists in the target table.

You can set up the processing method that best suits the profile of your data by configuring

- the update detection method
- whether or not incoming record keys are unique
- which columns should be updated in the target table
- the control attributes

Using different configurations can result in

- increased efficiency when writing records that already exist in the target table
- fewer unnecessary updates of records
- improved database performance due to fewer logged database operations during loading
- reduced complexity and maintenance overhead of update and insert loads

While it is possible to improve performance and efficiency when updating or inserting data, the overall performance also depends upon other factors, such as the logging options you choose when executing builds, and whether or not your tables are indexed.

**Note:** It is recommended that you use indexes on tables as this dramatically improves performance when using this delivery method.
Primary Key Settings for Update/Insert

IBM® Cognos® Data Manager uses the primary key settings to override the way in which the Relational Table delivery module identifies unique records. These key columns are then used to form the WHERE clause for an update.

Note: If null values are allowed in the dimension elements, then the update may fail to find records.

The update operation is constrained by an SQL WHERE clause formed as follows:

- Columns marked as keys are used.
- If no key is used and there are no dimension elements in the transformation model, then Data Manager always performs an insert.

For information on specifying key columns, see "Define the Properties for a Table Delivery" (p. 209).

Update Detection Method for Update/Insert

The way in which IBM® Cognos® Data Manager updates or inserts data in the target table depends upon the update detection method you choose:

- **Update**
  Data Manager attempts to update every row in the target table.
  If the delivery record has the same key as a row that exists in the target table, Data Manager performs the update. If there is no match, Data Manager appends the delivery record to the target table.

- **Select**
  Data Manager searches the target table for a row with the same key as the delivery record.
  If a match is found, Data Manager updates the row. If there is no match, Data Manager appends the delivery record to the target table.

- **Select/Compare**
  Data Manager searches the target table for a row with the same key as the delivery record.
  If a match is found, Data Manager compares each column in the delivery record to the row that exists in the target table. If any columns differ, Data Manager updates them. Any identical columns remain unchanged. If no match is found, Data Manager appends the delivery record to the target table.
  
  - Select/Compare is only available if you have selected the Incoming record keys are unique check box.
  
  - If Select/Compare is selected, and you later clear the Incoming record keys are unique check box, the upgrade detection method is automatically changed to Select. The reason for this is that if the keys are not unique, when Data Manager updates a row, it does not attempt to compare individual columns. Instead it updates the entire row using the Select update detection method.

- **Key Cache**
Data Manager reads back the values of the key columns, and if appropriate, the record identity column, from the target table. It then caches them to either memory or to file depending on what you specified in the Key Cache max memory (MB) box. Data Manager then searches the cached keys for a row with the same key as the delivery record. If a match is found, Data Manager updates the row. If no match is found, the delivery record is appended to the target table.

For more information, see "Unique Incoming Record Keys for Update/Insert" (p. 479).

**Guidelines For Deciding Which Update Detection Method to Use**

The decision on which update detection method to use is determined by the content of the data to be delivered. It is recommended that you use:

- Update if most delivery rows already exist in the target table, but need to be updated
- Select if most delivery rows contain new data so will need to be inserted rather than updated
- Select/Compare if
  - the delivery data contains unique incoming record keys
  - most delivery rows already exist in the target table and there are minimal changes to the data
  - you want individual columns to be updated rather than the whole row
- Key Cache in preference to Select if performance is an issue and either sufficient memory is available for a key cache in memory, or an appropriate restriction can be used to restrict the number of key value combinations for a key cache on disk.

For more information about unique incoming record keys, see "Unique Incoming Record Keys for Update/Insert" (p. 479).

The following diagram illustrates, in percentage terms, approximately how your data should be split for each recommended method.
Performance Considerations

You should take into consideration how each update detection method can affect performance:

- Whilst Update and Select perform the same task, performance may be faster if you use Select because IBM® Cognos® Data Manager does not attempt to update every row.

- Select/Compare is generally slower than Select because Data Manager must check individual columns in matching rows. However, if you are executing builds across a network, performance is more likely to be improved using Select/Compare.

- Key cache should be used in preference to Select if performance is an issue and either sufficient memory is available for a key cache in memory, or an appropriate restriction can be used to restrict the number of key value combinations for a key cache on disk.

Unique Incoming Record Keys for Update/Insert

If your delivery data does not contain multiple records to update a single row in the target table, then the incoming record keys are unique. For example, in the following table, the incoming record keys are unique, because there is only a single delivery record for each product ID.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>500</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>400</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>300</td>
<td>400</td>
</tr>
</tbody>
</table>

For databases that support array insert, if the incoming record keys are unique in the delivery data, selecting this option considerably improves performance when updating rows in the target table.
Appendix A: Fact Delivery Module Properties

Notes

- If unique keys exist in the delivery data, and you want to use the Select/Compare update detection method, ensure that you select the Incoming record keys are unique option. For more information, see "Update Detection Method for Update/Insert" (p. 477).

- If unique keys exist in the delivery data, and the target table is empty, IBM® Cognos® Data Manager does not attempt to update any rows, it just inserts them.

Updated Columns

By default, when IBM® Cognos® Data Manager performs an update, all columns in the target table are updated, with the exception of any columns identified as key columns.

There may be certain columns that you do not want to update in the target table. Excluding them from update reduces processing time and improves performance.

You can specify which columns to update, and which to exclude from update, in the Table Delivery Properties window.

Control Attributes

You can use the following control attributes in a fact delivery table to track when rows are updated or inserted into the target table:

- **Create Date** specifies the date on which a new row was inserted into the target table

- **Last Update Date** specifies the date on which an existing row was updated

The Create Date attribute is inserted once in the target table, and never changes. The way in which Last Update Date changes depends on the update detection method you are using:

- If you are using Update or Select detection methods, IBM® Cognos® Data Manager sets the Last Update Date each time it performs an update, even if no data has changed in a row. This happens because Data Manager refreshes the entire row without checking individual columns.

- If you are using Select/Compare, Data Manager only sets the Last Update Date when specific columns in a row have changed.

  **Note:** You must use this update detection method if you want to ensure the Last Update Date is always set accurately.

For more information on the update detection method, see "Update Detection Method for Update/Insert" (p. 477).

For information on adding control attributes to a fact delivery table, see "Define the Properties for a Table Delivery" (p. 209).
**Update/Insert Delivery Example**

Suppose a fact delivery table contains the following data, where Product ID is defined as the key, and the shaded columns are selected for update.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>500</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>400</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>Black pen</td>
<td>200</td>
<td>350</td>
</tr>
<tr>
<td>1</td>
<td>Red pen</td>
<td>100</td>
<td>1500</td>
</tr>
</tbody>
</table>

Note that there are two delivery records for product ID 1, in other words incoming record keys are not unique.

The target table into which the data will be delivered, currently contains the following rows. Control attributes are included in the target table.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>700</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>205</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

When you load the fact data using the Update or Select update detection method, IBM® Cognos® Data Manager performs four updates (row 1 is updated twice) and one insert. The target table looks as follows.
Notice that

- The Forecast Sales column has not been updated for any existing row in the target table because it is excluded from update.

- Rows 1, 2, and 3 have all been updated (indicated by blue shading). Even though the data in row 2 has not changed, Data Manager has refreshed the row with data from the fact delivery table.

- Row 4 is a new row that has been inserted (indicated by yellow shading).

- Row 10 remains unchanged.

- The Last Update Date is set for every row that has been updated or inserted.

Now suppose a fact delivery table contains the following data. Product ID is defined as the key, shaded columns are selected for update, and incoming record keys are unique.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>Black pen</td>
<td>200</td>
<td>350</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

The target table into which the data will be delivered, currently contains the following rows. Control attributes are included in the target table.
When you load the fact data into the target table using the Select/Compare update detection method (the Incoming record keys are unique option is selected), Data Manager performs two updates and one insert. The target table looks like this:

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>700</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>205</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

When you load the fact data into the target table using the Select/Compare update detection method (the Incoming record keys are unique option is selected), Data Manager performs two updates and one insert. The target table looks like this:

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>150</td>
<td>01/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>400</td>
<td>01/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>4</td>
<td>Black pen</td>
<td>200</td>
<td>350</td>
<td>06/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

Notice that

- In rows 1 and 3, Data Manager has only updated the columns that have changed (indicated by blue shading).
- Row 4 is a new row that has been inserted (indicated by yellow shading).
- Row 10 remains unchanged.
- The Last Update Date is set only for those rows that have been updated or inserted.

**Update Delivery Method**

The Update delivery method allows you to control the way in which existing fact records are updated in the target table. You can improve the efficiency of updates to the target table by also specifying

- the update detection method
- which columns to update
- which control attributes to use
Using different configurations can result in

- increased efficiency when updating records
- fewer unnecessary updates of records
- improved database performance due to fewer logged database operations during loading
- reduced complexity and maintenance overhead of update loads

While it is possible to improve performance and efficiency when updating data, the overall performance also depends upon other factors, such as the logging options you choose when executing builds, and whether or not your tables are indexed.

**Note:** It is recommended that you use indexes on tables as this dramatically improves performance when using this delivery method.

### Primary Key Settings for Update

IBM® Cognos® Data Manager uses the key settings to override how the Relational Table delivery module identifies unique records. These key columns are then used to form the WHERE clause on the update.

**Note:** If null values are allowed in the dimension elements, then the update may fail to find records.

The update operation is constrained by an SQL WHERE clause formed as follows:

- Columns marked as keys are used.
- If no key is used and there are no dimension elements in the transformation model, then Data Manager produces an error message and fails.

For information on specifying key columns, see "Define the Properties for a Table Delivery" (p. 209).

### Update Detection Method for Update

The way in which IBM® Cognos® Data Manager updates data in the target table depends upon the update detection method you choose:

- **Update**
  
  Data Manager attempts to update every row in the target table.
  
  If the delivery record has the same key as a row that exists in the target table, Data Manager performs the update. If there is no match, Data Manager discards the delivery record.

- **Select**
  
  Data Manager searches the target table for a row with the same key as the delivery record.
  
  If a match is found, Data Manager updates the row. If there is no match, Data Manager discards the delivery record.

- **Select/Compare**
  
  Data Manager searches the target table for a row with the same key as the delivery record.
If a match is found, Data Manager compares each column in the delivery record to the row that exists in the target table. If any columns differ, Data Manager updates them. Any identical columns remain unchanged. If there is no match, Data Manager discards the delivery record.

- **Select/Compare** is only available if you have selected the Incoming record keys are unique check box.

- If Select/Compare is selected, and you later clear the Incoming record keys are unique check box, the upgrade detection method is automatically changed to Select. The reason for this is that if the keys are not unique, when Data Manager updates a row, it does not attempt to compare individual columns. Instead it updates the entire row using the Select update detection method.

- **Key Cache**

  Data Manager reads back the values of the key columns, and if appropriate, the record identity column, from the target table. It then caches them to either memory or to file depending on what you specified in the Key Cache max memory (MB) box. Data Manager then searches the cached keys for a row with the same key as the delivery record. If a match is found, Data Manager updates the row. If no match is found, Data Manager discards the delivery record.

**Guidelines For Deciding Which Update Detection Method to Use**

The decision on which update detection method to use is determined by the content of the data to be delivered. It is recommended that you use

- **Update** if most delivery rows that exist in the target table need to be updated

- **Select** if
  - most delivery rows already exist in the target table and there are minimal changes to the data
  - the delivery data does not contain unique incoming record keys

- **Select/Compare** if
  - most delivery rows already exist in the target table and there are minimal changes to the data
  - you want individual columns to be updated rather than updating the whole row

- **Key Cache** in preference to **Select** if performance is an issue and either sufficient memory is available for a key cache in memory, or an appropriate restriction can be used to restrict the number of key value combinations for a key cache on disk.

The following diagram illustrates, in percentage terms, approximately how your data should be split for each recommended method.
Performance Considerations
You should take into consideration how each update detection method can affect performance:

- While Update and Select perform the same task, performance may be faster if you use Select because IBM® Cognos® Data Manager does not attempt to update every row.

- Select/Compare is generally slower than Select because Data Manager must check individual columns in matching rows. However, if you are executing builds across a network, performance is more likely to be improved using Select/Compare.

- Key cache should be used in preference to Select if performance is an issue and either sufficient memory is available for a key cache in memory, or an appropriate restriction can be used to restrict the number of key value combinations for a key cache on disk.

Update Delivery Example
Suppose a fact delivery table contains the following data, where Product ID is defined as the key, and the shaded columns are selected for update.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>500</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>400</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>Black pen</td>
<td>200</td>
<td>350</td>
</tr>
<tr>
<td>1</td>
<td>Red pen</td>
<td>100</td>
<td>1500</td>
</tr>
</tbody>
</table>
Note that there are two delivery records for product ID 1, in other words incoming record keys are not unique.

The target table into which the data will be delivered, currently contains the following rows. Control attributes are included in the target table.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>700</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>205</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

When you load the fact data using the Update or Select update detection method, IBM® Cognos® Data Manager performs four updates (row 1 is updated twice). The target table looks as follows.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>1500</td>
<td>01/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>400</td>
<td>01/01/2005</td>
<td>06/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

Notice that

- The Forecast Sales column has not been updated for any existing row in the target table because it is excluded from update.
- Rows 1, 2, and 3 have all been updated (indicated by blue shading). Even though the data in row 2 has not changed, Data Manager has refreshed the row with data from the fact delivery table.
- Row 4 in the delivery data has been discarded because Data Manager could not find a match in the target table.
- Row 10 remains unchanged.
- The Last Update Date is set for every row that has been updated.

Now suppose a fact delivery table contains the following data. Product ID is defined as the key, shaded columns are selected for update, and incoming record keys are unique.
The target table into which the data will be delivered, currently contains the following rows. Control attributes are included in the target table.

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>500</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>400</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>300</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Black pen</td>
<td>200</td>
<td>350</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When you load the fact data into the target table using the Select/Compare update detection method (the Incoming record keys are unique option is selected), Data Manager performs two updates. The target table looks like this:

<table>
<thead>
<tr>
<th>Product ID</th>
<th>Specification</th>
<th>Forecast Sales</th>
<th>Actual Sales</th>
<th>Create Date</th>
<th>Last Update Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red pen</td>
<td>0</td>
<td>700</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>2</td>
<td>Blue pen</td>
<td>0</td>
<td>200</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>3</td>
<td>Green pen</td>
<td>0</td>
<td>205</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
<tr>
<td>10</td>
<td>Orange pen</td>
<td>0</td>
<td>50</td>
<td>01/01/2005</td>
<td>01/01/2005</td>
</tr>
</tbody>
</table>

Notice that

- In rows 1 and 3, Data Manager has only updated the columns that have changed (indicated by blue shading).
- Row 10 remains unchanged.
- The Last Update Date is set for only those rows that have been updated.
Element Properties

The properties available depend on the type of element that you are defining.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Additive update</td>
<td>When updating a table, specifies whether to add new values to the values that exist in the table (rather than replace them). This property does not apply to dimensions.</td>
</tr>
<tr>
<td>Maximum UTF-8 bytes per character</td>
<td>When using Key Cache for key elements that comprise character data, this can be used to reduce the size of the cache. By default, IBM® Cognos® Data Manager calculates the size of such elements by multiplying the length of the target column by this value (default 4). For example, if it is known that the values of the element are always composed of ASCII or numeric characters, then the value should be specified as 1.</td>
</tr>
</tbody>
</table>

SQL Server Bulk Copy via API Delivery Module

The SQL Server Bulk Copy via API delivery module supports bulk copying to Microsoft® SQL Server and Sybase SQL Server. It uses a programmatic interface supplied by your DBMS vendor.

To bulk copy to Microsoft® SQL Server, use either a Microsoft® SQL Server ODBC connection or a Microsoft® SQL Server OLE-DB connection. To bulk copy to Sybase SQL Server, use a Sybase SQL Server connection.

**Note:** Depending on the type of database connection you are using, this delivery module uses different interfaces to the underlying database. For example, API delivery to Microsoft® SQL Server via ODBC is not the same as API delivery to Microsoft® SQL Server via OLE-DB. You should consider any performance implications as a result of changing the interface type.
Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load mode</td>
<td>Specifies one of these load modes:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Append</strong> appends the data to the target table. (default)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Truncate</strong> deletes the contents of the target table and drops all indexes before delivering the data. This is equivalent to a DELETE FROM statement but is usually faster.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Replace</strong> deletes the contents of the target table but not the indexes, before delivering the data. This is a DELETE FROM statement.</td>
</tr>
<tr>
<td>Batch size</td>
<td>Specifies the number of rows to be copied in a batch. By default, the delivery commits all the rows in one batch.</td>
</tr>
<tr>
<td>Bulk copy hints</td>
<td>Use to set load hints ROWS_PER_BATCH, KILOBYTES_PER_BATCH, TABLOCK, CHECK_CONSTRAINTS, and ORDERS.</td>
</tr>
<tr>
<td>Treat empty string as null value</td>
<td>Specifies whether empty string values should be read as null values.</td>
</tr>
</tbody>
</table>

Element Properties

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
</tbody>
</table>

Sybase ASE BCP Command Delivery Module

The Sybase ASE BCP Command delivery module delivers data directly to Sybase ASE databases using the Sybase ASE Server BCP bulk load utility.

The BCP command utility is a component of Sybase ASE client installation. It provides users with an intermediate load file that is passed to the BCP utility.
## Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader</strong> to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader (leave files)</strong> to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>● <strong>Create Loader Files</strong> to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td>Tip: To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create Loader Files</strong> is selected in the <strong>Operation</strong> box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td><strong>Log file success message</strong></td>
<td>If <strong>No row count check</strong> is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
</tbody>
</table>
### Property Description

**Record delimiter**
Specifies the character sequence to use to separate records.
Specify a character string or use one of these: Tab, Comma, Space, None, NL (new line), CRLF (carriage return and line feed combination) on a Microsoft® Windows® operating system, or CR (carriage return) on a UNIX® operating system. NL is the default.

**Field delimiter**
Specifies the character sequence to use to separate fields.
Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).

**Load mode**
Specifies one of these load modes:
- **Append** appends the data to the target table. (default)
- **Truncate** deletes the contents of the target table, and drops all indexes, before delivering the data. This is equivalent to a DELETE FROM statement, but is usually faster.
- **Replace** deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.
- **Insert** inserts the data in the target table.

**Data file encoding**
Specifies the data file encoding to use.

**Sybase ASE BCP command**
Specifies the path and command for the loader process.
*Note:* You must specify the exact path name and command, including any double quotation marks required.
For more information, see "Specify Executables to Deliver Data" (p. 511).

**Additional command line options**
Allows additional BCP command line options to be specified.

---

### Element Properties

For this delivery module, the element properties are the same for all element types.
**Value if null**
Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.

**Floating point scale**
Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.

---

**Sybase IQ LOAD Delivery Module**

The Sybase IQ LOAD delivery module delivers data directly to Sybase IQ databases.

If IBM® Cognos® Data Manager is installed on a different computer than the database from which you are loading data, the directory containing the database must be shared and the database must have read and write access.

**Module Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader</strong> to deliver the text data and control files, load the data</td>
</tr>
<tr>
<td></td>
<td>into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader (leave files)</strong> to deliver the text data and control files,</td>
</tr>
<tr>
<td></td>
<td>load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Create Loader Files</strong> to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>File directory</td>
<td>Specifies the location in which to load data. If Data Manager is installed on a different computer to the database, you must include the server location using this format \servername\file_directory\ If Data Manager is installed on the same computer as the database, by default, Data Manager loads data into its Data directory. <strong>Tip:</strong> To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td>Filename prefix</td>
<td>If Create Loader Files is selected in the <strong>Operation</strong> box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td>Dump log file</td>
<td>Specifies whether to print the log file that the bulk loader generates. <strong>Note:</strong> Log files are stored on the Sybase IQ server.</td>
</tr>
<tr>
<td>Use named pipes</td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data. <strong>Note:</strong> You can only use named pipes when Data Manager is installed on the same computer as the database.</td>
</tr>
<tr>
<td>Record delimiter</td>
<td>Specifies the character sequence to use to separate records. Specify a character string or use one of these: Tab, Comma, Space, None, NL (new line), CRLF (carriage return and line feed combination) on a Microsoft® Windows® operating system, or CR (carriage return) on a UNIX® operating system. NL is the default.</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
<tr>
<td>Load mode</td>
<td>Specifies one of these load modes: <strong>Insert</strong> inserts the data in the target table. <strong>Replace</strong> deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.</td>
</tr>
</tbody>
</table>
## Property Description

**Data file encoding**
Specifies the data file encoding to use.

**Additional options**
Specifies a string of extra load options to append to the command.

**Sybase CLP command**
Specifies the path and command for the Sybase IQ loader process.

*Note:* You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

**Sybase quotes option**
Specifies whether to use quotation marks to enclose data values. By default, no quotation marks are used.

### Element Properties

The element properties available depend on the type of element being defined.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value if null</strong></td>
<td>Specifies the value to represent null data values in the element.</td>
</tr>
<tr>
<td></td>
<td>This is necessary when the column in the target table does not directly</td>
</tr>
<tr>
<td></td>
<td>support null. By default, this value is a blank string for character values</td>
</tr>
<tr>
<td></td>
<td>and 0 for numeric values.</td>
</tr>
<tr>
<td><strong>Perform escape processing</strong></td>
<td>Specifies whether to use the escape character () before all</td>
</tr>
<tr>
<td></td>
<td>backslash, tab, and new line characters in string values to ensure</td>
</tr>
<tr>
<td></td>
<td>that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td><strong>Floating point scale</strong></td>
<td>Specifies the number of decimal places for a numeric value that must be</td>
</tr>
<tr>
<td></td>
<td>converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

## Teradata Fastload Delivery Module

The Teradata Fastload delivery module delivers data directly to Teradata Fastload databases.

*Note:* You can access Teradata on a UNIX® operating system using an ODBC driver. However, you must ensure that, in the ODBC.ini file, you use the data source name that you used on a Microsoft® Windows® operating system.
### Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader</strong> to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>● <strong>Run Loader (leave files)</strong> to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>● <strong>Create Loader Files</strong> to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td>Tip: To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create Loader Files</strong> is selected in the <strong>Operation</strong> box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td><strong>Log file success message</strong></td>
<td>If <strong>No row count check</strong> is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td><strong>Use named pipes</strong></td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
<tr>
<td>Load mode</td>
<td>Specifies one of these load modes:</td>
</tr>
<tr>
<td></td>
<td>• Append appends the data to the target table. (default)</td>
</tr>
<tr>
<td></td>
<td>• Truncate deletes the contents of the target table, and drops all indexes, before delivering the data. This is equivalent to a DELETE FROM statement, but is usually faster.</td>
</tr>
<tr>
<td></td>
<td>• Replace deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.</td>
</tr>
<tr>
<td>Data file encoding</td>
<td>Specifies the data file encoding to use.</td>
</tr>
<tr>
<td>Max sessions</td>
<td>Specifies the maximum number of loader sessions to use. If no value is specified, the Fastload default is used.</td>
</tr>
<tr>
<td>Min sessions</td>
<td>Specifies the minimum number of loader sessions to use. If no value is specified, the Fastload default is used.</td>
</tr>
<tr>
<td>Error file name</td>
<td>Specifies an alternate file name for Fastload error messages. This produces a duplicate record of all Fastload error messages.</td>
</tr>
<tr>
<td>Error limit</td>
<td>Specifies the maximum number of records that can be rejected before the loader stops. If no value is specified, the Fastload default is used.</td>
</tr>
<tr>
<td>Verbose</td>
<td>Specifies whether to print every request sent to the Teradata Database.</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Specifies whether the session is to be encrypted.</td>
</tr>
<tr>
<td>Additional options</td>
<td>Specifies a string of extra load options to append to the command. ERRORFILES specifies the file names to use for error tables. The default file names are fload_error1 and fload_error2. This option is mandatory.</td>
</tr>
</tbody>
</table>
### Element Properties

The element properties available depend on the type of element being defined.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value if null</strong></td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td><strong>Perform escape processing</strong></td>
<td>Specifies whether to use the escape character () before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td><strong>Floating point scale</strong></td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

### TM1 Turbo Integrator Delivery Module

The TM1® Turbo Integrator delivery module can deliver data directly to IBM® Cognos® TM1® cubes. Alternatively, it can be used to create flat files for loading into TM1® cubes. In this instance, it also creates the TM1® process required to load data and runs the process, if required.

You can deliver fact data and dimension data.

IBM® Cognos® Data Manager dimension elements are delivered as dimensions into a TM1® cube. You can also deliver other Data Manager transformation model elements as elements of a TM1® measures dimension. In this case, you must specify the name of the TM1® measures dimension to use for delivery.

Alternatively, a single non-dimensional transformation model element must be delivered. In this case, you should not specify any TM1® measures dimension name.
**Installation Considerations**

The TM1® client and Data Manager must be installed on the same computer. For a distributed installation, the TM1® client, Data Manager, and the Data Manager engine must be installed on the same computer.

For Data Manager to find the required libraries, you must add a TM1® installation bin directory.

To install the TM1® server locally, use TM1® 9.4 and accept the default values.

When the installation is complete, ensure that the TM1® service has started, and then start Data Manager.

**Notes**

- Currently, you can only deliver to TM1® using Data Manager on a Microsoft® Windows® operating system.
- You can ignore specified minor errors when delivering data to TM1® by setting the variable `DM_TM1_LOAD_SUPPRESS_ERROR_CODES`. For more information, see "DM_TM1_LOAD_SUPPRESS_ERROR_CODES" (p. 326).
- If you decide to ignore minor errors, information about the number of rows successfully delivered may not be accurate.

**Module Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run TM1 Process</strong> to deliver the data files and the TM1® process, load</td>
</tr>
<tr>
<td></td>
<td>the data into the cube and clean up the process and any other intermediate</td>
</tr>
<tr>
<td></td>
<td>files. (default)</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If the Direct delivery check box is selected, data files</td>
</tr>
<tr>
<td></td>
<td>are purged but not deleted from the TM1® server.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run and Leave TM1 Process</strong> to deliver the data files and the TM1®</td>
</tr>
<tr>
<td></td>
<td>process, load the data into the cube but leave the process and any other</td>
</tr>
<tr>
<td></td>
<td>intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Create TM1 Process</strong> to create the data files and the TM1® process,</td>
</tr>
<tr>
<td></td>
<td>but not to load the data into the cube.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>Specifies the connection for the TM1® server.</td>
</tr>
</tbody>
</table>
### Appendix A: Fact Delivery Module Properties

<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to write the data files. By default, Data Manager writes data into its Data directory.</td>
</tr>
<tr>
<td><strong>Tip:</strong></td>
<td>To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>If the <strong>Direct delivery</strong> check box is selected, you do not need to specify a file directory.</td>
</tr>
<tr>
<td><strong>File directory on server</strong></td>
<td>Specifies the server location of the file directory. This information is required only if Data Manager is installed on a different computer to the TM1® server.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>If the <strong>Direct delivery</strong> check box is selected, you do not need to specify a file directory.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create TM1 Process</strong> is selected in the <strong>Operation</strong> box, specify the file name to use for the data files and process. If you do not specify a file name, Data Manager uses a default name.</td>
</tr>
<tr>
<td><strong>Direct delivery</strong></td>
<td>Specifies whether to deliver data directly to the TM1® server rather than creating flat files for loading. Select this check box to deliver data directly to TM1®.</td>
</tr>
<tr>
<td><strong>Process name on server</strong></td>
<td>Specifies the name to use for the TM1® process. If you do not specify a name, Data Manager uses a unique name by default.</td>
</tr>
<tr>
<td><strong>Cube name</strong></td>
<td>Specifies the name of the TM1® cube to create or update. If you do not specify a cube name, Data Manager ignores the specified cube action and data action.</td>
</tr>
<tr>
<td><strong>Cube action</strong></td>
<td>Specifies the cube action to take:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Create</strong> to create a cube for the delivered data. (default)</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The specified cube name must not already exist.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Recreate</strong> to delete an existing cube and recreate it.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Update</strong> to deliver the data to an existing cube.</td>
</tr>
<tr>
<td></td>
<td>When this is selected, for all dimensions (including a measures dimension), any dimension action that is set to <strong>Create</strong> or <strong>Update</strong> is treated as an update.</td>
</tr>
<tr>
<td></td>
<td>• <strong>NoAction</strong> to not create or update a cube.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Truncate</strong> to reset all values in an existing cube to zero.</td>
</tr>
</tbody>
</table>
### Property | Description
--- | ---
**View name** | If **Truncate** is selected in the **Cube action** box, specify which view name to truncate in the cube. If you do not specify a view name, by default, Data Manager truncates the whole cube using a default view name.  
**Data action** | Specifies whether TM1® should  
- **Accumulate values** to deliver the delivery data values by accumulating them to existing data values.  
- **Store values** to deliver the delivery data values by overwriting existing data values. (default)  
**Measure dimension name** | Specifies the name to use for the measure dimension in the cube. If you do not specify a name, Data Manager ignores the specified measures dimension action.  
**Measure dimension action** | Specifies the dimension action to take for the measure:  
- **Create** to create a measures dimension for the delivered data. (default)  
- **Recreate** to delete an existing measures dimension and recreate it.  
- **Update** to deliver the data to an existing measures dimension.  
- **NoAction** to not create or update a measures dimension.  
**Floating point scale** | Specifies the number of decimal places (for a numeric value) which must be converted to text. By default, this value is three decimal places.  
**TM1 prolog script** | You can add to the TM1® prolog scripts if required.  
**Tip:** Click the browse button to type your script amendments in a text window.  
**TM1 epilog script** | You can add to the TM1® epilog scripts if required.  
**Tip:** Click the browse button to type your script amendments in a text window.

### Element Properties
The element properties available depend on the type of element being defined.
## Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TM1 dimension name</strong></td>
<td>Specifies the name of the TM1® dimension that corresponds to the IBM® Cognos® Data Manager dimensions.</td>
</tr>
</tbody>
</table>
| **Dimension action** | Specifies the dimension action to take:  
  - **Create** to create a dimension for the delivered data. (default)  
  - **Recreate** to delete an existing dimension and recreate it.  
  - **Update** to deliver the data to an existing dimension.  
  - **NoAction** to not create or update a dimension. |
| **TM1 element name expression** | An expression that specifies how to construct the element names from the TM1® dimension. By default, Data Manager uses ID. This value is an expression, for example, `concat(CAPTION, '(', ID, ',')')`. This value is mandatory. |
| **Caption TM1 attribute name** | Specifies the name of the TM1® attribute that corresponds to Data Manager dimension caption attribute (if one exists). |
| **Caption TM1 attribute expression** | An expression that specifies how to construct the TM1® attribute names from the TM1® dimension. By default, Data Manager uses CAPTION. This value is an expression, for example, `concat(CAPTION, '(', ID, ',')')`. |
| **Caption TM1 attribute type** | Specifies the TM1® attribute type to use for captions:  
  - **String** a string attribute. (default)  
  - **Alias** a special attribute type. |

---

### Teradata Multiload Delivery Module

The Teradata Multiload delivery module delivers data directly to Teradata Multiload databases.  
**Note:** You can access Teradata on a UNIX® operating system using an ODBC driver. However, you must ensure that, in the ODBC.ini file, you use the data source name that you used on a Microsoft® Windows® operating system.
# Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td><strong>Run Loader</strong></td>
<td>to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td><strong>Run Loader (leave files)</strong></td>
<td>to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td><strong>Create Loader Files</strong></td>
<td>to deliver the text data and control files, but not invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td><strong>Tip:</strong></td>
<td>To view the Data directory, from the Tools menu, click Browse Data Files.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If Create Loader Files is selected in the Operation box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td><strong>Log file success message</strong></td>
<td>If No row count check is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td><strong>Use named pipes</strong></td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
<tr>
<td>Load mode</td>
<td>Specifies one of these load modes:</td>
</tr>
<tr>
<td></td>
<td>- <strong>Append</strong> appends the data to the target table. (default)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Truncate</strong> deletes the contents of the target table, and drops all indexes, before delivering the data. This is equivalent to a DELETE FROM statement, but is usually faster.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Replace</strong> deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.</td>
</tr>
<tr>
<td>Restart log table</td>
<td>Specifies a log table file name. Multiload uses the log table to restart a load from the last checkpoint if the load failed because the error limit was reached.</td>
</tr>
<tr>
<td>Data file encoding</td>
<td>Specifies the data file encoding to use.</td>
</tr>
<tr>
<td>Max sessions</td>
<td>Specifies the maximum number of loader sessions to use. If no value is specified, the Multiload default is used.</td>
</tr>
<tr>
<td>Min sessions</td>
<td>Specifies the minimum number of loader sessions to use. If no value is specified, the Multiload default is used.</td>
</tr>
<tr>
<td>Error file name</td>
<td>Specifies an alternate file name for Multiload error messages. This produces a duplicate record of all Multiload error messages.</td>
</tr>
<tr>
<td>Brief</td>
<td>Specifies that run-time output messages should be limited to the minimum required to determine whether the load is successful.</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Specifies whether the session is to be encrypted.</td>
</tr>
<tr>
<td>Additional options</td>
<td>Specifies a string of extra load options to append to the command.</td>
</tr>
<tr>
<td>Route messages to (filename)</td>
<td>Specifies that run-time output messages should be routed to an alternative file.</td>
</tr>
</tbody>
</table>
**Description Property**
Specifies the path and command for the Multiload loader process.

*Note:* You must specify the exact path name and command, including any double quotation marks required.

For more information, see "Specify Executables to Deliver Data" (p. 511).

---

## Element Properties

The element properties available depend on the type of element being defined.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Perform escape processing</td>
<td>Specifies whether to use the escape character () before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>

---

### Teradata TPump Delivery Module

The Teradata TPump delivery module delivers data directly to Teradata TPUMP databases.

*Note:* You can access Teradata on a UNIX® operating system using an ODBC driver. However, you must ensure that, in the ODBC.ini file, you use the data source name that you used on a Microsoft® Windows® operating system.
### Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation</strong></td>
<td>Specifies whether IBM® Cognos® Data Manager should</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader</strong> to deliver the text data and control files, load the data into the database and clean up the control files and any other intermediate files. (default)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Run Loader (leave files)</strong> to deliver the text data and control files, load the data into the database but leave the control files and any other intermediate files so that you can view them.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Create Loader Files</strong> to deliver the text data and control files, but not to invoke the loader to load the data into the database.</td>
</tr>
<tr>
<td><strong>File directory</strong></td>
<td>Specifies the location in which to load data. By default, Data Manager loads data into its Data directory.</td>
</tr>
<tr>
<td></td>
<td><strong>Tip:</strong> To view the Data directory, from the <strong>Tools</strong> menu, click <strong>Browse Data Files</strong>.</td>
</tr>
<tr>
<td><strong>Filename prefix</strong></td>
<td>If <strong>Create Loader Files</strong> is selected in the <strong>Operation</strong> box, specify the prefix to use to generate the file names. If this box is empty, a default value is used.</td>
</tr>
<tr>
<td><strong>Dump log file</strong></td>
<td>Specifies whether to print the log file that the bulk loader generates.</td>
</tr>
<tr>
<td><strong>No row count check</strong></td>
<td>Specifies whether to check for successful delivery of the data. Clear this check box to prevent an error if the target DBMS does not report successful delivery in the expected format. Select this check box to use a newer version of the DBMS where the vendor changed the format of the completion message.</td>
</tr>
<tr>
<td><strong>Log file success message</strong></td>
<td>If <strong>No row count check</strong> is not selected, this is used to determine whether the load was successful by searching for the specified substring in the log file or output file. Data Manager searches for this string and the number of rows delivered as substrings in each line of the log file or output file.</td>
</tr>
<tr>
<td><strong>Use named pipes</strong></td>
<td>Specifies that Data Manager should use named pipes where possible. By default, Data Manager generates an intermediate text file and runs the bulk loader on it after delivering all data.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Field delimiter</td>
<td>Specifies the character sequence to use to separate fields. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
<tr>
<td>Load mode</td>
<td>Specifies one of these load modes:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Append</strong> appends the data to the target table. (default)</td>
</tr>
<tr>
<td></td>
<td>• <strong>Truncate</strong> deletes the contents of the target table, and drops all indexes, before delivering the data. This is equivalent to a DELETE FROM statement, but is usually faster.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Replace</strong> deletes the contents of the target table but not the indexes, before delivering the data. This is equivalent to a DELETE FROM statement.</td>
</tr>
<tr>
<td>Restart log table</td>
<td>Specifies a log table file name. TPump uses the log table to restart a load from the last checkpoint if the load failed because the error limit was reached.</td>
</tr>
<tr>
<td>Data file encoding</td>
<td>Specifies the data file encoding to use.</td>
</tr>
<tr>
<td>Control file encoding</td>
<td>Specifies the control file encoding to use.</td>
</tr>
<tr>
<td>Error file name</td>
<td>Specifies an alternate file name for TPump error messages. This produces a duplicate record of all TPump error messages.</td>
</tr>
<tr>
<td>Config file name</td>
<td>Specifies the configuration file that contains configuration and tuning parameters for TPump.</td>
</tr>
<tr>
<td>Brief</td>
<td>Specifies that run-time output should be limited to the minimum required to determine whether the load is successful.</td>
</tr>
<tr>
<td>Verbose</td>
<td>Specifies whether additional statistical data is required.</td>
</tr>
<tr>
<td>Encrypt</td>
<td>Specifies whether the session is to be encrypted.</td>
</tr>
<tr>
<td>Additional options</td>
<td>Specifies a string of extra load options to append to the command.</td>
</tr>
<tr>
<td></td>
<td>SESSIONS specifies the number of TPump sessions to use.</td>
</tr>
<tr>
<td>Route messages to (filename)</td>
<td>Specifies that run-time output messages should be routed to an alternative file.</td>
</tr>
</tbody>
</table>
## Text File Delivery Module

The Text File delivery module formats build data into simple text files. These files can then be imported into spreadsheets for crosstab (pivot table) analysis, or can be distributed to other systems.

**Note:** You can also deliver to text files that have been defined using IBM® Cognos® Data Manager SQLTXT Designer. This gives you more control over the format of the text file, although it is more time consuming to set up. For more information on using Data Manager SQLTXT Designer, see "IBM Cognos Data Manager SQLTXT Designer" (p. 385).

### Element Properties

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value if null</td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td>Perform escape processing</td>
<td>Specifies whether to use the escape character () before all backslash, tab, and new line characters in string values to ensure that the delivery module correctly interprets the statement.</td>
</tr>
<tr>
<td>Floating point scale</td>
<td>Specifies the number of decimal places for a numeric value that must be converted to text. By default, this value is three decimal places.</td>
</tr>
</tbody>
</table>
## Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output filename</strong></td>
<td>Specifies the path and name of the file to receive the data. If you enter only a file name, IBM® Cognos® Data Manager adds the file to its Data directory. This value is mandatory.</td>
</tr>
<tr>
<td><strong>Line delimiter</strong></td>
<td>Specifies the character sequence to use to separate data rows in the output file.</td>
</tr>
<tr>
<td><strong>Default field delimiter</strong></td>
<td>Specifies the character sequence to use to separate fields in the output file.</td>
</tr>
<tr>
<td><strong>Default value if null</strong></td>
<td>Specifies the value to represent null data values. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
<tr>
<td><strong>Default date format</strong></td>
<td>Specifies the format to use to write date/time values to the output file.</td>
</tr>
<tr>
<td><strong>Default scale</strong></td>
<td>Specifies the number of decimal places to use for all numeric values in the output file.</td>
</tr>
<tr>
<td><strong>Include field titles</strong></td>
<td>Specifies whether to include field titles, containing the name of each element, as the first line of the output file. By default, field titles are not included.</td>
</tr>
<tr>
<td><strong>Quote fields</strong></td>
<td>Specifies whether to use double quotation marks to enclose field values. By default, no quotation marks are used.</td>
</tr>
</tbody>
</table>

**Tip:** To view the Data directory, from the **Tools** menu, click **Browse Data Files**.
## Appendix A: Fact Delivery Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encoding</strong></td>
<td>Specifies the encoding in which the text file is to be produced. (Default) is the local computer encoding, for example windows-1252 on a Microsoft® Windows® operating system in Western Europe. The others are standard Unicode encodings.</td>
</tr>
</tbody>
</table>

### Element Properties

For this delivery module, the element properties are the same for all element types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Field delimiter</strong></td>
<td>Specifies the character sequence to use to separate fields in the output file. Specify a character string or use one of these named characters: Tab (the default), Comma, Space, None, or NL (new line).</td>
</tr>
<tr>
<td><strong>Value if null</strong></td>
<td>Specifies the value to represent null data values in the element. This is necessary when the column in the target table does not directly support null. By default, this value is a blank string for character values and 0 for numeric values.</td>
</tr>
</tbody>
</table>
| **Print format**       | Specifies the format used to print values for an element in the output file. The syntax is 

\[-[0]<precision>].\[.<scale>]

The symbols represent the following:
- - left justify the field
- 0 leading zero padding for numeric values
- `<precision>` the minimum number of characters to write
- `<scale>` for numeric values, the maximum number of decimal places or character values to write

By default, IBM® Cognos® Data Manager applies no special formatting to character and long values. However, Data Manager writes double-precision floating point values with this format: 

.6
### Date Format

Specifies the format to use to write date/time values to the output file.

If Data Manager recognizes a value as a date or time, it formats the value in its native format. Otherwise, it formats the value in the default format for the originating database.

If the specified format is invalid for a particular value, Data Manager performs no formatting on that value.

For information about date formats, see "Date, Time, and Interval Formats" (p. 537).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Format</td>
<td>Specifies the format to use to write date/time values to the output file. If Data Manager recognizes a value as a date or time, it formats the value in its native format. Otherwise, it formats the value in the default format for the originating database. If the specified format is invalid for a particular value, Data Manager performs no formatting on that value. For information about date formats, see &quot;Date, Time, and Interval Formats&quot; (p. 537).</td>
</tr>
</tbody>
</table>

### Specify Executables to Deliver Data

For some delivery modules, you can specify the executable which IBM® Cognos® Data Manager uses to deliver the data. You may want to do this to use a non-default loader. For example, by default, Data Manager uses sqlldr to deliver to Oracle SQL*Net. However, Oracle 8 uses sqlldr80 and you would specify this loader for computers that use older versions of Oracle. You can specify executables for each:

- delivery, by setting a module property
- session, by setting an environment variable

When delivering data, Data Manager searches for the executable to use as follows:

**Steps**

1. Data Manager checks whether a delivery module command property is set. If found, this is used to deliver the data.

2. If no delivery module command property is set, Data Manager searches for the appropriate dm.ini entry or environment variable value. If found, the value is double-quoted as required, then used to deliver the data.
   
   **Note:** Environment variables take precedence over the dm.ini file entry.

3. If the two searches fail, Data Manager uses the default command.
   
   For a list of default commands, see "Set the Name of the Loader Executable in a Module Property" (p. 511).

### Set the Name of the Loader Executable in a Module Property

This section describes how to set the name of the loader executable in a module property.

**Steps**

1. Open the appropriate Delivery Properties window.
2. Click the **Module Properties** tab.

3. In the relevant **Command** box, enter the name of the required loader executable.
   
The following table contains default command names for each delivery module.

<table>
<thead>
<tr>
<th>Delivery module</th>
<th>Default command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB2® LOAD</strong></td>
<td>Microsoft® Windows® operating system:</td>
</tr>
<tr>
<td></td>
<td>db2cmd /c /w /i db2</td>
</tr>
<tr>
<td></td>
<td>UNIX® operating system: db2</td>
</tr>
<tr>
<td><strong>Informix® LOAD</strong></td>
<td>dbaccess</td>
</tr>
<tr>
<td><strong>Microsoft® SQL Server BCP Command</strong></td>
<td>bcp</td>
</tr>
<tr>
<td><strong>Netezza NZLoad</strong></td>
<td>nzload</td>
</tr>
<tr>
<td><strong>Oracle SQL® Loader</strong></td>
<td>sqlldr</td>
</tr>
<tr>
<td><strong>Red Brick® Loader (TMU)</strong></td>
<td>rb_tmu</td>
</tr>
<tr>
<td><strong>Sybase ASE BCP Command</strong></td>
<td>bcp</td>
</tr>
<tr>
<td><strong>Sybase IQ LOAD</strong></td>
<td>dbisql</td>
</tr>
<tr>
<td><strong>Teradata Fastload</strong></td>
<td>fastload</td>
</tr>
<tr>
<td><strong>Teradata Multiload</strong></td>
<td>mload</td>
</tr>
<tr>
<td><strong>Teradata TPump</strong></td>
<td>Windows®: tpump</td>
</tr>
<tr>
<td></td>
<td>UNIX®: tpmump</td>
</tr>
</tbody>
</table>

## Set the Name of the Executable in an Environment Variable

The following table gives the name of the environment variables and the associated delivery modules.

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Delivery module</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DS_DB2_LOADER</strong></td>
<td>DB2 UDB Microsoft® Windows®/UNIX® operating system bulk loader command</td>
</tr>
<tr>
<td><strong>DS_INFORMIX_LOADER</strong></td>
<td>Informix bulk loader command</td>
</tr>
<tr>
<td><strong>DS_MSSQLSERVER_BCP</strong></td>
<td>Microsoft® SQL Server BCP bulk loader command</td>
</tr>
</tbody>
</table>
Specify the Location of Executables

For each executable that you specify, you must set up its location on the computer on which you execute builds.

**Note:** For DB2® LOAD delivery module, ensure you use db2.exe, not db2cmd.exe.

**Steps**

1. Open the file named dm.ini which, by default, is located in Program Files\ibm\cognos\c10\datamanager

2. In the Program Locations section, type the locations of the required executables.

3. Save the file.
Appendix B: Database Drivers

The IBM® Cognos® Data Manager database drivers present a common interface to the Data Manager software. This allows you to treat all data sources in the same manner. In addition to performing common functions such as extracting data, Data Manager can

- connect to the required database
- convert the native data types of each database to and from its own (internal) data types
- convert the native date formats to and from the internal format that Data Manager uses

Data Manager supports these DBMS drivers:

- IBM® Cognos® Data Source
- DB2®
- DTS Package (using OLE-DB)
- Essbase
- INFORMIX® (using Connect)
- ODBC
- Oracle (using SQL*Net)
- Published FM Package
- Microsoft® SQL Server (using OLE-DB)
- SAP R/3
- SQLTXT
- Sybase (using CT-Library)
- TM1®

Notes

- Data Manager only enables those drivers that it detects are installed.
- Some databases impose limits on the length of table names. This may result in the names being truncated and possibly becoming duplicates.

Data Manager uses connections to connect to the required data sources. For more information, see "Connections" (p. 63).

For an up-to-date list of environments supported by IBM® Cognos® products, such as operating systems, browsers, Web servers, directory servers, database servers, OLAP servers and more, see http://www.ibm.com/software/data/support/cognos_crc.html.
**Data Types**

To provide consistency, Data Manager converts the native data types of each DBMS to its internal data types. This allows it to acquire, transform, and deliver data, irrespective of the source and destination DBMS.

For information on Data Manager internal data types, see "Data Types" (p. 543).

Data Manager can automatically convert most data types. However, unless mentioned in the DBMS Drivers section, other types are not supported unless they can be cast by the database into a type that Data Manager supports.

**IBM Cognos Data Source Driver**

The IBM® Cognos® Data Manager driver for IBM® Cognos® data sources provides a method of accessing any data source set up in IBM® Cognos® Business Intelligence that accepts Cognos® SQL. When the IBM® Cognos® BI environment has been installed and configured, these data sources become available for consumption as Data Manager sources.

Data type conversion follows the rules of the driver that the data source represents.

**DB2 DBMS Driver**

**Data Type Conversions**

The conversion between DB2® data types and IBM® Cognos® Data Manager data types varies slightly depending on the version of DB2® and the operating system on which it runs.

You can discover the conversions for a particular DB2® data source by connecting to the data source in SQLTERM (see CONNECT and DBALIAS) and issuing a TYPES command. For more information, see "sqlterm" (p. 439).

**Note:** If you are using DB2® version 8.2 client, to use the GRAPHIC, VARGRAPHIC, and DBGRAPHIC data types you must add the following line to the [COMMON] section of the db2cli.ini file:

```
Graphic=1
```

**Example**

UDA Driver Information

==================================================================
Driver is DB2
Data Source is SQL
Client Version is 03.01.0000
DBMS Version is 08.02.0002
Maximum Schema length is 30
Maximum Table length is 128
Maximum Column length is 30
Default Identifier case 'UPPER CASE', quote character '"'
Transactions supported.
Cursor COMMIT behavior DISRUPTIVE  
DB/2 cli Information  
==================================================================
ODBC version 3.52  
DBMS is 'DB2/NT'  
Driver version 08.01.0000, ODBC version 03.0  
ODBC conformance is 'Level 2'  
NOT NULLs supported, Transactions fully supported.

**DBMS to ODBC type conversion:**

<table>
<thead>
<tr>
<th>DBMS Type (precision, min, max)</th>
<th>(No) Parameters</th>
<th>Nullable?</th>
<th>ODBC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB (2147483647,0,0)</td>
<td>-4</td>
<td>Nullable</td>
<td>BLOB</td>
</tr>
<tr>
<td>VARCHAR () FOR BIT DATA (32672,0,0)</td>
<td>-3</td>
<td>Nullable</td>
<td>VARBINARY</td>
</tr>
<tr>
<td>CHAR () FOR BIT DATA (254,0,0)</td>
<td>-2</td>
<td>Nullable</td>
<td>BINARY</td>
</tr>
<tr>
<td>CLOB (2147483647,0,0)</td>
<td>-1</td>
<td>Nullable</td>
<td>TEXT</td>
</tr>
<tr>
<td>CHAR (254,0,0)</td>
<td>1</td>
<td>Nullable</td>
<td>CHAR</td>
</tr>
<tr>
<td>NUMERIC (31,0,31)</td>
<td>2</td>
<td>Nullable</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>DECIMAL (31,0,31)</td>
<td>3</td>
<td>Nullable</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>INTEGER (10,0,0)</td>
<td>4</td>
<td>Nullable</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SMALLINT (5,0,0)</td>
<td>5</td>
<td>Nullable</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>FLOAT (53,0,0)</td>
<td>6</td>
<td>Nullable</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>REAL (24,0,0)</td>
<td>7</td>
<td>Nullable</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DOUBLE (53,0,0)</td>
<td>8</td>
<td>Nullable</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATE (10,0,0)</td>
<td>91</td>
<td>Nullable</td>
<td>DATE2</td>
</tr>
<tr>
<td>TIME (8,0,0)</td>
<td>92</td>
<td>Nullable</td>
<td>TIME2</td>
</tr>
<tr>
<td>TIMESTAMP (26,6,6)</td>
<td>93</td>
<td>Nullable</td>
<td>TIMESTAMP2</td>
</tr>
<tr>
<td>VARCHAR (32672,0,0)</td>
<td>12</td>
<td>Nullable</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARCHARGRAPHIC (16336,0,0)</td>
<td>-96</td>
<td>Nullable</td>
<td>NVARCHAR</td>
</tr>
<tr>
<td>GRAPHIC (127,0,0)</td>
<td>-95</td>
<td>Nullable</td>
<td>NCHAR</td>
</tr>
<tr>
<td>DBCLOB (1073741823,0,0)</td>
<td>-97</td>
<td>Nullable</td>
<td>TEXT</td>
</tr>
<tr>
<td>DATALINK (254,0,0)</td>
<td>-400</td>
<td>Nullable</td>
<td>(none)</td>
</tr>
<tr>
<td>BIGINT (20,0,0)</td>
<td>-5</td>
<td>Nullable</td>
<td>QUAD</td>
</tr>
</tbody>
</table>

**ODBC to DBMS type conversion:**

<table>
<thead>
<tr>
<th>ODBC Type</th>
<th>DBMS Type</th>
<th>Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>NCHAR</td>
<td>GRAPHIC</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>VARCHARGRAPHIC</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SMALLGRAPHIC</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC</td>
<td>ODBC Match</td>
</tr>
<tr>
<td>FLOAT</td>
<td>REAL</td>
<td>ODBC Match</td>
</tr>
</tbody>
</table>
### Appendix B: Database Drivers

| DOUBLE          | DOUBLE         | ODBC Match |
| QUAD            | BIGINT         | ODBC Match |
| DATE            | DATE           | Best Match |
| TIME            | TIME           | Best Match |
| TIMESTAMP       | TIMESTAMP      | Best Match |
| INTERVAL        | VARCHAR        | Best Match |
| DATE2           | DATE           | ODBC Match |
| TIME2           | TIME           | ODBC Match |
| TIMESTAMP2      | TIMESTAMP      | ODBC Match |
| TIMETZ          | VARCHAR        | Best Match |
| TIMESTAMPTZ     | VARCHAR        | Best Match |
| INTERVAL2       | VARCHAR        | Best Match |
| INTERVALYM      | VARCHAR        | Best Match |
| BINARY          | CHAR () FOR BIT DATA | ODBC Match |
| VARBINARY       | VARCHAR () FOR BIT DATA | ODBC Match |
| TEXT            | CLOB           | ODBC Match |
| BLOB            | BLOB           | ODBC Match |

### Date Format Conversions

DB2® date formats are identical to IBM® Cognos® Data Manager date formats so no conversion is required.

### DTS Package Driver

You can process data in DTS packages before connecting to it in IBM® Cognos® Data Manager to reuse it. This means that you do not have to stage the data in Data Manager.

You can connect to data stored in DTS packages, but you cannot deliver data to them as this is a read-only driver.

You can use the following SQL statements:

```sql
SELECT *
SELECT * FROM ALL
SELECT * FROM <packagename>
SELECT * FROM <stepname>
```

**Note:** When using DTS packages in Data Manager, in the Microsoft® DTS Workflow Properties window, select DSO Rowset.

### Essbase DBMS Driver

Essbase drivers do not require data type conversion or date format conversion.
## INFORMIX (via Connect) DBMS Driver

### Data Type Conversion

The following table shows the INFORMIX® data types that IBM® Cognos® Data Manager supports.

<table>
<thead>
<tr>
<th>Data Manager</th>
<th>INFORMIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>BOOLEAN if prec=1, otherwise CHAR</td>
</tr>
<tr>
<td>BLOB</td>
<td>BYTE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>CHAR (p)</td>
<td>CHAR (p&gt;255)</td>
</tr>
<tr>
<td></td>
<td>VARCHAR (p &lt;= 255)</td>
</tr>
<tr>
<td>CLOB</td>
<td>TEXT</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME YEAR TO SECOND</td>
</tr>
<tr>
<td></td>
<td>DATETIME YEAR TO DAY</td>
</tr>
<tr>
<td>DATE WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>INTEGER (p)</td>
<td>NUMERIC(10) (p&lt;=2)</td>
</tr>
<tr>
<td></td>
<td>NUMERIC(19) (p&gt;2)</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td>NUMBER</td>
<td>NUMERIC(32)</td>
</tr>
<tr>
<td>TIME</td>
<td>DATETIME HOUR TO SECOND</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
</tbody>
</table>

### Date Format Conversion

Specify the date format within the build or environment variable 'DS_DATEFMT_INFORMIX'.
ODBC DBMS Driver

IBM® Cognos® Data Manager supports ODBC data sources for Microsoft® Windows® and UNIX® operating systems.

ODBC allows applications to interact with many DBMSs, using a single common interface, using SQL. DBMS-specific drivers provide an interface between client applications and the ODBC core. You can purchase these drivers from a number of vendors. In addition, individual software manufacturers may provide ODBC drivers for their products.

An ODBC driver must conform to the ODBC 2.5 specification or higher.

SQL Syntax

Each ODBC driver should provide at least ANSI SQL compatibility. In many cases, an ODBC driver allows the native SQL dialect of the associated DBMS. Because IBM® Cognos® Data Manager passes the SQL statements that you provide unchanged to the ODBC driver, you can use extensions. As with every other driver, Cognos® SQL can also be used.

Network Data Sources

Some ODBC drivers require the existence of other software to establish their connections to the data. Some drivers have server and client portions that provide an alternative to the DBMS-specific network software.

Connecting to ODBC Data Sources

IBM® Cognos® Data Manager connects to ODBC data sources by reference to the appropriate ODBC data source name (DSN). How you configure a particular DSN depends upon the operating system and upon the particular driver that you use to perform the connection. Because there are many differences between the operating system and ODBC driver combinations, we do not attempt to describe the DSN configuration process. For this information, see the documentation that accompanies individual ODBC drivers and the ODBC core software.

Data Type Conversion

For data manipulation, ODBC reconciles disparate data types before the information reaches IBM® Cognos® Data Manager. Data Manager does not normally need to perform data type conversion on an ODBC data source. However, Data Manager must work with the native data types of the DBMS when it performs some data definition operations (for example, table creation). In such cases, Data Manager selects compatible native data types onto which to map its internal data types. Sometimes the DBMS has more than one candidate for a single Data Manager internal data type. For example, a DBMS may support NUMBER, MONEY, and DECIMAL as floating-point data types and each is a suitable candidate onto which Data Manager can map its FLOAT data type. It is recommended that you do not use Data Manager to perform data definition operations where such ambiguity exists.
**Date Format Conversion**

ODBC date formats are identical to IBM® Cognos® Data Manager date formats, so date format conversion is not required.

**Oracle (via SQL*Net) DBMS Driver**

**Data Type Conversion**

The following table shows the Oracle data types that IBM® Cognos® Data Manager supports.

<table>
<thead>
<tr>
<th>Data Manager</th>
<th>ORACLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>RAW</td>
</tr>
<tr>
<td>BLOB</td>
<td>BLOB</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td></td>
<td>NCHAR (Unicode)</td>
</tr>
<tr>
<td></td>
<td>VARCHAR2</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR2 (Unicode)</td>
</tr>
<tr>
<td>CLOB</td>
<td>CLOB</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP (date and time)</td>
</tr>
<tr>
<td></td>
<td>DATE (date only)</td>
</tr>
<tr>
<td>DATE WITH TIME ZONE</td>
<td>TIMESTAMP WITH TIMEZONE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>INTEGER (p)</td>
<td>NUMBER(5) (p&lt;=2)</td>
</tr>
<tr>
<td></td>
<td>NUMBER(10) (2&lt;p&lt;=4)</td>
</tr>
<tr>
<td></td>
<td>NUMBER(15) (p&gt;4)</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td>NUMBER</td>
<td>NUMBER(38)</td>
</tr>
<tr>
<td>TIME</td>
<td>VARCHAR(12)</td>
</tr>
</tbody>
</table>
If Data Manager encounters other Oracle data types, it returns null.

**Date Format Conversion**

In previous versions of IBM® Cognos® Data Manager, by default, the returned value was set to YYYY-MM-DD HH24:MI:SS.

This default has been removed. If you want to set the returned value to YYYY-MM-DD HH24:MI:SS, you must now enable it using the variable named DM_NLS_DATE_FORMAT_COMPATIBILITY.

**Published FM Package Driver**

The IBM® Cognos® Data Manager driver for published IBM® Cognos® Framework Manager packages differs from other database drivers because it is a model that exists over many other data sources. Data Manager does not perform any data type and date conversion operations; these are performed in the source model.

The driver provides a method of accessing any data source that has been modeled and published by Framework Manager. When the IBM® Cognos® Business Intelligence environment has been installed and configured, published packages become available for consumption as Data Manager sources. This allows you to extend the reach of Data Manager to new data sources, such as OLAP sources and additional ODBC coverage.

You can interact with, and select, query items from the data source using a visual query-building environment within Data Manager Designer. This creates an XML-based query specification document that describes the data to be returned. Normally, Data Manager does this automatically, but you can choose to supply the query specification externally for Data Manager to use directly as a data source. This is an advanced capability for query users who are familiar with the IBM® Cognos® Software Development capability.

**Note:** Although it is possible to read any supported relational data source using published packages, such as Oracle sources, unless the sources have been modeled and published for other reasons, it is far more practical to use the Data Manager connection types directly. Data Manager provides a more direct access method for data sources. The driver for published packages is intended for situations where modeled data from relational or OLAP sources is required (such as dimensional data structures) or, for access to web-based data sources (such as web services data).

**Microsoft SQL Server (via OLE-DB) DBMS Drivers**

**Data Type Conversions**

The following table shows the Microsoft® SQL Server 2000 and above data types that IBM® Cognos® Data Manager supports.
<table>
<thead>
<tr>
<th>Data Manager</th>
<th>MSSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>BINARY VARBINARY</td>
</tr>
<tr>
<td></td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>UNIQUEIDENTIFIER</td>
</tr>
<tr>
<td>BLOB</td>
<td>IMAGE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td></td>
<td>NCHAR (Unicode)</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR (Unicode)</td>
</tr>
<tr>
<td></td>
<td>SQL_VARIANT</td>
</tr>
<tr>
<td>CLOB</td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>NTEXT</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
</tr>
<tr>
<td></td>
<td>SMALLDATETIME</td>
</tr>
<tr>
<td>DATE WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td>INTEGER (p)</td>
<td>TINYINT (p=1)</td>
</tr>
<tr>
<td></td>
<td>SMALLINT (p=2)</td>
</tr>
<tr>
<td></td>
<td>INTEGER (2&lt;p&lt;=4)</td>
</tr>
<tr>
<td></td>
<td>BIGINT (4&lt;p&lt;=8)</td>
</tr>
<tr>
<td></td>
<td>BIT</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>VARCHAR(30)</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>VARCHAR(75)</td>
</tr>
</tbody>
</table>
### Date Format Conversions

Specify the date format within the build or environment variable ‘DS_DATEFMT_MSSQL’.

For more information, see "DBMS-related Variables" (p. 335).

### SAP R/3 DBMS Driver

#### Data Type Conversion

The following table shows the SAP/ABAP data types that IBM® Cognos® Data Manager supports.

<table>
<thead>
<tr>
<th>Data Manager</th>
<th>SAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>ACCP, CHAR, CLNT, CUKY, LANG, LCHR, LRAW, NUMC, PREC, RAW, UNIT</td>
</tr>
<tr>
<td>NUMBER</td>
<td>CURR, DEC, QUAN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Manager</th>
<th>MSSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>NUMERIC(38)</td>
</tr>
<tr>
<td></td>
<td>DECIMAL</td>
</tr>
<tr>
<td></td>
<td>MONEY</td>
</tr>
<tr>
<td></td>
<td>SMALLMONEY</td>
</tr>
<tr>
<td>TIME</td>
<td>VARCHAR(12)</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
</tbody>
</table>
No date format conversion is required.

**SQLTXT DBMS Driver**

The SQLTXT DBMS driver lets you access data in delimited text format using SQL.

For the SQLTXT format specification, see "IBM Cognos Data Manager SQLTXT Designer" (p. 385).

**Data Type Conversion**

SQLTXT has three data types that correspond to the IBM® Cognos® Data Manager FLOAT data type. These are FLOAT, PACKED, and ZONED.

**FLOAT**

The FLOAT data type is valid in both delimited and fixed-width text data files. It is a text representation of a floating-point number, space-padded in fixed-length data, encoded as ASCII or EBCDIC to comply with the character set of the text data file.

**PACKED**

The PACKED data type is valid only in fixed-width text data files. It is a packed decimal type. This format stores each decimal digit (in hexadecimal) in one nibble (half a byte), with a further nibble storing the sign (plus or minus). This data type requires half the storage of the FLOAT data type. The low-order nibble stores the sign of the number. This format does not indicate the position of the decimal point. SQLTXT stores scale information in the SQLTXT catalog.

Because the least significant byte stores one decimal digit and the sign of the number, this format can only represent an odd number of decimal digits. If the number to be encoded has an even number of digits, then you must add a leading zero.

The following table gives the permitted values of the sign nibble, together with the meaning of each value.

<table>
<thead>
<tr>
<th>Value (Hex)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0A</td>
<td>Positive</td>
</tr>
<tr>
<td>Value (Hex)</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>0x0B</td>
<td>Negative</td>
</tr>
<tr>
<td>0x0C</td>
<td>Positive (preferred)</td>
</tr>
<tr>
<td>0x0D</td>
<td>Negative (preferred)</td>
</tr>
<tr>
<td>0x0E</td>
<td>Positive</td>
</tr>
<tr>
<td>0x0F</td>
<td>Positive</td>
</tr>
</tbody>
</table>

### Example

The following table shows the PACKED encoding of some example decimal numbers.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>PACKED (Hex)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,234,567.89</td>
<td>12 34 56 78 9C</td>
<td>Positive</td>
</tr>
<tr>
<td>123,456.789</td>
<td>12 34 56 78 9C</td>
<td>Positive</td>
</tr>
<tr>
<td>1,234,567.89</td>
<td>12 34 56 78 9F</td>
<td>Positive</td>
</tr>
<tr>
<td>123,456.789</td>
<td>12 34 56 78 9F</td>
<td>Positive</td>
</tr>
<tr>
<td>-1,234,567.89</td>
<td>12 34 56 78 9D</td>
<td>Negative</td>
</tr>
<tr>
<td>-123,456.789</td>
<td>12 34 56 78 9D</td>
<td>Negative</td>
</tr>
</tbody>
</table>

Each nibble, except the least significant, encodes a decimal digit as a hexadecimal number. The least significant nibble of each number indicates its sign. The examples use the preferred 0x0C for positive numbers, and 0x0D for negative numbers. This method of encoding does not indicate the position of the decimal point. Therefore, the packed representations of 1,234,567.89 and 123,456.789 are identical. Similarly, the representations of -1,234,567.89 and -123,456.789 are identical.

### ZONED

The ZONED data type is valid only in fixed-width text data files. It uses one byte for each digit. Except for the least significant byte, the most significant nibble of each byte contains the zone code (0xF0). For the least significant byte, this nibble stores the sign. The least significant nibble of every byte contains the decimal digit, represented as a hexadecimal number. This format does not indicate the position of the decimal point. SQLTXT stores scale information in the SQLTXT catalog. This method encodes decimal digits as their EBCDIC character equivalents. Except for the least significant digit, you can view the decimal numbers in a standard EBCDIC text editor.

The following table gives the permitted sign values and their meanings.
The following table shows the ZONED encoding of some example decimal numbers.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>ZONED (Hex)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,234,567.89</td>
<td>F1 F2 F3 F4 F5 F6 F7 F8 C9</td>
<td>Positive</td>
</tr>
<tr>
<td>123,456.789</td>
<td>F1 F2 F3 F4 F5 F6 F7 F8 C9</td>
<td>Positive</td>
</tr>
<tr>
<td>-1,234,567.89</td>
<td>F1 F2 F3 F4 F5 F6 F7 F8 D9</td>
<td>Negative</td>
</tr>
<tr>
<td>-123,456.789</td>
<td>F1 F2 F3 F4 F5 F6 F7 F8 D9</td>
<td>Negative</td>
</tr>
</tbody>
</table>

The low-order nibble of each byte encodes a single digit as a hexadecimal number. For all but the least significant byte, the high-order nibble contains the zone code (0xF0). The high-order nibble of the least significant byte indicates the sign of the value. For the first two numbers, this is 0xC0, indicating positive numbers; for the last two, this is 0xD0, indicating negative numbers. Because this method of encoding does not indicate the position of the decimal point, the zoned representations of the two positive numbers are identical. Similarly, the zoned representations of the two negative numbers are identical.

**BOOLEAN**

SQLTXT also has a BOOLEAN data type. This encodes the Boolean values TRUE and FALSE, as string representations of numbers with zero values representing FALSE, and non-zero values representing TRUE.

**Other Data Types**

For other data types, SQLTXT uses the same data types as IBM® Cognos® Data Manager, so no conversion is required.
## Date Format Conversion

Internally, SQLTXT handles dates in IBM® Cognos® Data Manager native format, so no conversion is required.

## Sybase (via CT-Library) DBMS Driver

### Data Type Conversion

The following table shows the Sybase data types that IBM® Cognos® Data Manager supports.

<table>
<thead>
<tr>
<th>Data Manager</th>
<th>SYBASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td></td>
<td>VARBINARY</td>
</tr>
<tr>
<td>BLOB</td>
<td>IMAGE</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td></td>
<td>NCHAR (Unicode)</td>
</tr>
<tr>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>NVARCHAR (Unicode)</td>
</tr>
<tr>
<td>CLOB</td>
<td>TEXT</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
</tr>
<tr>
<td>DATE WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>INTEGER (p)</td>
<td>TINYINT (p=1)</td>
</tr>
<tr>
<td></td>
<td>SMALLINT (p=2)</td>
</tr>
<tr>
<td></td>
<td>INTEGER (2&lt;p&lt;=4)</td>
</tr>
<tr>
<td></td>
<td>BIGINT (4&lt;p&lt;=8)</td>
</tr>
<tr>
<td>INTERVAL DAY TO SECOND</td>
<td>VARCHAR(30)</td>
</tr>
<tr>
<td>INTERVAL YEAR TO MONTH</td>
<td>VARCHAR(75)</td>
</tr>
<tr>
<td>NUMBER</td>
<td>NUMERIC(38)</td>
</tr>
<tr>
<td>Data Manager</td>
<td>SYBASE</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>TIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>TIME WITH TIME ZONE</td>
<td>VARCHAR(75)</td>
</tr>
</tbody>
</table>

If Data Manager encounters other Sybase data types, it produces a run-time error.

**Date Format Conversion**

You must specify the date format within the build or environment variable 'DS_DATEFMT_SYBASE'.

For more information, see "DBMS-related Variables" (p. 335).

**TM1 DBMS Driver**

IBM® Cognos® TM1® drivers do not require data type conversion or date format conversion.
Appendix C: Audit Tables

The audit tables are used to provide summary information about the audit history, the audit trail, the audit messages, and the JobStream details.

There are the following audit tables:

- component run (dsb_component_run)
- context run (dsb_run_context)
- delivery history (dsb_delivery_hist)
- audit trail (dsb_audit_trail)
- audit message (dsb_audit_msg)
- audit message line (dsb_audit_msg_line)
- job node run (dsb_jobnode_run)

You can view the audit tables by creating a connection to the catalog and opening the connection in SQLTerm.

Component Run Table (dsb_component_run)

The component run table contains two records for each execution of a build or JobStream. IBM® Cognos® Data Manager creates a unique run_id for each type of component, and a unique component_uuid for each component executed.

At the start of the execution process, a record indicating the start time and the execution mode is created, and the status column is set to R (running).

On completion, the record is updated to add the end time and the number of rows rejected, and the completion status is set to S if execution was successful, or F if it failed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_id</td>
<td>A number that is unique for each execution process. This is a surrogate key for the component_uuid and run_id natural keys.</td>
</tr>
<tr>
<td>component_uuid</td>
<td>A number that is universally unique for each component executed.</td>
</tr>
<tr>
<td>run_id</td>
<td>A number that is unique for each type of component in the build or JobStream.</td>
</tr>
<tr>
<td>start_timestamp</td>
<td>The start time of the execution process.</td>
</tr>
</tbody>
</table>
# Column | Description
---|---
end_timestamp | The end time of the execution process.
status | The status of the execution process:
  - R indicates the process is running
  - S indicates the process reached a successful completion
  - F indicates the process failed due to errors
  - K indicates the process was successfully stopped manually
  - D indicates the process failed
process_info | A unique number to identify the process that ran the component.
component_type | The component type:
  - B indicates a fact build
  - S indicates a dimension build
  - J indicates a JobStream
  - X indicates that this is an additional record
component_name | The name of the component, or for type X components, the name of the computer on which the component was executed.

## Run Context Table (dsb_run_context)

The run context table contains the context details of a build or JobStream execution.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
</table>
audit_id  | A number that is unique for each component executed.                        |
run_mode  | The execution mode and the delivery types:
  - N indicates normal execution mode
  - C indicates check only execution mode
  - O indicates object creation execution mode
  - F indicates a fact delivery
  - D indicates a dimension delivery
A Boolean value that indicates whether the execution process occurred from within a package:
- Y indicates the execution process occurred from within a package
- N indicates the execution process did not occur from within a package

The name of the computer on which the execution process occurred.

### Delivery History Table (dsb_delivery_hist)

The delivery history table records the number of rows inserted or updated for each table targeted by the execution process. For each table, the name of the database, the table name, whether there were any existing rows in the table, the number of rows that were inserted, and the number of rows that were updated are also recorded.

**Note:** In the case of text file delivery, the file name is recorded in the table_name column and the dbalias column is left blank.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_id</td>
<td>A number that is unique for each component executed.</td>
</tr>
<tr>
<td>dbalias</td>
<td>The name of the database to which the data was delivered.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table into which the data was delivered.</td>
</tr>
<tr>
<td>existing_data</td>
<td>A Boolean value that indicates whether there was existing data in the table:</td>
</tr>
<tr>
<td></td>
<td>- Y indicates there was existing data</td>
</tr>
<tr>
<td></td>
<td>- N indicates there was no existing data</td>
</tr>
<tr>
<td>rows_inserted</td>
<td>The number of rows that were successfully inserted.</td>
</tr>
<tr>
<td>rows_updated</td>
<td>The number of rows that were successfully updated.</td>
</tr>
</tbody>
</table>

### Audit Trail Table (dsb_audit_trail)

The audit trail table records all events from the fact build, dimension build, or JobStream execution process. Events are classified by group and item with a message for each. Some of the messages are text, and others are numbers that you can use for analysis.

You must specify in the relevant Properties window the events to record.
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_id</td>
<td>A number that is unique for each component executed.</td>
</tr>
<tr>
<td>audit_timestamp</td>
<td>The start and end time that each row of data was written.</td>
</tr>
<tr>
<td>audit_group</td>
<td>The audit group to which information is written.</td>
</tr>
<tr>
<td>audit_item</td>
<td>The audit item of the group to which information is written.</td>
</tr>
<tr>
<td>audit_msg</td>
<td>The message from the audit item.</td>
</tr>
</tbody>
</table>

### Audit Message Table (dsb_audit_msg)

The audit message table identifies a message that has been recorded when a fact build, dimension build, or JobStream is executed.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_id</td>
<td>A number that is unique for each component executed.</td>
</tr>
<tr>
<td>message_no</td>
<td>A unique number that identifies each message that is recorded.</td>
</tr>
<tr>
<td>message_code</td>
<td>The code for the message, as shown in the log file.</td>
</tr>
<tr>
<td>severity</td>
<td>The severity of the message:</td>
</tr>
<tr>
<td></td>
<td>• F indicates a fatal message</td>
</tr>
<tr>
<td></td>
<td>• E indicates an error</td>
</tr>
<tr>
<td></td>
<td>• W indicates a warning</td>
</tr>
</tbody>
</table>

### Audit Message Line Table (dsb_audit_msg_line)

The audit message line table records, for each message, the actual lines of message text.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>audit_id</td>
<td>A number that is unique for each component executed.</td>
</tr>
<tr>
<td>message_no</td>
<td>A unique number that identifies each message that is recorded.</td>
</tr>
<tr>
<td>line_no</td>
<td>A unique number that identifies the line number of each message.</td>
</tr>
<tr>
<td>message_text</td>
<td>The message text.</td>
</tr>
</tbody>
</table>
Job Node Run Table (dsb_jobnode_run)

This table stores a record for each node in a JobStream. If the node is a fact build node, dimension build node, or JobStream node, it is allocated an audit_id and run_id. These IDs are recorded in the `ds_component_run` table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>job_audit_id</code></td>
<td>A number that is unique for the JobStream currently being executed.</td>
</tr>
<tr>
<td><code>node_id</code></td>
<td>The number of the JobStream node.</td>
</tr>
<tr>
<td><code>node_type</code></td>
<td>The node type:</td>
</tr>
<tr>
<td></td>
<td>● B indicates a fact build node</td>
</tr>
<tr>
<td></td>
<td>● D indicates a dimension build node</td>
</tr>
<tr>
<td></td>
<td>● C indicates a condition node</td>
</tr>
<tr>
<td></td>
<td>● S indicates an SQL node</td>
</tr>
<tr>
<td></td>
<td>● P indicates a procedure node</td>
</tr>
<tr>
<td></td>
<td>● E indicates an email node</td>
</tr>
<tr>
<td></td>
<td>● A indicates an alert node</td>
</tr>
<tr>
<td></td>
<td>● J indicates a JobStream node</td>
</tr>
<tr>
<td><code>node_caption</code></td>
<td>The name of the JobStream node.</td>
</tr>
<tr>
<td><code>start_timestamp</code></td>
<td>The start time of the execution process.</td>
</tr>
<tr>
<td><code>end_timestamp</code></td>
<td>The end time of the execution process.</td>
</tr>
<tr>
<td><code>status</code></td>
<td>The status of the process:</td>
</tr>
<tr>
<td></td>
<td>● R indicates the process is running</td>
</tr>
<tr>
<td></td>
<td>● S indicates the process reached a successful completion</td>
</tr>
<tr>
<td></td>
<td>● F indicates the process failed due to errors</td>
</tr>
<tr>
<td></td>
<td>● K indicates the process was successfully stopped manually</td>
</tr>
<tr>
<td></td>
<td>● D indicates the process failed</td>
</tr>
<tr>
<td><code>masterjob_audit_id</code></td>
<td>A number that is unique for the master JobStream execution.</td>
</tr>
<tr>
<td><code>comp_audit_id</code></td>
<td>For fact build, dimension build, and JobStream node types, a number that identifies the component execution.</td>
</tr>
</tbody>
</table>

User Guide 535
Appendix D: Date, Time, and Interval Formats

This section describes the date, time, and interval formats used by IBM® Cognos® Data Manager.

Date Formats

IBM® Cognos® Data Manager uses syyyy-mm-dd [hh:mi:ss[.fffffffff]] as the default date format. English is the default language used to specify months and days of the week. However, you can customize Data Manager to use an alternative language for these names.

If you want to use French or German date formats, Data Manager includes both month and day files translated into French and German. For more information, see the IBM® Cognos® Data Manager Installation and Configuration Guide.

Date formats can contain any of these symbols, interspersed with format text.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
</table>
| - / . ; : "text" | Punctuation and text that is reproduced in the result  
  Note: You must enclose literal text within double quotation marks. |
<p>| am/pm | am/pm indicator |
| d | Day of the week (1 to 7) |
| day | Name of the day padded with blanks to a length of nine characters |
| dd | Day of the month (1 to 31) |
| ddd | Day of the year (1 to 366) |
| dy | Abbreviated name of the day of the week |
| fffffff | Fractions of a second (maximum of 9) |
| hh | Hour of the day using the 24 hour clock |
| hh12 | Hour of the day using the 12 hour clock |
| hh24 | Hour of the day using the 24 hour clock |
| iw | Week of the year (1 to 52 or 1 to 53) based on the ISO standard |
| iyy | Last three digits of the ISO year |</p>
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iy</td>
<td>Last two digits of the ISO year</td>
</tr>
<tr>
<td>i</td>
<td>Last digit of the ISO year</td>
</tr>
<tr>
<td>iyyyy</td>
<td>Four digit year based on the ISO year</td>
</tr>
<tr>
<td>j</td>
<td>Julian day. That is, the number of days since January 1st, 4712 BC. This number must be an integer</td>
</tr>
<tr>
<td>m</td>
<td>Month number (1 to 12). Months 1 to 9 are not zero padded</td>
</tr>
<tr>
<td>mm</td>
<td>Month number (01 to 12). Months 1 to 9 are zero padded</td>
</tr>
<tr>
<td>mi</td>
<td>Minute (00 to 59)</td>
</tr>
<tr>
<td>mmm (or mon)</td>
<td>Lower case, three letter abbreviation of the month name (jan to dec)</td>
</tr>
<tr>
<td>Mmm (or Mon)</td>
<td>Proper case, three letter abbreviation of the month name (Jan to Dec)</td>
</tr>
<tr>
<td>MMM (or MON)</td>
<td>Upper case, three letter abbreviation of the month name (JAN to DEC)</td>
</tr>
<tr>
<td>month</td>
<td>Lower case month name (january to december). Where necessary months are zero padded</td>
</tr>
<tr>
<td>Month</td>
<td>Proper case month name (January to December). Where necessary months are zero padded</td>
</tr>
<tr>
<td>MONTH</td>
<td>Upper case month name (JANUARY to DECEMBER). Where necessary months are zero padded</td>
</tr>
<tr>
<td>Q</td>
<td>Quarter of year (1 to 4) where quarter 1 is January to March inclusive</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds (00 to 59)</td>
</tr>
<tr>
<td>sssss</td>
<td>Seconds since midnight (0 to 86399)</td>
</tr>
<tr>
<td>stzh</td>
<td>Signed number of hour for the time zone difference (-12 to 13)</td>
</tr>
<tr>
<td>tzm</td>
<td>Number of minutes for the time zone difference (00 to 59)</td>
</tr>
<tr>
<td>w</td>
<td>Week of the month (1 to 5) where week 1 starts on the first day of the month and ends on the seventh day of the month</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>ww</td>
<td>Week of the year (1 to 53) where week 1 starts on the first day of the year and continues to the seventh day of the year</td>
</tr>
<tr>
<td>yyyy</td>
<td>Four digit year</td>
</tr>
<tr>
<td>syyyy</td>
<td>Four digit year. Dates BC are prefixed by -</td>
</tr>
<tr>
<td>yyy</td>
<td>Last three digits of the year</td>
</tr>
<tr>
<td>yy</td>
<td>Last two digits of the year</td>
</tr>
<tr>
<td>y</td>
<td>Last digit of the year</td>
</tr>
</tbody>
</table>

**Examples**

The following examples refer to a date and time of June 22 2006 17:35:29.123.

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd/mm/yy hh:mi:ss</td>
<td>22/06/06 17:35:29</td>
</tr>
<tr>
<td>hh:mi:ss.ffff</td>
<td>17:35:29.1230</td>
</tr>
<tr>
<td>dd-Mmm-yyyy hh12:mi pm</td>
<td>22-Jun-2006 05:35 pm</td>
</tr>
</tbody>
</table>

The following examples refer to a date and time of June 22 2006 17:35:29.123 -5:00.

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd/mm/yy hh:mi:ss stzh:stzm</td>
<td>22/06/06 17:35:29.123 -5:00</td>
</tr>
<tr>
<td>dd-Mmm-yyyy hh12:mi pm stzh:stzm</td>
<td>22-Jun-2006 05:35 pm -500</td>
</tr>
</tbody>
</table>

**Default Century Behavior**

If IBM® Cognos® Data Manager encounters a year without an explicit century (for example, 56), it assumes that the year is in the 1900s if it is 70 or greater, whereas if it less than 70 it assumes that it is in the 2000s.
Appendix D: Date, Time, and Interval Formats

Examples

<table>
<thead>
<tr>
<th>Year value</th>
<th>Interpreted as</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>2000</td>
</tr>
<tr>
<td>56</td>
<td>2056</td>
</tr>
<tr>
<td>69</td>
<td>2069</td>
</tr>
<tr>
<td>70</td>
<td>1970</td>
</tr>
<tr>
<td>99</td>
<td>1999</td>
</tr>
</tbody>
</table>

Time Formats

IBM® Cognos® Data Manager uses hh:mm:ss[.fffffffff] as the default time format.

Time formats can contain any of these symbols, interspersed with format text.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- / . ; : &quot;text&quot;</td>
<td>Punctuation and text that is reproduced in the result</td>
</tr>
<tr>
<td></td>
<td>Note: You must enclose literal text within double quotation marks.</td>
</tr>
<tr>
<td>am/pm</td>
<td>am/pm indicator</td>
</tr>
<tr>
<td>fffffff</td>
<td>Fractions of a second (maximum of 9)</td>
</tr>
<tr>
<td>hh</td>
<td>Hour of the day using the 24 hour clock</td>
</tr>
<tr>
<td>hh12</td>
<td>Hour of the day using the 12 hour clock</td>
</tr>
<tr>
<td>hh24</td>
<td>Hour of the day using the 24 hour clock</td>
</tr>
<tr>
<td>mi</td>
<td>Minutes (00 to 59)</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds (00 to 59)</td>
</tr>
<tr>
<td>sssss</td>
<td>Seconds since midnight (0 to 86399)</td>
</tr>
<tr>
<td>stzh</td>
<td>Signed number of hour for the time zone difference (-12 to 13)</td>
</tr>
<tr>
<td>tzm</td>
<td>Number of minutes for the time zone difference (00 to 59)</td>
</tr>
</tbody>
</table>
Examples

The following examples refer to a time of 17:35:29.329.

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>hh:mi:ss</td>
<td>17:35:29</td>
</tr>
<tr>
<td>hhmiss.ff</td>
<td>173529.32</td>
</tr>
<tr>
<td>hh12:mi pm</td>
<td>05:35 pm</td>
</tr>
</tbody>
</table>

Interval Formats

IBM® Cognos® Data Manager supports two interval formats, day to second and year to month.

Day to Second Intervals

The default format for day to second intervals is sdddddddddd hh:mi:ss[.fffffffff]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- / . ; : &quot;text&quot;</td>
<td>Punctuation and text that is reproduced in the result</td>
</tr>
<tr>
<td></td>
<td>Note: You must enclose literal text within double quotation marks.</td>
</tr>
<tr>
<td>ffffffffff</td>
<td>Fractions of a second (maximum of 9)</td>
</tr>
<tr>
<td>hh</td>
<td>Hour of the day using the 24 hour clock</td>
</tr>
<tr>
<td>mi</td>
<td>Minutes (00 to 59)</td>
</tr>
<tr>
<td>sdddddddddd</td>
<td>Signed days in interval (maximum of 9)</td>
</tr>
<tr>
<td>ss</td>
<td>Seconds (00 to 59)</td>
</tr>
<tr>
<td>sssss</td>
<td>Seconds since midnight (0 to 86399)</td>
</tr>
</tbody>
</table>

Examples

The following examples refer to a day to second interval of 22 days 17 hours 35 minutes and 19.999 seconds.

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sddd hh:mi:ss.fff</td>
<td>022 17:35:29.999</td>
</tr>
</tbody>
</table>
Appendix D: Date, Time, and Interval Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdd hh:mi:ss.f</td>
<td>22 17:35:29.9</td>
</tr>
<tr>
<td>sdd hhmiss</td>
<td>22 173529</td>
</tr>
<tr>
<td>dd-hhmi</td>
<td>11-1705</td>
</tr>
</tbody>
</table>

**Year to Month Intervals**

The default format for year to month intervals is syyyyyyyyy-mm

Year to month intervals can contain any of these symbols, interspersed with format text.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>- / . ; : &quot;text&quot;</td>
<td>Punctuation and text that is reproduced in the result</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: You must enclose literal text within double quotation marks.</td>
</tr>
<tr>
<td>mm</td>
<td>Months in interval</td>
</tr>
<tr>
<td>syyyyyyy</td>
<td>Signed years in interval (maximum of 9)</td>
</tr>
</tbody>
</table>

**Examples**

The following examples refer to a year to month interval of 100 years 11 months.

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>syyyy-mm</td>
<td>100-11</td>
</tr>
<tr>
<td>syyyy/mm</td>
<td>0100/11</td>
</tr>
</tbody>
</table>
To provide consistency, IBM® Cognos® Data Manager converts the native data types of each DBMS to its own (internal) data types. This allows it to acquire, transform, and deliver data, irrespective of the source and destination DBMS.

The following table contains the Data Manager internal data types.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>An array of values, which may be of mixed type. The index of the first item is 1. The maximum number of values is limited only by available memory.</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary values which exceed the limit of the BINARY data type may be held as a BLOB (Binary Large Object) data type.</td>
</tr>
<tr>
<td></td>
<td>BLOB values which are less than DS_LOB_BUF_SIZE in length (default 8000 bytes) are held in memory and can be regarded as BINARY.</td>
</tr>
<tr>
<td></td>
<td>BLOB values which are larger than DS_LOB_BUF_SIZE are held in temporary files.</td>
</tr>
<tr>
<td>Binary (&lt;n&gt;)</td>
<td>A binary value of fixed or variable length &lt;n&gt;, where 0 &lt; &lt;n&gt; &lt;= 8000.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean data types have a value of TRUE or FALSE. They are held internally as 0 or 1.</td>
</tr>
<tr>
<td>Char(&lt;p&gt;)</td>
<td>A character value of fixed or variable precision &lt;p&gt;, where 0 &lt;= &lt;p&gt; &lt;= 8000.</td>
</tr>
<tr>
<td></td>
<td>Character data can be in any encoding which is supported by Data Manager.</td>
</tr>
<tr>
<td>CLOB</td>
<td>Character values which exceed the limit of the CHAR data type can be held as a CLOB (Character Large Object) data type.</td>
</tr>
<tr>
<td></td>
<td>CLOB values which are less than DS_LOB_BUF_SIZE in length (default 8000 bytes) are held in memory and can be regarded as CHAR. CLOB values which are larger than DS_LOB_BUF_SIZE are held in temporary files.</td>
</tr>
</tbody>
</table>
### Data type

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>It follows from this that CLOB values in Data Manager can be manipulated in the same way as CHAR values providing they are less than DS_LOB_BUF_SIZE in length. CLOB is a string type so wherever a function takes a parameter of type 'string', a CLOB value may be substituted, but with the proviso above. Attempts to manipulate CLOB values which are more than DS_LOB_BUF_SIZE in length will generally result in a NULL value. Exceptions to this are the built in functions Length, which returns the CLOB length regardless, and Checksum, which calculates a CRC based on the actual data or on the temporary file name. The precision of CLOB values is unknown until the value is fetched and this can lead to differences between CLOB and CHAR processing. CLOB columns are assigned the default maximum precision of 4000 on input and derived data items which return CHAR have a default precision of 255. This means that you may have to manually create tables when delivering derivations of CLOB values, as Data Manager cannot calculate the required precision. In SQLTerm, a maximum limit of 4000 characters is imposed for CLOB values. Any values containing more than 4000 characters are truncated.</td>
</tr>
</tbody>
</table>

| Date |
| Can be used to represent either a date only with a default format of yyyy-mm-dd, or a date time with a default format of yyyy-mm-dd hh:mm:ss.fffffffff. For more information on formats, see "Date, Time, and Interval Formats" (p. 537). |

| Float |
| Double precision, floating point numbers from 2.2250738585072014 x e -308 to 1.7976931348623158 x e308 |

| Integer |
| Whole numbers, that is, those without a decimal point, both positive and negative, ranging from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |

| Interval day to second |
| A signed interval value where the interval is expressed as days, hours, minutes, and seconds. Default format sdddddhh:mm:ss.fffffffff. For more information, see "Day to Second Intervals" (p. 541). |
### Data Type Description

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval year to month</td>
<td>A signed interval value where the interval is expressed as years and months. Default format syyyyyyyyy-mm. For more information, see &quot;Year to Month Intervals&quot; (p. 542).</td>
</tr>
<tr>
<td>Number(&lt;p&gt;,&lt;s&gt;)</td>
<td>Precise numbers with a precision &lt;p&gt; of up to 77 significant numbers with a scale &lt;s&gt; factor of up to the number of significant figures. Precision is the total number of digits included in the number. Scale is the number of digits after the decimal point. For information about operations on numbers, see &quot;Rules for Performing Calculations on Numbers&quot; (p. 545).</td>
</tr>
<tr>
<td>Numeric</td>
<td>A combination of the number, integer, and float data types.</td>
</tr>
<tr>
<td>String</td>
<td>A combination of the char and CLOB data types.</td>
</tr>
<tr>
<td>Time</td>
<td>Represents a time value.                                                     Default format hh:mm:ss.fffffffff. For more information, see &quot;Time Formats&quot; (p. 540).</td>
</tr>
</tbody>
</table>

For descriptions of the conversions between the internal data types that Data Manager uses and the DBMS native data types, see "Database Drivers" (p. 515).

### Rules for Performing Calculations on Numbers

A number has a left side (digits prior to the decimal point) and right side (digits that follow the decimal point), where

- the left side (LS) is equal to the precision minus the scale
- the right side (RS) is equal to the scale

For example, 12.349 is a number with the left side equal to 2, the right side equal to 3, and has a precision and scale of [5,3].

When you perform a calculation using two numbers (A and B), the length of the left and right sides of the resultant number (C) depend on

- the lengths of the left side and right side of the original numbers (A and B)
- the operation you are performing

For example C(LS.RS) = A(LS.RS) op B(LS.RS)

Having determined the lengths of the left and right side of a calculated number, the precision and scale can be set.
The rules used to determine the lengths of the left and right side of a calculated number \(C(\text{LS.RS})\) are described in the following tables.

### Rule One

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>No RS for A or B</td>
<td>LS (C) add LS lengths of A and B</td>
</tr>
<tr>
<td></td>
<td>LS length &lt;19 for A and B</td>
<td>Maximum precision of C is 19</td>
</tr>
</tbody>
</table>

### Rule Two

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition, subtraction</td>
<td>LS (C) use maximum LS length (A or B) +1</td>
<td>Maximum precision of C is 19</td>
</tr>
</tbody>
</table>

### Rule Three

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any (except division)</td>
<td>LS length &gt;19 for A or B</td>
<td>LS (C) use maximum LS length (A or B)</td>
</tr>
</tbody>
</table>

### Rule Four

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication</td>
<td>RS length &gt;0 for A or B</td>
<td>RS (C) use RS maximum length (A or B)</td>
</tr>
<tr>
<td></td>
<td>Any LS length for A or B</td>
<td>LS (C) add LS lengths of A and B</td>
</tr>
</tbody>
</table>
### Rule Five

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition, subtraction</td>
<td>LS (C) use maximum LS length (A or B) +1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RS (C) use maximum RS length (A or B)</td>
<td></td>
</tr>
</tbody>
</table>

### Rule Six

<table>
<thead>
<tr>
<th>Operation</th>
<th>RS and LS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Division</td>
<td>LS (C) add LS length (A) to RS length (B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RS (C) set to 16</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Unless stated otherwise, the maximum precision of a calculated Number is 77.

### Example Calculations on Numbers

- This example uses rule 4 to determine the precision and scale of the calculated number X:
  
  \[123.456 \ [6,3] \times 100 \ [3,0] = X \ (Px, Sx)\]

  The LS of X is 3, and the RS is 3. The precision and scale of X is \([6,3]\).

- This example uses rule 6 to determine the precision and scale of the calculated number Y:
  
  \[123.456 \ [6,3] / 100 \ [3,0] = Y \ (Py, Sy)\]

  The LS of Y is 3, and the RS of Y is 16. The precision and scale of Y is therefore \([19,16]\).
Appendix F: Buttons and Icons

This appendix provides a reference to the icons and buttons used in IBM® Cognos® Data Manager:

- toolbar buttons
- fact build visualization icons
- dimension build visualization icons
- reference structure visualization icons
- JobStream visualization icons
- metadata visualization icons
- source code control icons
- audit history icons

Toolbar Buttons

This section describes the following IBM® Cognos® Data Manager toolbars:

- Standard toolbar
- JobStream toolbar
- SQLTerm and data source toolbar
- Reference Explorer toolbar

Standard Toolbar Buttons

The following table shows and briefly describes the buttons on the Standard toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="New Catalog Icon" /></td>
<td>Opens the New Catalog dialog box where you can create a catalog.</td>
</tr>
<tr>
<td><img src="image" alt="Open Catalog Icon" /></td>
<td>Opens the Open Catalog dialog box where you can select the catalog that you want to open.</td>
</tr>
<tr>
<td><img src="image" alt="Save Changes Icon" /></td>
<td>Saves all the changes that you have made to the current catalog.</td>
</tr>
<tr>
<td><img src="image" alt="Properties Icon" /></td>
<td>Opens the Properties window for the selected component.</td>
</tr>
<tr>
<td><img src="image" alt="Execute Build Icon" /></td>
<td>Executes the current build using the default settings.</td>
</tr>
</tbody>
</table>
### Appendix F: Buttons and Icons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="fact_build.png" alt="Fact Build" /></td>
<td>Opens the Fact Build wizard which guides you through the creation of a fact build.</td>
</tr>
<tr>
<td><img src="dimension_build.png" alt="Dimension Build" /></td>
<td>Opens the Dimension Build wizard which guides you through the creation of a dimension build.</td>
</tr>
<tr>
<td><img src="hierarchy.png" alt="Hierarchy" /></td>
<td>Opens the Hierarchy wizard which guides you through the creation of a hierarchy.</td>
</tr>
<tr>
<td><img src="date_hierarchy.png" alt="Date Hierarchy" /></td>
<td>Opens the Date Hierarchy wizard which guides you through the creation of a date hierarchy.</td>
</tr>
<tr>
<td><img src="sqlterm.png" alt="SQLTerm" /></td>
<td>Opens SQLTerm, allowing you to examine data or construct SQL statements.</td>
</tr>
<tr>
<td><img src="sqltxt.png" alt="SQLTXT Designer" /></td>
<td>Opens IBM® Cognos® Data Manager SQLTXT Designer which you can use to configure SQLTXT definition files that are used within IBM® Cognos® Data Manager to access text files as though they are in an SQL database.</td>
</tr>
<tr>
<td><img src="reference_explorer.png" alt="Reference Explorer" /></td>
<td>Opens the reference explorer that you can use to test and browse reference structures.</td>
</tr>
<tr>
<td><img src="visualization.png" alt="Visualization Pane" /></td>
<td>Hides or reveals the Visualization pane.</td>
</tr>
<tr>
<td><img src="object_size.png" alt="Object Size" /></td>
<td>Enables you to choose the size of the objects shown in the Visualization pane.</td>
</tr>
<tr>
<td><img src="original_size.png" alt="Original Size" /></td>
<td>Restores the objects in the Visualization pane to their original size.</td>
</tr>
<tr>
<td><img src="clipboard.png" alt="Clipboard" /></td>
<td>Copies the contents of the Visualization pane to the clipboard, from where you can paste them into another application.</td>
</tr>
<tr>
<td><img src="navigator.png" alt="Navigator" /></td>
<td>Opens the navigator that you can use to navigate through the Tree pane, search for components, and ascertain component dependencies.</td>
</tr>
<tr>
<td><img src="data_manager_help.png" alt="Data Manager Help" /></td>
<td>Opens the Data Manager online help.</td>
</tr>
</tbody>
</table>

### JobStream Toolbar Buttons

The following table shows and briefly describes the buttons on the JobStream toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="select_mode.png" alt="Select Mode" /></td>
<td>Returns to select mode, from link mode.</td>
</tr>
</tbody>
</table>
### Buttons and Description

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Fact Build Node" /></td>
<td>Adds a fact build node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Dimension Build Node" /></td>
<td>Adds a dimension build node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="JobStream Node" /></td>
<td>Adds a JobStream node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Condition Node" /></td>
<td>Adds a condition node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Procedure Node" /></td>
<td>Adds a procedure node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="SQL Node" /></td>
<td>Adds an SQL node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Alert Node" /></td>
<td>Adds an alert node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Email Node" /></td>
<td>Adds an email node to the selected JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Link Mode" /></td>
<td>Changes to link mode so that you can link nodes in the JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Grid Alignment" /></td>
<td>Automatically positions nodes.</td>
</tr>
<tr>
<td><img src="image" alt="Grid Alignment" /></td>
<td>Aligns nodes to a hidden grid.</td>
</tr>
</tbody>
</table>

### SQLTerm and Data Source Toolbar Buttons

The following table shows and briefly describes the buttons on the SQLTerm toolbar and the Data Source Properties toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Execute All" /></td>
<td>Executes the current SQL query and returns all the selected rows in the database.</td>
</tr>
<tr>
<td><img src="image" alt="Execute First" /></td>
<td>Executes the current SQL query and returns the first row in the database.</td>
</tr>
<tr>
<td><img src="image" alt="Execute Statement" /></td>
<td>Executes the current SQL statement and returns the result of the command. Used for the execution of commands that do not return result rows.</td>
</tr>
<tr>
<td><img src="image" alt="Interrupt" /></td>
<td>Interrupts the execution of an SQL query.</td>
</tr>
<tr>
<td><img src="image" alt="Show SQL" /></td>
<td>Shows the SQL query that has been executed.</td>
</tr>
<tr>
<td><img src="image" alt="Show XML" /></td>
<td>Shows the generated XML for the query in the Query Items pane.</td>
</tr>
</tbody>
</table>
### Reference Explorer Toolbar Buttons

The following table shows and briefly describes the buttons on the Reference Explorer toolbar.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Button" /></td>
<td>Shows the elements at each level of the current reference structure.</td>
</tr>
<tr>
<td><img src="image2" alt="Button" /></td>
<td>Shows the reference structure data as a true hierarchical structure.</td>
</tr>
<tr>
<td><img src="image3" alt="Button" /></td>
<td>Opens the Reference Explorer dialog box, allowing you to refresh the current reference structure information or select another reference structure.</td>
</tr>
<tr>
<td><img src="image4" alt="Button" /></td>
<td>Hides or reveals the Attributes pane.</td>
</tr>
<tr>
<td><img src="image5" alt="Button" /></td>
<td>Opens the Find dialog box where you can search for members by name.</td>
</tr>
</tbody>
</table>

### Fact Build Visualization Icons

The following table shows and briefly describes the icons that may be used to show a fact build in the Visualization pane.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>The database where IBM® Cognos® Data Manager acquires the source data, or delivers the transformed data.</td>
</tr>
<tr>
<td><img src="image7" alt="Icon" /></td>
<td>The data source that Data Manager is to use to extract the data from the source database.</td>
</tr>
<tr>
<td><img src="image8" alt="Icon" /></td>
<td>A data source column.</td>
</tr>
<tr>
<td>Icons</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td>A data source derivation.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Image" /></td>
<td>A data source literal value.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
<td>A DataStream.</td>
</tr>
<tr>
<td><img src="image4.png" alt="Image" /></td>
<td>A DataStream to which a filter has been applied.</td>
</tr>
<tr>
<td><img src="image5.png" alt="Image" /></td>
<td>A DataStream item.</td>
</tr>
<tr>
<td><img src="image6.png" alt="Image" /></td>
<td>A private DataStream item.</td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td>A DataStream derivation.</td>
</tr>
<tr>
<td><img src="image8.png" alt="Image" /></td>
<td>A private DataStream derivation.</td>
</tr>
<tr>
<td><img src="image9.png" alt="Image" /></td>
<td>A DataStream pivot value.</td>
</tr>
<tr>
<td><img src="image10.png" alt="Image" /></td>
<td>The transformation model.</td>
</tr>
<tr>
<td><img src="image11.png" alt="Image" /></td>
<td>An attribute element in the transformation model.</td>
</tr>
<tr>
<td><img src="image12.png" alt="Image" /></td>
<td>A derivation element in the transformation model.</td>
</tr>
<tr>
<td><img src="image13.png" alt="Image" /></td>
<td>A dimension element in the transformation model.</td>
</tr>
<tr>
<td><img src="image14.png" alt="Image" /></td>
<td>A derived dimension element in the transformation model.</td>
</tr>
<tr>
<td><img src="image15.png" alt="Image" /></td>
<td>A measure element in the transformation model.</td>
</tr>
<tr>
<td><img src="image16.png" alt="Image" /></td>
<td>Allow records with duplicate keys.</td>
</tr>
<tr>
<td><img src="image17.png" alt="Image" /></td>
<td>Reject records with duplicate keys.</td>
</tr>
<tr>
<td><img src="image18.png" alt="Image" /></td>
<td>Merge records with duplicate keys.</td>
</tr>
<tr>
<td><img src="image19.png" alt="Image" /></td>
<td>Breaking is not performed.</td>
</tr>
<tr>
<td><img src="image20.png" alt="Image" /></td>
<td>Breaking is performed.</td>
</tr>
</tbody>
</table>
### Icons

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="104x676" alt="Image" /></td>
<td>A hierarchy where unmatched members are not included.</td>
</tr>
<tr>
<td><img src="121x639" alt="Image" /></td>
<td>A hierarchy where aggregation is to be performed.</td>
</tr>
<tr>
<td><img src="133x608" alt="Image" /></td>
<td>A hierarchy where unmatched members are included, and no aggregation is to be performed.</td>
</tr>
<tr>
<td><img src="104x565" alt="Image" /></td>
<td>A fact delivery.</td>
</tr>
<tr>
<td><img src="121x529" alt="Image" /></td>
<td>A fact delivery where either a level filter or output filter has been applied.</td>
</tr>
<tr>
<td><img src="119x494" alt="Image" /></td>
<td>A dimension delivery.</td>
</tr>
<tr>
<td><img src="122x452" alt="Image" /></td>
<td>The table to which the fact delivery is to deliver the data.</td>
</tr>
<tr>
<td><img src="125x356" alt="Image" /></td>
<td>A primary partitioned table.</td>
</tr>
<tr>
<td><img src="122x309" alt="Image" /></td>
<td>A secondary partitioned table.</td>
</tr>
</tbody>
</table>

### Dimension Build Visualization Icons

The following table shows and briefly describes the icons that may be used to show a dimension build in the Visualization pane.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="104x206" alt="Image" /></td>
<td>The hierarchy to be delivered.</td>
</tr>
<tr>
<td><img src="122x173" alt="Image" /></td>
<td>The auto-level hierarchy to be delivered.</td>
</tr>
<tr>
<td><img src="121x145" alt="Image" /></td>
<td>A balanced auto-level hierarchy.</td>
</tr>
<tr>
<td><img src="119x111" alt="Image" /></td>
<td>The lookup to be delivered.</td>
</tr>
<tr>
<td><img src="120x77" alt="Image" /></td>
<td>The dimension build.</td>
</tr>
<tr>
<td><img src="127x45" alt="Image" /></td>
<td>The dimension template.</td>
</tr>
</tbody>
</table>
### Metadata Visualization Icons

The following table shows and briefly describes the icons that may be used to show the metadata model of the conformed data mart or warehouse. This model can be output to an XML file so that it can be read by IBM® Cognos® Framework Manager.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>A dimension within the conformed metadata model.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A fact table within the conformed metadata model.</td>
</tr>
</tbody>
</table>

### Reference Structure Visualization Icons

The following table shows and briefly describes the icons that may be used to show a reference structure in the Visualization pane.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Icon]</td>
<td>The database where IBM® Cognos® Data Manager acquires the source data.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>The data source that Data Manager is to use to extract the data from the source database.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A DataStream.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A DataStream to which a filter has been applied.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A table.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>The template that Data Manager is to use to extract the data from the source database.</td>
</tr>
<tr>
<td>![Icon]</td>
<td>A hierarchy.</td>
</tr>
</tbody>
</table>
### JobStream Visualization Icons

The following table shows and briefly describes the icons that may be used to show a JobStream in the Visualization pane.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>The starting point for a JobStream.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>A fact build node.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>A dimension build node.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>An SQL node.</td>
</tr>
</tbody>
</table>
### Source Code Control Icons

The following table shows and briefly describes the icons that may be used when a catalog is under source code control.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Icon 1" /></td>
<td>Component checked into source code control.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Icon 2" /></td>
<td>Component checked out of source code control by current developer.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Icon 3" /></td>
<td>Component checked out of source code control by another developer.</td>
</tr>
</tbody>
</table>

### Audit History Icons

The following table shows and briefly describes the status icons that may be used in the audit history for a build or JobStream execution.

<table>
<thead>
<tr>
<th>Icons</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Icon 1" /></td>
<td>The node listed in the audit history is running.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Icon 2" /></td>
<td>The node listed in the audit history has successfully completed.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Icon 3" /></td>
<td>The node listed in the audit history has failed.</td>
</tr>
<tr>
<td><img src="image4.png" alt="Icon 4" /></td>
<td>The execution process was manually stopped, and the process was successful.</td>
</tr>
<tr>
<td><img src="image5.png" alt="Icon 5" /></td>
<td>The execution process was manually stopped, but the process failed prior to being stopped.</td>
</tr>
</tbody>
</table>
Appendix G: Cross Platform Deployment

You can create builds and JobStreams on one operating system for deployment on another. For example, you can use IBM® Cognos® Data Manager Designer (on a Microsoft® Windows® operating system) to develop builds that you deploy to a UNIX® operating system. Overall, this is a straightforward process. However, some conditions require additional consideration.

Note: You can execute builds and JobStreams on a remote server using IBM® Cognos® Data Manager Network Services. For more information, see "Execute a Build or JobStream on a Remote Computer" (p. 279).

Use IBM® Cognos® Data Manager Designer to Deploy on UNIX®

Very few considerations are necessary before a build that is developed on one platform is deployed on another. This is because little environment-specific information is stored with a build. Where this information does exist however, it should be reviewed. There are two main areas, file names and database connections.

File Names to Use on UNIX

A file name (for example, for a text delivery or a reject file) specified for a Microsoft® Windows® operating system may be of this form

C:\temp\file.txt

This is not acceptable on a UNIX® operating system platform. The simplest way to resolve this is to represent the file path as an IBM® Cognos® Data Manager variable, for example

{$FILE_PATH}file.txt

The variable resolves appropriately on each platform. For information about using variables, see "Variables" (p. 319).

Database Connections

IBM® Cognos® Data Manager uses a system of connection aliases to describe the connections to databases. For example, a connection named Sales may resolve to an Oracle connection of username/password@service1. This system simplifies how connections are specified and reused throughout the design process. Unfortunately, while the connection string username/password@service1 may be the correct definition of the connection from the Microsoft® Windows® operating system computer where Data Manager Designer is running to a UNIX® operating system host computer, it will not necessarily be correct if the same build is executed from a different computer.

For example, if the build were executed from a UNIX® computer where the database resides, the connection may be username/password. If it were executed from a different UNIX® computer, the connection may be username/password@service2.
You must create a database alias definition file that holds the connection details for each environment in which the connection is needed. The database alias definition file replaces the connection definitions defined as part of the Data Manager catalog. When the Data Manager engine is run, it is given the name of the database alias definition file and the environment in which it is running using this syntax:

```
DATABUILD -A<dbalias_file> -e<runenv> <other parameters>
```

### Create a Database Alias Definition File

Database alias definition files are text files that specify to which databases IBM® Cognos® Data Manager can connect. A database alias definition file also specifies the parameters that Data Manager must use when connecting to the data. Each database alias definition file must define all the connections that a build requires.

You can create database alias definition files manually using a standard text editor, by exporting the connection configurations from a catalog, or by combining these two methods.

### Create a Database Alias Definition File Manually

Use a standard text editor to create database alias definition files manually. A database alias definition file consists of one line of text for each alias (or connection definition). The syntax for each line is:

```
<runenv> <name> <driver> <driver_items> [<notes>]
```

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;runenv&gt;</td>
<td>The run environment context for the database alias. By default, this is &lt;DEFAULT&gt;. However, you can create additional run environments for special purposes.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>The alias for the connection.</td>
</tr>
<tr>
<td>&lt;driver&gt;</td>
<td>The name of the database driver to use to connect to the data. For example, ORACLE.</td>
</tr>
<tr>
<td>&lt;driver_items&gt;</td>
<td>The information required by the database driver to establish the connection. For example, user ID and password.</td>
</tr>
<tr>
<td>&lt;notes&gt;</td>
<td>Optional notes to describe the alias. You can only view these notes within the alias definition file.</td>
</tr>
</tbody>
</table>

This is an example of a database alias definition file.

```
//******************
//* DEFAULT ALIASES *
//******************
<DEFAULT> 'Tutorial Catalog' ODBC 'DSN=DS Tutorial Catalog' ''
<DEFAULT> 'Tutorial DataMart' ODBC 'DSN=DS Tutorial Output' ''
<DEFAULT> 'Tutorial Reference' ODBC 'DSN=DS Tutorial Reference' ''
<DEFAULT> 'Tutorial Sales' ODBC 'DSN=DS Tutorial Sales' ''
```
Run Environments

Run environments help determine to which database IBM® Cognos® Data Manager connects when you specify a particular alias. To establish valid run environment contexts, in the alias definition file, precede each alias definition with the name of the context to which that alias relates.

Within the default run environment (<DEFAULT>), you must provide an alias for each database to which Data Manager must connect. In addition, you may provide additional sets of database aliases if each run environment context and alias name combination is unique. If you do not define an alias for a run environment context, then Data Manager uses the alias from <DEFAULT>.

Create a Database Alias Definition File from a Catalog

You can create a database alias definition file using CATEXP to export the connections from a catalog. To do this, in a Microsoft® Windows® operating system command window, use this syntax:

CATEXP <catdb_driver_name> <catdb_driver_items> <export_filename> A [-a]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;catdb_driver_name&gt;</code></td>
<td>Specifies the database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td><code>&lt;catdb_driver_items&gt;</code></td>
<td>Specifies the connection string to pass to the database driver to connect to the catalog.</td>
</tr>
<tr>
<td></td>
<td>For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, you would specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td><code>&lt;export_filename&gt;</code></td>
<td>Specifies the path and name of the file to which you want to export the database aliases.</td>
</tr>
<tr>
<td>A</td>
<td>Specifies that you want to export database aliases.</td>
</tr>
<tr>
<td>-a</td>
<td>Appends the exported component to the named file.</td>
</tr>
</tbody>
</table>

The aliases in the resulting database alias definition file contain the same information as the connections in the catalog from which you exported them. This means that you cannot export the connections from the catalog that you use for Microsoft® Windows® operating system and then use the database alias definition file without modification. Therefore, you can either
Edit the database alias definition file so that the aliases are valid for a UNIX® operating system environment. That is, add new entries to the file, modifying the connection information and the environment.

Create a temporary catalog in which you create the connections for the UNIX® environment. Use CATEXP to export these connections to create the required database alias definition file.

**Password Encryption**

When you create alias definitions manually, you specify database passwords in plain text. Therefore, anyone who can access the database alias definition files can discover the associated passwords. However, when you create a connection in a catalog, IBM® Cognos® Data Manager encrypts the passwords. When you create alias definitions by exporting these connections, Data Manager retains the password encryption.

To use password encryption, you must create the connection within a catalog and then export the connection to a database alias definition file.

*Note:* You can also use CATIMP to import a database alias component that contains an unencrypted password. CATIMP encrypts any unencrypted passwords on import.

**Execute a Fact Build under UNIX**

If IBM® Cognos® Data Manager can access the catalog from a UNIX® operating system, then you can execute the fact build.

**Step**

- Use the DATABUILD command with this syntax

  DATABUILD<catdb_driver><catdb_items><databuild_name><uuid> -A<dbalias_file> -e<runenv>

  When you invoke DATABUILD with this syntax, it executes the named fact build that resides in the named catalog.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;catdb_driver&gt;</td>
<td>The database driver to use to connect to the catalog.</td>
</tr>
<tr>
<td>&lt;catdb_items&gt;</td>
<td>The information required by the database driver to connect to the catalog. For example, to connect as user DBA, with password PASS, to a catalog that resides in an ODBC data source named Sample, specify &quot;DSN=Sample;UID=DBA;PWD=PASS&quot;</td>
</tr>
<tr>
<td>&lt;databuild_name&gt;</td>
<td>Specifies the name of the fact build to execute.</td>
</tr>
<tr>
<td>&lt;uuid&gt;</td>
<td>Specifies the universally unique identifier for the component to execute.</td>
</tr>
</tbody>
</table>
## Example

DATABUILD ODBC "DSN=SalesCat;UID=Sales;PWD=rm123" US_Sales -Aalias.txt -eunix

This example invokes DATABUILD to execute the fact build named US_Sales. To retrieve the fact build definition, DATABUILD connects to the ODBC data source named SalesCat as user Sales, with password rm123. It obtains the connection specifications from the file alias.txt.
Appendix H: Problems When Using IBM Cognos Data Manager

This section provides solutions for problems you may encounter when using IBM® Cognos® Data Manager.

Source Code Control

When using a catalog that is under source code control, if the root of the catalog is checked out, other users are unable to add builds or JobStreams.

To avoid this, individual folders should be used, including one for each developer that is working on the project with new builds and JobStreams. When the builds and JobStreams are complete, move them into a permanent folder.

Writing of Large Objects to DB2 via Bulk Loader

The writing of CLOB and binary data to IBM® DB2® is currently unsupported.

32 bit and 64 bit Database Clients

IBM® Cognos® Data Manager requires 32 bit database clients on UNIX® operating system platforms.

Error Message Codes

The format used for error message codes has changed from DS-XXXXX-TNNN to DM-XXX-NNNNN. As a result, the codes used in previous versions of IBM® Cognos® Data Manager are now invalid. If you use these codes for downstream or conditional processing, you must update them.

Note: Error message codes are stored in the audit logs.

Bind Error When Using SQL Server Bulk Copy via API Delivery Module

When using any load mode in the SQL Server Bulk Copy via API delivery module, when you execute a build, you may encounter the following bind error:

[S1000][Microsoft][ODBC SQL Server Driver] Not enough columns bound

This occurs when a delivery element does not exist for every column in the target table.
To overcome this issue, you can manually create delivery elements for the required columns. Alternatively, you can use a Microsoft® SQL Server (via OLE-DB) connection or change the delivery module to Microsoft® SQL Server BCP Command.

**Netezza ODBC 3.0 Driver Incompatible With IBM Cognos Data Manager**

The Netezza ODBC 3.0 driver is not compatible with IBM® Cognos® Data Manager. You must install the Netezza ODBC 3.5 driver to save catalogs in Data Manager.

**Limited Shared Memory on AIX**

There is a default limit of 11 shared memory segments per 32 bit process on AIX®. This may cause issues with shared processes in a JobStream, and you may encounter error messages like this:

```
DM-IPC-0330 Failed to map/attach shared memory XXX
DM-LIB-0500 OS function 'shmat' produced error 24; The process file table is full
```

To overcome this issue, you must set an AIX® environment variable:

```
EXTSHM=ON
```

**Note:** You cannot set this variable programmatically using putenv().

**Additional Permissions Required for DB2 Connection**

You must set up user access to the SYSIBM schema when using an IBM® DB2® connection in IBM® Cognos® Data Manager.

**TM1 Turbo Integrator delivery via the Data Movement service fails when sharing a local folder**

If you are running IBM® Cognos® Data Manager on your local computer, and you execute a build containing a TM1® Turbo Integrator delivery module via the Data Movement service installed on a server, the build may fail.

This occurs if you specify a local folder in the TM1® Turbo Integrator delivery module properties. The Data Movement service cannot create the required temporary file on your machine because it runs using a local system logon.

To overcome this problem, the IBM® Cognos® Business Intelligence server logon must be changed to an account that can access your local shared folder.

Alternatively, you can use the the direct delivery option in the TM1® Turbo Integrator delivery module properties.
Glossary

**aggregate rule**
A separate aggregate function that can be set up for Data Manager to use for specific dimensions to override the regular aggregate.

**aggregation**
The process of reading data across one level in a hierarchy or auto-level hierarchy and summarizing it.

**ancestor**
A member that exists at a higher level than another member in a hierarchy and is connected by a series of parent-child relationships.

**attribute**
In dimensional models, a property that provides qualitative information about members of a level in a dimension. For example, the Store level within the Retailer dimension might have properties such as address or retail space. In general, dimensional attributes do not have measure values or rollups associated with them, but are used to locate or filter members.

In relational models, a query item that is not a measure or identifier. When a query item is an attribute, it is not intended to be aggregated, or used for grouping or generating prompt pick lists.

In BI modeling, a characteristic of an entity which is descriptive rather than a unique identifier or an aggregative measure.

In TM1, a property that provides qualitative information about dimensions.

**attribute element**
An element in the transformation model that holds non-additive information that is not a dimension or a measure but that may be of interest. For example, descriptive information is typically treated as an attribute.

**audit**
To record information about build or JobStream execution for subsequent analysis.

**auto-level hierarchy**
A hierarchy that does not have a fixed number of levels. An auto-level hierarchy is structured solely in terms of parent-child relationships. For example, employees report to managers, who may report to other managers.

**build**
A specification for the acquisition, transformation, and delivery of data.
**bulk loader**
A program supplied with a relational database management system that allows fast loading of records into a database.

**business dimension**
A category of data, such as products or time periods, that an organization might want to analyze.

**business key**
A level attribute or dimension table column that identifies each member of a hierarchy, level or row of dimension data.

**business name**
A descriptive name for a Data Manager object. Assigning a business name enables analysis software to make reports more comprehensible by using the business name in place of the business key.

**caption**
A description associated with an identifier. The caption is often used in preference to the identifier in reports to make them more comprehensible.

**catalog**
A repository for storing specifications for builds, reference structures, connections, and other Data Manager components.

**child**
In a hierarchy or auto-level hierarchy, a member that has at least one parent.

**column token**
A level attribute that has certain properties, irrespective of the reference structure level. For example, the level attribute that provides the identifier for each row of data is represented by the $ID column token.

**component**
A connection, build, reference dimension, reference structure, template, JobStream, user-defined function, metadata dimension, or metadata collection in a catalog.

**component package**
A Data Manager utility that allows the import and export of components between catalogs.

**conformed dimension**
A dimension, with a single definition, that is reused or shared across multiple data subject areas. The common definitions for common dimensions allow data from different subject areas to be meaningfully compared and used in calculations and drill throughs from one area to another.

In Data Manager,
a dimension with a single definition that can be reused or shared across multiple coordinated data marts.

**connection**
The information required to connect to a database. The actual information required varies according to the database management system (DBMS) and connection method.

**connection alias**
A short name used to identify a database connection.

**coordinated data mart**
A data mart that shares its dimensions with many other data marts.

**credential**
Information acquired during authentication that describes a user, group associations, or other security-related identity attributes, and that is used to perform services such as authorization, auditing, or delegation. For example, a user ID and password are credentials that allow access to network and system resources.

**database alias**
A short name used to identify the parameters required to connect to a specific database.

**database schema**
See schema.

**DataStream**
A Data Manager object that gathers together all the data sources from which a build or reference structure acquires data.

**DataStream item**
A Data Manager object that is required wherever mapping is performed. A DataStream item is used to map a data source column to a transformation model element in a fact build, or a level attribute in a reference structure.

**DBMS driver**
See database management system (DBMS) driver.

**delivery module**
A software module used to deliver data to a variety of target systems, such as relational databases.

**delivery module property**
A property that can be used to configure a delivery module. For example, the relational table delivery module has a property that corresponds to the commit interval.
**derivation**
In a data source or DataStream, a value that Data Manager calculates rather than obtains directly from the source data, using an expression defined by the user.

**derivation element**
An element in the transformation model that Data Manager calculates rather than obtains directly from the source data, using an expression defined by the user.

**derived dimension element**
An element in the transformation model that allows the user to use a calculated expression as the source for a dimension attribute.

**dimension**
In Cognos Planning, a list of related items such as Profit and Loss items, months, products, customers, and cost centers, including calculations. The rows, columns, and pages of a cube are created from dimensions.

In Cognos BI, TM1, and Express, a broad grouping of descriptive data about a major aspect of a business, such as products, dates, or locations. Each dimension includes different levels of members in one or more hierarchies and an optional set of calculated members or special categories.

**dimensional framework**
The collection of reference dimensions in a catalog. Data Manager uses the dimensional framework to provide information when merging data and delivering dimension data to data mart.

**dimension attribute**
See level attribute.

**dimension break**
The process of tracking changes in the dimension sequence to improve engine performance and reduce memory usage.

**dimension build**
A Data Manager build that delivers data to describe a single business dimension, such as Product or Customer.

**dimension delivery**
An object in a fact build that delivers data to describe a single business dimension, such as Product or Customer.

**dimension delivery module**
A delivery module used to deliver data, describing a single dimension, to a target data mart.
**dimension element**
An element in the transformation model that is used to give context to a measure element. For example, a Product_Number dimension element, gives context to a Quantity measure element.

**dynamic domain**
A dimension in which the set of members is constructed from data sources as rows are processed.

**dynamic member**
A member of a reference structure that is defined by reference database tables.

**fact build**
A Data Manager build that delivers fact data. It can also deliver dimension data, allowing the user to construct a private data mart from within a single build.

**fact delivery**
An object in a fact build that delivers fact data.

**fact delivery module**
A delivery module used to deliver fact data to a target data mart.

**fact table**
A database table, commonly located at the center of a star schema surrounded by dimension tables, that contains measures of a business process and foreign keys from the dimension tables.

**filter**
The process of restricting the data delivered by a fact build.

**foster parent**
In a hierarchy or auto-level hierarchy, an artificially introduced member that acts as a parent for members that either have no defined parent or whose defined parent cannot be found at the next highest level.

**hierarchy**
The organization of a set of entities into a tree structure, with each entity (except the root) having one or more parent entities and an arbitrary number of child entities.

In Data Manager, a particular view of a business dimension. A hierarchy contains the definition of related reference data that is organized into a tree structure of members related as parents and children.

**identifier**
The value that is used to represent a value in data. For example, the identifier for a product may be its product code.
input level
In the transformation model, the hierarchy level at which Data Manager accesses source data.

JobStream
A specification of a process that Data Manager can execute; a Jobstream contains a series of steps that can be executed in sequence or in parallel.

JobStream variable
A variable that is declared in the definition of a JobStream. The scope of a JobStream variable is the set of nodes that the JobStream contains. Where a process flow branches in a JobStream, each flow has its own copy of the variable.

key
A column or an ordered collection of columns that is identified in the description of a table, index, or referential constraint. The same column can be part of more than one key.

late arriving fact processing
The process of dealing with transaction records that arrive in the data warehouse out of sequence. These transactions are linked back to their dimension records that were current at the time of the transaction.

level
A set of entities or members that form one section of a hierarchy in a dimension and represent the same type of object. For example, a geographical dimension might contain levels for country, region, and city.

level attribute
A specific piece of information stored about each member of a level. For example the color attribute could be used to store the color of a product. Some attributes can have special significance, for example, when the attribute contains the business key for the member.

level filter
A filter used to restrict the delivery of data to specific dimensions and hierarchy levels.

link
A connection between two nodes in a JobStream.

literal
Static data that can be returned by a DataStream. A literal remains constant for every row that the data source returns.

log file
A text file that records the execution of a fact build, dimension build, or JobStream.
logging
The recording of data about specific events on the system, such as errors.

lookup
A simple, single-level reference structure with no parent-child relationships. Members of a lookup are not arranged into hierarchical levels.

mapping
The process of defining the relationship between Data Manager components to enable the flow of data from a data source to fact build or reference structure.

master table
A table that exists to define the range of an entity in a relational database. For example, a product master table would provide information about all the products of a company.

measure
A performance indicator that is quantifiable and used to determine how well a business is operating. For example, measures can be Revenue, Revenue/Employee, and Profit Margin percent.

measure element
An element in the transformation model that represents a business measure.

member
In Data Manager, a node in a reference structure.

merge
To consolidate source data in the transformation model using a specified merge method. Data can be merged for measure elements or attributes.

merge method
The type of calculation to perform when merging data.

message
Status information that the Data Manager engine returns about a build or JobStream execution.

message frequency
The rate at which Data Manager generates messages as it executes a build.

metadata collection
A metadata component that groups together all the metadata star models that you want to include in a metadata export. A metadata collection can contain an entire data warehouse model, or a subset of a model, such as a single subject area.
**metadata dimension**
A metadata component that contains the description of a conformed dimension to include in a metadata export.

**metadata export**
A utility that allows the user to export descriptions of the target conformed model in a data mart or data warehouse to an XML file. This file can then be used to produce a model of the target data mart or data warehouse in Framework Manager, without the need to recreate the complete model.

**metadata schema**
A set of database tables that holds information about the dimensions of a model. In particular, it holds the members of each dimension, and information about how the members relate to each other.

**metadata star**
A catalog object containing the description of the fact table to include in a metadata export, with reference to the metadata dimensions that have been set up.

**model**
In Data Manager, a system, consisting of fact data and metadata, that represents the aspects of a business.

**multi-developer support**
A concept that allows developers to share information in a catalog. Multi-developer support can be achieved in and across catalogs using source code control, component packages, and backing up and restoring catalogs.

**multidimensional**
Pertaining to any system for which the dimension is the fundamental basis of data organization.

**multidimensional aggregation**
The process of reading data across one level of a hierarchy or auto-level hierarchy, and simultaneously summarizing it across a number of dimensions.

**multilevel aggregation**
The process of reading data across one level of a hierarchy or auto-level hierarchy, and summarizing it to lower levels.

**namespace**
For authentication and access control, a configured instance of an authentication provider that allows access to user and group information. In Framework Manager, namespaces uniquely identify query items, query subjects, and so on. You import different databases into separate namespaces to avoid duplicate names.
In XML and XQuery, a uniform resource identifier (URI) that provides a unique name to associate with the element, attribute, and type definitions in an XML schema or with the names of elements, attributes, types, functions, and errors in XQuery expressions.

**navigator**
A Data Manager utility that allows the user to navigate catalog components, determine component dependencies, and locate components.

**operational key**
The primary key of a table in an operational data store.

**optimal snowflake schema**
A snowflake schema, in which each dimension is represented by a table. Each row represents the leaf level members of the dimension and has columns that contain foreign keys to access all of the higher levels of the dimensions.

**optimal star schema**
A variant of the star schema that includes a main table that contains the keys of all the levels of the delivered hierarchy. It contains the description of only the leaf level. To save storage space, the descriptions of the higher levels reside in supplementary tables.

**orphan**
A member of a hierarchy or auto-level hierarchy for which no explicit parent exists. Data Manager can provide foster parents automatically for such orphans.

**output filter**
A filter used to restrict delivery of data to specific data rows. The output filter takes any expression that evaluates to TRUE or FALSE.

**output level**
In the transformation model, the hierarchy level at which Data Manager is to output data.

**package**
A subset of a model, which can be the whole model, to be made available to the Cognos server.

**parent**
In a hierarchy or auto-level hierarchy, a member that has one or more child members at the level immediately below.

**parent-child schema**
A schema that has a single table per dimension with each row in the table representing a member from the dimension, together with its parent. The table may or may not have a column that identifies the level of each member. A variant of this schema uses one table for all dimensions and has an additional column to identify the dimension.
**parse**
To process an SQL statement to return the result set of columns as written in the SQL statement.

**partitioning**
A technique whereby infrequently used columns can be stored in separate, but linked, tables.

**pivot**
To treat multiple table columns as though they were a single column with multiple values. The specified table columns rotate through 90 degrees to form rows.

**precision**
An attribute of a number that describes the total number of binary or decimal digits.

**prepare**
To send an SQL query to the database, to verify that it is syntactically correct, and returning the result set columns.

**procedure**
A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually has one or more input parameters and yields one or more output parameters.

**ragged hierarchy**
A hierarchy that contains members that have parents at a level higher than the immediate parent level in the hierarchy.

**recursive level**
A hierarchy structured solely in terms of parent-child relationships, for which the user can explicitly name the levels contained in the recursive relationships.

**reference dimension**
A dimensional framework that holds reference structures and templates used to provide information when merging data and delivering dimension data into data marts.

**reference domain**
A dimension in which the set of members is constructed from all the members in a hierarchy or lookup, regardless of whether the fact data includes them.

**reference explorer**
A Data Manager utility that can be used to explore the data and structure of reference structures.

**reference structure**
An object that helps define the dimensional framework. Reference structures include both hierarchies and lookups.
**regular aggregate**
A single aggregate function, used by default to aggregate data across all dimensions of the transformation model.

**reject file**
A text file in which Data Manager stores records that the Data Manager engine rejects. In these files, each rejected record has the reason for rejection as its first field.

**row limit**
The maximum number of source data rows that a DataStream may process.

**run environment**
An environment where a database definition file collects alias definitions that the user can select, using a variable or command-line parameter.

**scale**
The magnitude of the number stored. Storing numbers with precision and scale allows a wide range of numbers to be stored in the same amount of space; however, large numbers are not stored as exact integers.

**script**
A series of commands, combined in a file, that carry out a particular function when the file is run. Scripts are interpreted as they are run.

**semaphore**
A protected variable, used in a UNIX environment, that allows multiple program threads to share the same resource, such as file access, but not simultaneously.

**snowflake schema**
A schema that represents a dimension in a series of tables that correspond to the levels of the dimensions. The primary key of each table is the member ID of each level. Each table has a foreign key to the level above.

**source code control**
A Data Manager utility that allows the user to share catalogs between developers where concurrent access to the catalogs is required.

**SQL Helper**
See SQLTerm.

**SQL query**
Any valid SQL statement in the dialect of the target database. This may be either data definition, for example, CREATE TABLE, or data manipulation, for example, SELECT.
**SQLTerm**
A Data Manager utility that helps the user build SQL statements. It is available whereever SQL statements are required.

**SQLTXT Designer**
A Data Manager utility that allows the user to configure the SQLTXT definition files used within Data Manager to access text files as though they are in an SQL database.

**star schema**
A simple schema that represents each dimension in a single table. Levels of the dimension are represented as columns within the table. The primary key of this table is the member identifier of the lowest dimension level.

**static member**
A member of a reference structure that is defined as part of the specification, and not by reference to tables in a database.

**structure data**
Semi-static data that is used to provide context for other data within a relational database. This is typically used to construct a reference structure. For example, a product master contains a list of products with which the company deals. The product master may also contain supporting information, such as product characteristics.

**subscribed element**
A transformation model element that has been selected for delivery to a target system.

**substitution variable**
A variable with a user-specified value that is used to perform a one-off replacement wherever a substitution is required. Even if the value of the variable changes throughout the process, this is not reflected in the substitution variable.

**surrogate ID**
An artificial identifier used to replace the natural member identifier of a dimension. Whereas the natural member identifier may be text-based, the surrogate identifier is always an integer. Natural member identifiers may have meaning, for example, a product code. However, surrogate identifiers do not.

**surrogate key**
See surrogate ID.

**template**
In report authoring, a reusable report layout or style that can be used to set the presentation of a query or report.
In Data Manager, a component that can be used to define reference structure attributes or dimension table columns together with their semantics.

**transformation**
The process of changing the structure and values of data from one form to another.

**transformation model**
A model within a fact build that is used to manipulate the acquired source data. For example, the transformation model can merge data from different sources or aggregate data.

**transformation model element**
An element of the transformation model in a fact build. A transformation model element can be a dimension, a derived dimension element, a measure an attribute, or a derivation.

**tree pane**
A pane in which objects of a catalog are arranged in a tree-like hierarchical fashion.

**ultimate ancestor**
An ancestor member that resides at the top level of a hierarchy.

**unbalanced hierarchy**
A hierarchy that has leaf nodes at more than one level. The parent of every member comes from the level immediately above.

**unmatched member**
A dimension member that does not relate to any member in the reference data.

**user-defined function**
In Data Manager, an internal or external function created by the user. Internal functions are defined within a Data Manager catalog. External functions are defined in an external, compiled module.

In Cognos Real-Time Monitoring and Cognos Now!, a function defined by the user that provides a mechanism for extending C-SQL for use in formulas, including queries, field expressions, and rules.

**variable**
A representation of a changeable value.

**variable scope**
The collection of Data Manager objects for which a variable is visible.

**visualization pane**
A pane that provides a graphical visualization of an object in the tree pane.
**working memory**

The amount of memory that the Data Manager engine is allowed to use during processing.
Index

A
accepting source data containing unmatched dimension values, 179
accepting unmatched source data, 179
accessing
data, 99
data using pipes, 405
text files using SQLTXT Designer, 397
adding
alert nodes to JobStream, 263
attributes to fact builds, 164
auto-level hierarchies, 95
calculation to derivation, 178
calculation to derived dimension, 178
catalogs to source code control, 348
column names to SQLTerm, 57
columns to SQLTXT Designer table, 403
condition nodes to JobStream, 261
connections, 63
data sources, 143
DataStage nodes to JobStream, 265
DataStream items, 151
date hierarchies, 112
derivations to DataStreams, 155
derivations to fact builds, 165
derived dimensions to fact builds, 167
dimension builds, 235
dimension builds to JobStream, 259
dimension builds using the wizard, 232
dimension delivery to a fact build, 227
dimensions to fact builds, 166
elements to deliveries, 204
email nodes to JobStream, 264
fact builds, 135
fact builds to JobStream, 259
fact build using the wizard, 135
fact delivery to a fact build, 208
filter to Published FM Package data source, 147
hierarchies, 91
JobStream nodes to JobStream, 259
JobStreams, 258
levels to hierarchy, 92
literals to data source, 150
lookups, 98
measures to fact builds, 170
pivots, 363
procedure nodes to JobStream, 261
Published FM Package data sources, 145
recursive levels to hierarchy, 93
reference dimensions, 75
SELECT statements, 55
SQL nodes to JobStream, 262
statements to SQLTerm, 58
static members, 103
table names to SQLTerm, 57
tables to a dimension delivery, 228
tables to dimension builds, 236
unmatched members from multiple fact builds, 181
user-defined variables, 321
additional permissions
DB2, 566
advanced settings for data sources, 144
aggregate rule
definition, 567
specifying, 188
aggregating data, 183
aggregate rule, 188
duplicate data, 192
memory considerations, 184
regular aggregate, 185
aggregation
definition, 567
AIX, 566
alert nodes, 256
adding to JobStream, 263
alias
calling to database, 68
ancestors
definition, 567
application limits, 30
Index

applying
  database schema, 46
  maximum row limit, 158
  output filter, 220
array data type, 543
arrays in external user-defined functions, 313
attribute elements, 163
  adding, 164
  adding to deliveries, 204
  definition, 567
  merging data, 200
attributes
  defining for auto-level hierarchies, 97
  defining for levels, 92
  defining for lookups, 99
  definition, 567
  in dimensions, 78
  mapping to DataStream items, 101
AUDIT_VALUES, 332
audit history
  builds, 285
  icons, 557
  JobStreams, 285
  stop build execution, 291
  viewing audit messages for builds, 289
  viewing audit trail for builds, 288
  viewing JobStream node details from, 290
auditing
  audit message line table, 534
  audit message table, 534
  audit trail table, 533
  component run table, 531
  delivery history table, 533
  job node run table, 535
  specifying details, 287
audit-related variables, 332
  AUDIT_VALUES, 332
audits
  definition, 567
  audit tables, 285, 291, 531
auto-level hierarchies
  acquiring structure data, 99
  circular references, 110
  creating, 95, 97
  defining attributes, 97
  deleting, 119
  disable fostering, 104
  duplicate rows at level, 108
  features, 105
  fostering feature, 106
  foster parents, 103
  multiple parents, 106
  naming foster parents, 104
  non-unique captions, 109
  non-unique IDs, 107
  null/empty captions, 109
  null ID values, 105
  populating, 102
  testing, 117
  top parent ID, 96
  unbalanced hierarchy, 110
auto-level hierarchy
  definition, 567
  automatic table joins
    Fact Build wizard, 135
B
backing up
  catalogs, 360
  catalogs from the command line, 430
backup
  partially importing, 355
balanced hierarchy, 121
balancing
  dense balancing, 122
  hierarchies, 122
behavior types in templates, 80
  changing default names, 81
binary data
  troubleshooting, 565
binary data type, 543
BLOB data type, 543
Boolean data type, 543
breaking, 378
  determining dimensions, 379
  setting breaks, 379
  sorting data, 379
  specifying when, 380
browsing
  log files, 284
build nodes
  adding to JobStream, 259
aspecifying the load required for processing, 269
builds
  definition, 567
executing on remote computer, 279
exexecuting via Data Movement Service, 280
bulk fetch operations, 144
bulk loader
  definition, 567
troubleshooting, 565
business dimensions, 26
definition, 568
business keys, 80
definition, 568
null values, 105
business name
definition, 568

caching reference data, 177
calculation
  adding to derivation, 178
  adding to derived dimension, 178
captions
  definition, 568
  empty, 109
  non-unique, 109
  null, 109
catalogs, 43
  adding to source code control, 348
  advantages over text-based projects, 452
  backing up, 360
  backing up from the command line, 430
  changing properties, 45
  checking into source code control, 352
  closing, 48
  creating, 44
  creating from source code control, 349
  creating standalone from source code control, 353
definition, 568
detaching from source code control, 353
detailing contents, 45
documenting, 45
exporting, 425
importing, 427
listing contents from the command line, 423
managing from command line, 433
obtaining the lastest, 350
opening, 44
remove audit history, 46
remove previous version, 46
remove run details history, 46
restoring, 360
restoring from command line, 431
sharing, 347
source code control, 347
storing, 43
undoing check out, 352
upgrading, 49
upgrading from command line, 432
working with the command line, 452
catalog tables, 291, 531
catbackup command, 430
catdoc command, 435
categories of property variables, 324
catexp command, 425
catimp command, 427
catlist command, 427
catmanage command, 433
catpkgtoxml command, 452
catrestore command, 431
catupgrade command, 432
change
argument order in user-defined functions, 306
catalog properties, 45
column names when delivering data, 215
connection properties, 70
data source properties, 149
default behavior types, 81
element type, 205
log files default directory, 282
order of data in data sources, 159
reference dimension properties, 76
size of Visualization pane, 37
status of condition nodes, 268
template properties, 82
user-defined variables, 323
Visualization pane defaults, 37
char data type, 543
checking
  target table, 275
checking in catalogs, 352
checking out
  components, 351
un Undoing, 352
check only execution mode, 273
child
definition, 568
circular references, 110
CLOB
troubleshooting, 565
CLOB data type, 543
closing
catalogs, 48
COG_ROOT, 324
Cognos data source connections
  adding, 66
Cognos Data Source drivers, 516
Cognos SQL, 40, 63
collation sequences, 64
column names
  adding to SQLTerm, 57
  in SQLTXT Designer, 386
column tokens, 237
definition, 568
commands, 413
catbackup, 430
catdoc, 435
catexp, 425
catimp, 427
catlist, 423
catmanage, 433
catpkgtoxml, 452
catrestore, 431
catupgrade, 432
databuild, 414
dimbuild, 417
dsremote, 422
dstofm, 438
rundsjob, 419
showref, 436
sqlterm, 439
component packages, 354
definition, 568
components
  checking out, 351
  creating packages, 354
definition, 568
duplicating, 40
importing packages, 355
locating, 38
locating dependencies, 39
refreshing the status, 350
selecting multiple, 36
viewing source code control properties, 350
condition nodes, 256
  adding to JobStream, 261
  changing status, 268
configuring
  JobStream execution for IBM Cognos BI load balancing, 281
conformed dimension
definition, 568
connection
  Cognos data source, 66
  published FM package, 67
connection aliases
definition, 569
  using to connect to database, 68
connections, 63
  changing properties, 70
  collation sequences, 64
  creating, 63
  cross platform deployment, 559
definition, 569
deleting, 71
ODBC Data Source Administrator, 71
testing, 70
control attributes
  table deliveries, 212
conventions
  external user-defined functions, 307
converting
  nodes to JobStream nodes, 268
coordinated data marts
definition, 569
copying
  components, 40
  SQLTXT Designer table table, 404
  Visualization pane, 38
create
  Cognos Data Source connection, 66
  published FM package connection, 67
Create a Log File for Expression and Script Tracing, 292
create date, 80
creating, 91
  auto-level hierarchies, 95
catalogs, 44
catalog schema, 46
catalogs from source code control, 349
catalog tables, 46
component packages, 354
connections, 63
database alias definition files, 69, 560
data sources, 143
DataStream filters, 159
DataStream items, 151
date hierarchies, 112
derivations in DataStreams, 155
dimension builds manually, 235
dimension builds using wizard, 232
fact builds, 135
fact builds using the wizard, 133
filters for Published FM Package data sources, 147
hierarchies manually, 91
hierarchies using the Hierarchy wizard, 88
hierarchy levels, 92
internal user-defined functions, 305
JobStreams, 258
lookups, 98, 99
nested JobStreams, 268
 pivots, 363
Published FM Package data sources, 145
recursive hierarchy levels, 93
reference dimensions, 75
SQL statements, 55
SQLTXT database to access text files, 397
standalone catalogs from source code control, 353
templates, 79
user-defined variables, 321
credentials
  definition, 569
cross-platform
  portability, 316
cross platform deployment, 559, 561
database alias definition files, 560
database connections, 559
executing fact builds, 562
password encryption, 562
UNIX file names, 559

current indicator, 80

data
  merging, 200
  partitioning in fact deliveries, 221
  rejected, 371
  rejected duplicate source, 371
  rejected source, 371
  rejecting delivery data, 373
  restricting delivery using level filter, 219
  restricting delivery using output filter, 220
  saving rejected, 372
  vertically partitioning, 212
data access, 99
database alias definition files, 69
creating, 560
  password encryption, 70
database aliases
  definition, 569
database drivers, 515
  DTS package, 518
databases
  connecting with SQLTerm, 439
  limits on table names, 515
  listing data sources, 449
  listing objects, 447
  listing tables, 444
database schema, 46
database schemas
  definition, 569
databuild command, 414
eexecuting fact builds, 414
listing delivery module properties, 415
listing delivery modules, 415
options, 416
data foundation, 25
Data Manager
catalogs, 43
  language, 29
data marts
  private, 114
Data Movement Service
  executing builds, 280
  executing JobStreams, 280
  submitting JobStream build nodes to, 281
Index

data movement specification XML files
  converting from package files, 452
Data Movement tasks
  publishing, 271
data pivoting, 363
data sampling, 157
data sources, 148
  adding, 143
  advanced settings, 144
  bulk fetch operations, 144
  changing order of data, 159
  changing properties, 149
  Cognos SQL, 144
  deleting, 151
  disabling, 150
  enabling, 150
  isolation level, 145
  iterales, 150
  managing, 142
  mapping columns, 152
  parsing, 149
  processing in parallel, 159
  processing in series, 159
  processing order, 159
  Published FM Package, 145
  specifying SQL version, 144
DataStage nodes, 257
  adding to JobStream, 265
DataStream
  definition, 569
DataStream items
  creating, 151
  definition, 569
  deleting, 154
  managing, 151
  renaming, 152
  reordering, 152
DataStreamre, 141
  defining derivations, 155
  executing, 160
  filtering, 159
  literals, 150
  managing data sources, 142
  mapping data sources to, 152
  mapping to and create transformation model elements, 203
mapping to existing transformation model elements, 202
mapping to transformation model, 201
restricting number of rows, 158
sampling data, 157
setting up, 142
viewing, 141
Data Transfer
  fact build, 134
data types, 543
  DBMS drivers, 516
  expressions and scripts, 343
  external user-defined functions, 309
  rules for calculating numbers, 545
  SQLTXT Designer, 389
date data type, 543
date formats, 537
  default century behavior, 539
  symbols, 537
date hierarchies
  creating, 112
  Date Hierarchy wizard, 112
date strings
  SQLTXT Designer, 394
date values
  external user-defined functions, 311
DB_DBALIAS_FILE, 335
DB2
  additional permissions, 566
  DB2 database drivers, 516
  DB2 LOAD delivery module, 456
  DBMS driver
    definition, 569
  DBMS drivers, 515
    Cognos data source, 516
    DB2, 516
    DTS package, 518
    Essbase, 518
    INFORMIX (via Connect), 519
    Microsoft SQL Server (via OLE-DB), 522
    ODBC, 520
    Oracle (via SQL*Net), 521
    Published FM Package, 522
    SAP R/3, 524
    SQLTXT, 525
    Sybase (via CT-Library), 528

586 IBM Cognos Data Manager
Index

DBMS-related variables, 335
   DB_DBALIAS_FILE, 335
   DS_DB2_LOADER, 335
   DS_DBMS_TRIM, 336
   DS_DBMS_TRUNCATION, 336
   DS_INFORMIX_LOADER, 337
   DS_MAX_RECURSION, 337
   DS_MSSQLSERVER_BCP, 337
   DS_ORACLE_LOADER, 337
   DS_REDBRICK_LOADER, 338
   DS_RUNENV, 338
   DS_TERADATA_FASTLOAD, 339
   DS_TERADATA_MULTILOAD, 339
   DS_TERADATA_TPUMP, 339

DDL statements
   exporting, 276

debug-related variables, 330
   DS_REJECT_NULL_VALUE, 330
   MEMBER_FREQUENCY, 330
   OTRACE_FREQUENCY, 330
   REJECT_FILE, 331
   TRACE_FREQUENCY, 331
   TRACE_VALUES, 331

declaring
   external user-defined function, 308
   variables, 319

definition files
   database alias, 69
   SQLTXT, 385

DETERE FROM statement, 58
deleting
   argument in user-defined functions, 306
   auto-level hierarchies, 119
   connections, 71
   data sources, 151
   DataStream items, 154
   deliveries, 225, 230
   delivery filters, 221
   derivations, 157
   dimension builds, 250
   fact builds, 139
   hierarchies, 119
   level filters, 219
   levels, 119
   lookups, 119
   nodes from JobStream, 266
   reference dimensions, 76
   templates, 82
   transformation model elements, 205
   user-defined folders, 35
   user-defined functions, 317
   user-defined variables, 323

delimited records
   delimited columns, 397
   fixed columns, 399

delimiters
   special characters, 388
   SQLTXT Designer, 386, 388

deliveries
   adding dimension delivery, 227
   adding fact delivery, 208
   deleting, 225, 230
   disabling, 224, 229
   saving rejected data, 374

delivering data
   adding a dimension delivery, 227
   adding a fact delivery, 208
   adding tables to dimension delivery, 228
   add missing columns to target table, 215
   changing column names, 215
   defining properties for table deliveries, 209
   deleting level filters, 219
Index

delivery filters, 217
element properties, 214
level filter, 219
multiple tables, 212
output filter, 220
partitioning columns, 221
restricting delivery using level filter, 219
restricting delivery using output filter, 220
specific dimensions and hierarchy levels, 219
subscribing to elements, 212
delivering fact data
table properties, 209
target table name, 209
delivery data
rejecting, 373
delivery filters
applying to a fact delivery, 221
creating, 218
deleting, 221
level filters, 219
output filters, 220
setting up, 217
delivery module property
definition, 569
delivery modules
DB2 LOAD, 456
definition, 569
Essbase Direct Feed, 458
Essbase File Feed, 460
Informix LOAD, 460
listing from command line, 415
listing properties from command line, 415
Microsoft SQLServer BCP Command, 462
Netezza NZLoad, 465
ORACLE SQL*Loader, 468
Red Brick Loader (TMU), 471
Relational Table, 474
specify executables, 511
SQL Server Bulk Copy via API, 489
Sybase ASE BCP Command, 490
Sybase IQ LOAD, 493
Teradata Fastload, 495
Teradata Multiload, 502
Teradata TPump, 505
Text File, 508
TM1 Turbo Integrator, 498
dependencies
locating, 39
user-defined functions, 317
deploying on UNIX using Designer, 559
derivation
definition, 569
derivation elements, 163
adding, 165
adding calculation, 178
adding to deliveries, 204
aggregating data, 183
definition, 570
derivations
defining in DataStreams, 155
deleting, 157
reordering, 157
showing full text, 157
derived dimension elements, 163
adding, 167
adding calculation, 178
adding to deliveries, 204
associating with reference item, 171
definition, 570
setting input and output levels, 173
setting output levels, 175
DESCRIBE statement, 58
Designer window
Tree pane, 33
Visualization pane, 36
detaching
catalogs from source code control, 353
determining
breaking dimensions, 379
dimbuild command, 417
options, 418
dimbuild package
executing, 418
dimensional framework
definition, 570
dimensional history, 241
delivering, 241
setting up in dimension builds, 242
dimension attributes, 78
definition, 570
dimension break
definition, 570
dimension breaks, 378
dimension build nodes, 256
  adding to JobStream, 259
  specifying the load required for processing, 269
dimension builds, 231
  adding tables, 236
  column tokens, 237
  creating manually, 235
  creating using wizard, 232
  defining indexing, 248
  definition, 570
  deleting, 250
  delivering dimensional history, 241
dimensional history, 241
  excluding partially populated rows, 236
  executing, 273
  executing from command line, 417
  execution modes, 273
icons, 554
maintaining tables, 250
rejecting late arriving dimension details, 244
schema types, 233
setting the date source for timestamping data, 249
setting up dimensional history, 242
specifying effective date options, 247
specifying effective start date source, 246
specifying historical options, 244
stop execution, 291
tracking attribute changes, 238
viewing, 232
  viewing audit history for, 285
  viewing audit messages for, 289
  viewing audit trail for, 288
Visualization pane, 232
Dimension Build wizard, 232
dimension deliveries
  adding tables, 228
  column tokens, 237
dimension delivery
  definition, 570
dimension delivery module
  definition, 570
dimension elements, 163
  adding, 166
  adding to deliveries, 204
  associating with reference item, 171
definition, 570
  setting input and output levels, 173
  setting output levels, 175
dimensions, 26
  definition, 570
  delivering specific, 219
disable
  fostering, 104
disabling
  data source, 150
  deliveries, 224, 229
  nodes in JobStreams, 278
distributed handling
  unmatched members, 181
distributed sorting, 64
dividing
  JobStreams, 268
DM_DATASOURCE_CONNECTION_SIGNONS, 324
DM_DISTRIBUTED_HANDLING, 325
DM_EMAIL_VIA_SERVICE, 326
DM_SERVER_CAPACITY, 326
DM_TM1_LOAD_SUPPRESS_ERROR_CODES, 326
documenting
  catalogs from the command line, 435
documenting catalogs, 45
domains
  dynamic, 179
  reference, 183
  size, 183
  type, 183
drivers, 515
  Cognos data source, 516
database listing, 445
  DB2, 516
  DTS package, 518
  Essbase, 518
  INFORMIX (via Connect), 519
  Microsoft SQL Server (via OLE-DB), 522
  ODBC, 520
  Oracle (via SQL*Net), 521
  Published FM Package, 522
  SAP R/3, 524
  SQLTXT, 525
  Sybase (via CT-Library), 528
  TM1, 529
DROP TABLE statement, 58
Index

DS_AUDIT_ID, 329
DS_BREAK_DIMENSION, 333
DS_BUILD_FNAME, 329
DS_BUILD_NAME, 329
DS_CHAR_DEFAULT_PRECISION, 326
DS_DATA_DIR, 326
DS_DATA_TIMESTAMP, 329
DS_DB2_LOADER, 335
DS_DBMS_TRIM, 336
DS_DBMS_TRUNCATION, 336
DS_INFORMIX_LOADER, 337
DS_JOB_AUDIT_ID, 329
DS_JOBSTREAM_FNAME, 329
DS_JOBSTREAM_NAME, 330
DS_LOB_BUF_SIZE, 333
DS_LOB_MAXMEM_SIZE, 334
DS_LOG_DIR, 327
DS_LOG_NAME, 330
DS_MAX_RECURSION, 337
DS_MSSQLSERVER_BCP, 337
DS_ORACLE_LOADER, 337
DS_PIPE_CREATE_WAIT, 328
DS_REDBRICK_LOADER, 338
DS_REJECT_NULL_VALUE, 330
DS_RUN_ID, 330
DS_RUN_TIMESTAMP, 330
DS_RUNENV, 338
DS_TERADATA_FASTLOAD, 339
DS_TERADATA_MULTILOAD, 339
DS_TERADATA_TPUMP, 339
DS_TMP_DIR, 328
DS_UZA_BULKWRITE_ROWS, 328
DS_UZA_FETCH_ROWS, 328
dsb_audit_msg, 46
dsb_audit_msg_line, 46
dsb_audit_trail, 46
dsb_catalog, 46
dsb_component, 46
dsb_component_line, 46
dsb_component_run, 46
dsb_delivery_hist, 46
dsb_jobnode_run, 46
dsb_run_context, 46
dsb_sequence, 46
dsremote command, 422
dstofm command, 438

DTS package, 518
duplicate
   rows at a level, 108
duplicate data, 138
   aggregating, 192
duplicating
   components, 40
   SQLTXT Designer table, 404
dynamic
   domains, 179
   members, 102
dynamic domains
   definition, 571
dynamic members
   definition, 571

E
effective date options
   specifying, 247
effective end date, 80
effective start date, 80
   specifying the source, 246
elements
   adding to deliveries, 204
   attributes, 163
   changing types, 205
   deleting, 205
   derivations, 163
   derived dimensions, 163
   dimensions, 163
   measures, 164
   repositioning, 205
   setting delivery properties, 214
   subscribing to, 212
email nodes, 256
   adding to JobStream, 264
empty captions, 109
enabling
   data source, 150
   deliveries, 224, 229
   source code control, 348
encoding, 374
   reject files, 374
encrypting passwords, 70
engine, 29
environments
  engine, 29
error handling in reference structures, 105
error message codes
  troubleshooting, 565
errors
  JobStreams, 269
  SQLTXT Designer, 394
Essbase
database drivers, 518
Essbase Direct Feed delivery module, 458
Essbase File Feed delivery module, 460
establishing
  dependencies of user-defined functions, 317
excluding
  detail level input data, 176
  input data, 176
  nodes in JobStreams, 278
  partially populated rows from dimension builds, 236
  partially populated rows from dimension deliveries, 228
executing
  builds, 273
  builds default execution method, 281
  builds on remote computers, 279
  builds via Data Movement Service, 280
DataStreams, 160
dimbuild package, 418
dimension builds from command line, 417
excluding nodes in JobStreams, 278
execution modes, 273
fact builds and dimension builds, 273
fact builds from command line, 414
fact builds under UNIX, 562
JobStream package, 420
JobStreams, 276
JobStreams default execution method, 281
JobStreams from command line, 419
JobStreams on remote computers, 279
JobStreams via Data Movement Service, 280
log files, 282
package, 416
remotely from command line, 422
restarting JobStreams, 279
saving to audit tables, 285
SQL queries, 60, 149
  using default settings, 274
executing builds
  checking target table, 275
execution methods
  setting default, 281
execution modes, 273
  check only, 273
  external user-defined functions, 314
  normal, 273
  object creation, 273
exploring
  reference structures, 117
exporting
catalogs, 425
  DDL statements, 276
metadata, 301
expression and script tracing, 292
  example log file, 293
  specify log file details for, 292
expressions
data types, 343
  default values, 345
  in VALUES clauses, 407
  scope, 342
  setting data types, 345
  testing, 342
external user-defined functions, 307
  arrays, 313
  cross-platform portability, 316
  date values, 311
  declaring, 308
  deleting, 317
  execution modes, 314
  header files, 309
  macros, 314
  registering, 315
  rules and conventions, 307
  special data types, 309
  using, 316

F
fact build nodes, 256
fact builds
  accepting source data containing unmatched dimension values, 179
  adding attributes, 164
adding derivations, 165
adding derived dimensions, 167
adding dimension deliveries, 227
adding dimensions, 166
adding elements to deliveries, 204
adding fact deliveries, 208
adding measures, 170
adding unmatched members from multiple builds, 181
aggregating data, 183
associating derived dimension with reference item, 171
associating dimension with reference item, 171
attributes, 163
caching reference data, 177
definition, 571
deleting, 139
derivations, 163
derived dimensions, 163
dimension elements, 163
duplicate data, 138
excluding detail level input data, 176
excluding input data, 176
excluding partially populated rows, 228
executing, 273
executing from command line, 414
execution modes, 273
fine-tuning data structure, 276
IBM Cognos BI Mart (Snowflake), 134
IBM Cognos BI Mart (Star), 134
icons, 552
joining tables in the Fact Build wizard, 135
late arriving facts, 197
mapping the DataStream and create transformation model elements, 203
mapping the DataStream to existing transformation model elements, 202
mapping the DataStream to the transformation model, 201
measures, 164
merging data, 200
Relational Datamart, 134
renaming, 137
repositioning elements, 205
stop execution, 291
timestamping, 137, 197
viewing, 131
viewing audit history for, 285
viewing audit messages for, 289
viewing audit trail for, 288
viewing transformation model, 164
wizard, 133
Fact Build wizard, 133
adding a fact build, 135
automatic table joins, 135
fact build types, 133
fact data
separating from reference data, 113
fact deliveries
adding elements, 204
applying delivery filters to, 221
partitioning data, 221
fact delivery
definition, 571
fact delivery modules
DB2 LOAD, 456
definition, 571
Essbase Direct Feed, 458
Essbase File Feed, 460
Informix LOAD, 460
Microsoft SQLServer BCP Command, 462
Netezza NZLoad, 465
ORACLE SQL*Loader, 468
Red Brick Loader (TMU), 471
Relational Table, 474
SQL Server Bulk Copy via API, 489
Sybase ASE BCP Command, 490
Sybase IQ LOAD, 493
Teradata Fastload, 495
Teradata Multiloader, 502
Teradata TPump, 505
Text File, 508
TM1 Turbo Integrator, 498
fact tables
definition, 571
failed rows
table deliveries, 216
FILENAME, 409
file names
quotation characters in SQLTXT Designer, 388
record types in SQLTXT Designer, 386
wildcards in SQLTXT Designer, 386
FILEROWNUM, 409
filters
DataStream, 159
definition, 571
deleting delivery filters, 221
deleting level filters, 219
output, 220
using level filter to reject data, 373
using output filter to reject data, 374
fine-tuning data structures, 276
float data type, 543
fostering, 103
auto-level hierarchies, 103
disabling, 104
naming foster parents, 104
orphaned members, 106
foster parents, 103
definition, 571
naming, 104
functions
creating internal user-defined functions, 305
cross-platform portability, 316
date values, 311
declaring external user-defined functions, 308
deleting, 317
execution modes in external user-defined functions, 314
external user-defined, 307
header files, 309
internal user-defined, 305
macros in external user-defined functions, 314
maintaining user-defined, 316
registering external user-defined functions, 315
user-defined, 305
using, 316
G
generating surrogate keys, 86
generating test data, 410
H
handling
duplicate data, 138
JobStream errors, 269
rejected data, 371
HASH_TABLE_SIZE, 334
hiding
Visualization pane, 37
hierarchies, 91
acquiring structure data, 99
balancing, 122
constructing, 121
creating auto-level hierarchies, 95
creating date hierarchies, 112
creating hierarchies, 91
creating levels, 92
creating manually, 91
creating recursive levels, 93
definition, 571
deleting, 119
dense balancing, 122
disable fostering, 122
duplicate rows at level, 108
features, 105
flattening, 126
fostering feature, 106
foster parents, 103
Hierarchy wizard, 88
linking a bottom level to, 124
multiple parents, 106
naming foster parents, 104
non-unique captions, 109
non-unique IDs, 107
null/empty captions, 109
null ID values, 105
populating, 102
repositioning levels, 93
testing, 117
transforming, 121
unbalanced hierarchy, 110
hierarchy levels
delivering specific, 219
hierarchy types, 121
Hierarchy wizard, 88
historical data, 238
historical options
specifying in dimension builds, 244
I
IBM Cognos BI Mart (Snowflake) fact build, 134
IBM Cognos BI Mart (Star) fact build, 134

identifiers
definition, 571

IDs
null values, 105
immediate scope, 342
implementing
surrogate keys, 83

importing
catalogs, 427
component packages, 355
delimited records with delimited columns, 397
delimited records with fixed columns, 399
nondelimited records, 401
table definitions, 397
table manually, 402
Import wizard, 397

imposing memory limits, 381

including
substitution variables in SQL queries, 61
unmatched members, 179

indexing
dimension builds, 248
table deliveries, 211

INFORMIX (via Connect)
database drivers, 519
Informix LOAD delivery module, 460

initial
hash table size, 382
page table size, 382
input data
excluding, 176
reducing the amount, 157
input levels, 173
definition, 571

integer data type, 543

internal user-defined functions, 305
creating, 305
deleting, 317
using, 316
interval data type, 543
interval formats, 541
day to second, 541
year to month, 542
intervalYM data type, 543
isolation level, 145

J
JobStream
definition, 572
JobStream nodes, 256
adding to JobStream, 259
nested, 268
specifying the load required for processing, 269
specifying the weight, 269

JobStream package
executing, 420

JobStreams, 255
adding, 258
adding alert nodes, 263
adding condition nodes, 261
adding DataStage nodes, 265
adding dimension builds, 259
adding email nodes, 264
adding fact builds, 259
adding JobStream nodes, 259
adding procedure nodes, 261
adding SQL nodes, 262
alert nodes, 256
condition nodes, 256
converting nodes, 268
DataStage nodes, 257
deleting nodes, 266
dimension build nodes, 256
drilling down to audit history node details, 290
e-mail nodes, 256
error handling, 269
excluding nodes, 278
executing, 276
executing from command line, 419
executing on remote computer, 279
executing via DAta Movement Service, 280
fact build nodes, 256
icons, 556
JobStream nodes, 256
linking nodes, 267
nested JobStreams, 268
nodes, 255
procedure nodes, 256
restarting execution, 279
setting up load control, 269
splitting, 268
SQL nodes, 256
stop execution, 291
variables, 259
viewing audit messages for, 289
viewing audit trail for, 288
JobStream variables
definition, 572
joining
tables in Fact Build wizard, 135

K
keys
definition, 572

L
language, 29
last update date, 80
late arriving fact processing
definition, 572
late arriving facts, 197
level attributes
definition, 572
mapping to DataStream items, 101
level filters, 219
applying, 221
definition, 572
deleting, 219
syntax, 219
using to reject data, 373
levels
acquiring structure data, 99
creating in hierarchies, 92
defining attributes, 92
definition, 572
deleting, 119
repositioning, 93
limited shared memory
AIX, 566
limiting
memory usage, 381
limits, 30
length of tables names, 515
linking
bottom level to an unbalanced hierarchy, 124
condition nodes, 268
nodes in JobStream, 267
links
definition, 572
listing
delivery module properties from command line, 415
delivery modules from command line, 415
literals
adding to data source, 150
definition, 572
load control, 269
locating
component dependencies, 39
components, 38
log files, 282
changing default directory, 282
defining, 283
definition, 572
expression and script tracing, 292
viewing, 284
logging
definition, 572
lookups
acquiring structure data, 99
creating, 98, 99
defining attributes, 99
definition, 573
deleting, 119
duplicate rows at level, 108
features, 105
non-unique captions, 109
null/empty captions, 109
null ID values, 105
populating, 102
testing, 117
M
macros
external user-defined functions, 314
maintaining
dimension tables, 250
surrogate key values, 84
user-defined functions, 316
MAKEROWS database, 410
managing
catalogs from the command line, 433
DataStream items, 151
historical data, 238
Index

memory, 377
manually
  creating dimension builds, 235
  importing table into SQLTXT, 402
mapping
  data sources to DataStream items, 152
  DataStream to transformation model, 201
  definition, 573
  level attributes to DataStream items, 101
master tables, 102
  definition, 573
maximum
  memory usage, 381
  page size, 382
  row limit, 158
measure elements, 164
  adding, 170
  adding to deliveries, 204
  aggregating data, 183
  definition, 573
  merging data, 200
measures
  definition, 573
MEMBER_FREQUENCY, 330
members
  definition, 573
memory, 377
  dimension domains, 377
  initial hash table size, 382
  limits, 381
  maximum page size, 382
  maximum usage, 381
  setting limits, 383
  structure data, 377
MEMORY_LIMIT, 334
memory considerations when aggregating, 184
merge
  definition, 573
merge methods
  definition, 573
merging
  data, 200
  methods, 200
message frequency
  definition, 573
messages
  definition, 573
metadata
  icons, 555
  viewing, 295
metadata collection
  definition, 573
metadata collections
  adding star models to, 298
  creating, 298
metadata dimensions
  creating, 297
  definition, 573
metadata export
  definition, 574
metadata exports, 301
  creating metadata collections for, 298
  creating metadata dimensions for, 297
  partitioned data, 299
metadata schemas
  definition, 574
metadata star
  definition, 574
Microsoft SQL Server (via OLE-DB)
  database drivers, 522
Microsoft SQLServer BCP Command delivery mod-ule, 462
mixing operating systems, 559
models
  definition, 574
multibyte character sets
  quoted identifiers, 41
multi-developer support, 347
  backup and restore catalogs, 359
  component packages, 354
  definition, 574
  source code control, 347
multidimensional
  definition, 574
multidimensional aggregation
  definition, 574
multilevel aggregation
  definition, 574
multi-platform, 31
multiple parents, 106
multi-select, 36

596 IBM Cognos Data Manager
namespaces
  definition, 574
naming
  foster parents, 104
  user-defined variables, 320
navigator, 38
  definition, 575
nested JobStreams, 268
Netezza NZLoad delivery module, 465
Netezza ODBC driver, 566
new catalog, 44
nodes, 255
  alerts, 256
  condition, 256
  DataStage, 257
  dimension builds, 256
  email, 256
  excluding from JobStreams, 278
  fact builds, 256
  JobStreams, 256
  procedure, 256
  SQL, 256
nondelimited records, 401
non-unique
  captions, 109
  IDs, 107
normal execution mode, 273
null
  captions, 109
  ID values, 105
null values
  SQLTXT Designer, 395
number data type, 543
  rules for calculations, 545
numeric data type, 543

opening
catalogs, 44
  OBDC Data Source Administrator, 71
recent catalogs, 45
  source code control system, 352
operating systems
  mixing, 559
operational keys
  definition, 575
optimal snowflake schema, 234
  definition, 575
optimal star schema, 233
optimal star schemas
  definition, 575
Oracle (via SQL*Net)
  database drivers, 521
ORACLE SQL*Loader delivery module, 468
ORDER BY clause, 159
order of precedence
  property variables, 324
  user-defined variables, 320
orphans
  definition, 575
OTRACE_FREQUENCY, 330
output filters, 220
  applying, 221
  definition, 575
  using to reject data, 374
output levels, 175
  definition, 575

package
  executing, 416, 418, 420
packages, 354
  converting to data movement specification XML
    files, 452
    creating, 354
    definition, 575
    importing, 355
PAGE_SIZE, 334
PAGE_TABLE_SIZE, 335
page size
  maximum, 382
page table
  size, 382
Index

parallel processing of data sources, 159
parent-child schema, 234
definition, 575
parents
definition, 575
parse
definition, 575
parsing, 148
data sources, 149
partially populated rows
excluding from dimension builds, 236
excluding from dimension deliveries, 228
partitioned data
metadata exports, 299
partitioning
definition, 576
partitioning data, 221
password encryption, 70, 562
performance-related variables, 333
DS_BREAK_DIMENSION, 333
DS_LOB_BUF_SIZE, 333
DS_LOB_MAXMEM_SIZE, 334
HASH_TABLE_SIZE, 334
MEMORY_LIMIT, 334
PAGE_SIZE, 334
PAGE_TABLE_SIZE, 335
pipes
accessing data using, 405
defining in SQLTXT Designer, 405, 406
example SQLTXT Designer definition files for, 406
test access for, 407
pivot
definition, 576
pivoting data, 363
creating pivots, 363
double pivots, 367
multiple pivots, 369
single pivots, 365
populating reference structures, 102
portability of property variables, 323
precision, 545
definition, 576
prepare
definition, 576
preparing, 148
data sources, 148
primary partitioned table, 221
private data mart, 114
procedure node, 256
adding to JobStream, 261
procedures
definition, 576
processing
data sources in series, 159
late arriving facts, 197
order of data sources, 159
properties
catalog, 45
property variables, 323
categories, 324
order of precedence, 324
portability, 323
Published FM Package connections
adding, 67
Published FM Package data sources
adding, 145
filters, 147
generated query specification, 147
generated SQL or MDX, 148
Published FM Package drivers, 522
publishing Data Movement tasks, 271
Q
quotation characters in SQLTXT Designer, 388
quoted identifiers, 41
R
ragged hierarchy, 121
definition, 576
recent catalogs, 45
record delimiters
SQLTXT Designer, 386
record identity, 221
recording build events
audit trail table, 533
record types in SQLTXT Designer, 386
recursive levels
creating in hierarchies, 93
definition, 576
Red Brick Loader (TMU) delivery module, 471
reducing input data, 157
reference data
  caching on demand, 177
  separating from fact data, 113
reference dimensions
  changing, 76
  creating, 75
  definition, 576
  deleting, 76
  viewing, 75
reference domains, 183
  definition, 576
reference explorer
  definition, 576
Reference Explorer window, 118
reference structures, 87
  acquiring structure data, 99
  circular references, 110
  definition, 576
  duplicate rows at level, 108
  dynamic members, 102
  error handing, 105
  exploring, 117
  features, 105
  fostering feature, 106
  icons, 555
  listing from command line, 436
  multiple parents, 106
  non-unique captions, 109
  non-unique IDs, 107
  null/empty captions, 109
  null ID values, 105
  populating, 102
  static members, 102
  testing, 117
  top parent ID, 96
  unbalanced hierarchy, 110
  viewing, 87
reference surrogate keys, 85
refreshing
  components status, 350
register external user-defined functions, 315
regular aggregate
  definition, 576
  specifying, 185
REJECT_FILE, 331
rejected data, 371
  saving during delivery, 374
reject files
  definition, 577
  encoding, 374
rejecting
  late arriving dimension details, 244
  unmatched members, 372
rejecting data
  delivery data, 373
  duplicate source data, 371
  saving, 372
  source data, 371
  specifying the encoding, 374
  using level filter, 373
  using output filter, 374
Relational Datamart
  fact build, 134
Relational Table delivery module
  update, 483
  update/insert, 476
Relational Table delivery modules, 474
remote computers
  executing builds, 279
  executing JobStreams, 279
remote execution
  command line, 422
remove
  audit history from a catalog, 46
  previous version of catalog, 46
renaming
  DataStream items, 152
  fact builds, 137
  foster parents, 104
repositioning
  DataStream items, 152
  derivations, 157
  fact build elements, 205
  hierarchy levels, 93
  user-defined variables, 323
restarting JobStream execution, 279
restoring
  catalogs, 360
  catalogs from the command line, 431
  package, 360
restricting
  input data, 157
restricting delivery
  using level filter, 219
  using output filter, 220
restricting number of data source rows, 158
revealing
  Visualization pane, 37
rotating data, 363
row limits
  definition, 577
  ROWNUM, 409
rules
  external user-defined functions, 307
rundsjob command, 419
  options, 420
run environments, 561
  definition, 577
run-time variables, 329
  DS_AUDIT_ID, 329
  DS_BUILD_FNAME, 329
  DS_BUILD_NAME, 329
  DS_DATA_TIMESTAMP, 329
  DS_JOB_AUDIT_ID, 329
  DS_JOBSTREAM_FNAME, 329
  DS_JOBSTREAM_NAME, 330
  DS_LOG_NAME, 330
  DS_RUN_ID, 330
  DS_RUN_TIMESTAMP, 330
sample row frequency, 157
sampling data, 157
SAP R/3
  database drivers, 524
saving
  build execution information, 282
  data rejected during delivery, 374
  rejected data, 372
scale, 545
scales
  definition, 577
schema
  listing, 449
  schema types, 233
    optimal snowflake, 234
    optimal star, 233
    parent-child, 234
    snowflake, 234
    star, 233
scope
  expressions and scripts, 342
  of variables, 319
scripts
  data types, 343
  default values, 345
  definition, 577
  scope, 342
  setting data types, 345
  testing, 342
  working with, 451
secondary partitioned table, 221
SELECT FROM statement, 58
selecting
  multiple components, 36
SELECT statements, 55
semaphores
  definition, 577
separating
  reference data and fact data, 113
setting
  data types in expressions and scripts, 345
  date source for timestamping data, 249
  dimension breaks, 379
  error handling in reference structures, 105
  input levels, 173
  memory limits, 383
  output levels, 175
setting up
  DataStreams, 142
sharing catalogs, 347
shell
  SQLTerm, 441
showref command, 436
  options, 437
size
  initial hash table, 382
  initial page table, 382
  maximum page, 382
slowly changing dimensions, See tracking attribute changes
snowflake schema, 234
snowflake schemas
  definition, 577
sorting data, 379
source code control, 347
  adding catalogs, 348
  changing the default directory, 354
  checking in catalogs, 352
  checking out components, 351
  creating a standalone catalog, 353
  creating catalogs, 349
  definition, 577
  detaching a catalog, 353
  enabling, 348
icons, 557
obtaining the latest catalog, 350
opening external system, 352
project planning guidelines, 348
refreshing component status, 350
temporary files, 354
troubleshooting, 565
undoing a catalog check out, 352
viewing component properties, 350
source data, 371
  filtering, 159
  rejecting data, 371
source table joins in the Fact Build wizard, 135
special characters, 388
special columns
  FILENAME, 409
  FILEROWNUM, 409
  ROWNUM, 409
special data types in external user-defined functions, 309
specifying
  aggregate rule, 188
  auditing details, 287
  encoding for reject files, 374
  executables for delivery modules, 511
  input and output levels for fact builds, 173
  regular aggregate, 185
  trace log file details, 292
splitting JobStreams, 268
SQL
  specifying version, 144
SQL Helper
  definition, 577
SQL Helper, See SQLTerm
SQL nodes, 256
  adding to JobStream, 262
SQL queries
  adding SELECT statements, 55
  executing in SQLTerm, 60, 149
  including substitution variables, 61
SQL queries
  definition, 577
SQL scripts, 451
  redirection, 451
  running, 449
SQL Server Bulk Copy via API Delivery
  troubleshooting, 565
SQL Server Bulk Copy via API delivery modules, 489
SQL statements
  creating, 55
SQL support for SQLTXT files, 407
SQLTerm, 55
  adding column names, 57
  adding SELECT statements, 55
  adding statements, 58
  adding table names, 57
  connecting to MAKEROWS database, 411
  definition, 577
  executing queries, 60, 149
  obtaining additional information, 58
sqlterm command
  CONNECT, 443
  DBALIAS, 443
  DESCRIBE, 444
  DIFF, 444
  DISCONNECT, 443
  DRIVERS, 445
  ECHO, 445
  EXIT, 445
  FETCHSIZE, 445
  GO, 446
  HELP, 446
  ISOLEVEL, 446
  NATIVE SQL, 448
  NUMBER, 447
  OBJECTS, 447
  OUTPUT, 448
  PROGRESS, 448
  PROMPT, 448
  QUIT, 449
Sybase (via CT-Library)
database drivers, 528
Sybase ASE BCP Command delivery module, 490
Sybase IQ LOAD delivery module, 493
syntax for level filters, 219
system variables, 324
  COG_ROOT, 324
  DM_DATASOURCE_CONNECTION_SIGNONS, 324
  DM_DISTRIBUTED_HANDLING, 325
  DM_EMAIL_VIA_SERVICE, 326
  DM_SERVER_CAPACITY, 326
  DM_TM1_LOAD_SUPPRESS_ERROR_CODES, 326
  DS_CHAR_DEFAULT_PRECISION, 326
  DS_DATA_DIR, 326
  DS_LOG_DIR, 327
  DS_PIPE_CREATE_WAIT, 328
  DS_TMP_DIR, 328
  DS_UDA_BULKWRITE_ROWS, 328
  DS_UDA_FETCH_ROWS, 328
  TEMP, 328
  TM1_PATH, 329
  TMPDIR, 328

T

table deliveries
  control attributes, 212
  defining properties, 209
  handling failed rows in, 216
  indexing, 211

table names
  adding to SQLTerm, 57
  limits, 515

tables
  adding to dimension builds, 236
  adding to dimension deliveries, 228
  audit, 291, 531
  catalog, 291, 531
  creating catalog tables, 46

target table
  adding missing columns to, 215
  checking, 282

TEMP, 328

templates, 77
  behavior types, 80

changing default behavior, 81
changing properties of, 82
creating, 79
definition, 578
deleting, 82
temporary files
  changing default directory for source code control, 354
  source code control, 354
Teradata Fastload delivery module, 495
Teradata Multiload delivery module, 502
Teradata Tpump delivery module, 505
testing
  connections, 70
data types in expressions and scripts, 343
default values in expressions and scripts, 345
expressions and scripts, 342
reference structures, 117
scope of expressions and scripts, 342
SQLTXT Designer, 410
text-based projects, 30
Text File delivery module, 508
text file format, 386
time data type, 543
time formats, 540
timestamping
  dimension data in fact builds, 137
  fact builds, 197
timestamping data
  setting the date source, 249

TM1
  database drivers, 529
  TM1_PATH, 329
  TM1 Turbo Integrator delivery module, 498
  TMPDIR, 328
toolbar buttons, 549
top parent ID, 96
top scope, 342
TRACE_FREQUENCY, 331
TRACE_VALUES, 331
tracking attribute changes, 238
trailing spaces, 162
transaction
  isolation level, 145
  transaction tables, 102
transformation
definition, 579
transformation model, 163, 205
  adding attributes, 164
  adding derivations, 165
  adding derived dimensions, 167
  adding dimensions, 166
  adding elements to deliveries, 204
  adding measures, 170
  associating elements with reference item, 171
  attributes, 163
  changing element types, 205
definition, 579
  deleting elements, 205
  derivations, 163
  derived dimensions, 163
  dimensions, 163
  excluding input data, 176
  mapping DataStream, 201
  measures, 164
  viewing, 164
transformation model element
definition, 579
Tree pane, 33
  Builds and JobStream folder, 33
  Library folder, 34
  Metadata folder, 34
tabs, 33
  user-defined folders, 34
tree panes
definition, 579
troubleshooting
32 bit, 565
64 bit, 565
binary data, 565
bulk loader, 565
CLOB, 565
error message codes, 565
source code control, 565
SQL Server Bulk Copy via API Delivery, 565
writing large objects to DB2, 565
TRUNCATE TABLE statement, 58
truncating table names, 515
tuning large source extractions, 144
types of fact build
  Data Transfer, 134
IBM Cognos BI Mart (Snowflake), 134
IBM Cognos BI Mart (Star), 134
Relational Datamart, 134
U
ultimate ancestors
definition, 579
unbalanced hierarchies, 110
delivering, 228, 236
  linking a bottom level to, 124
unbalanced hierarchy, 121
definition, 579
UNIX
  executing fact builds, 562
  file names, 559
unmatched members
definition, 579
distributed handling of, 181
  including, 179
  rejecting, 372
update/insert delivery method, 476
update delivery method, 483
upgrade considerations
  command lines in scripts, 52
  Data Manager Network Services, 53
  DB-LIB connections, 53
  metadata exports, 53
  source code control projects, 52
  SQLTXT Designer definition files, 52
upgrading, 49
  catalogs, 49
  failure, 51
  importing package, 52
  post upgrade considerations, 52
  restoring backups, 52
upgrading catalogs from the command line, 432
user-defined folders
deleting, 35
  moving builds and JobStreams into, 35
  setting up, 34
  sub folders, 34
  user-defined functions, 305
  arrays, 313
  changing argument order, 306
  creating internal, 305
  cross platform portability, 316
date values, 311
declaring external, 308
definition, 579
deleting, 317
deleting arguments, 306
dependencies, 317
external, 307
header files, 309
internal, 305
maintaining, 316
special data types, 309
using, 316
user-defined variables, 320
adding, 321
changing, 323
deleting, 323
names, 320
order or precedence, 320
repositioning, 323
using
   Designer to deploy on UNIX, 559
user-defined functions, 316

V
VALUES clauses, 407
variable notation, 320
variables, 319
   adding, 321
   audit-related, 332
   changing, 323
   DBMS-related, 335
debug-related, 330
declaring, 319
definition, 579
deleting, 323
JobStreams, 259
names, 320
notation, 320
performance-related, 333
property, 323
repositioning, 323
run-time, 329
scope, 319
substitution variables, 320
system, 324
user-defined, 320

variable scopes
   definition, 579

viewing
   audit history for builds, 285
   audit history for JobStreams, 285
   DataStreams, 141
dimension builds, 232
   fact builds, 131
   generated query specification for Published FM
   Package data source, 147
generated SQL or MDX for Published FM Package
data sources, 148
   JobStreams, 257
   log files, 284
   metadata, 295
   reference dimensions, 75
   reference structures, 87
   source code control properties, 350
   transformation model, 164

Visualization pane, 36
   changing defaults, 37
   changing size, 37
copying, 38
dimension build icons, 554
dimension builds, 232
fact build icons, 552
fact delivery tab, 133
hide and reveal, 37
JobStream icons, 556
mapping tab, 132
metadata icons, 555
reference structure icons, 555
reference structures, 87
source code control icons, 557
transformation model, 164
transformation model tab, 132
viewing fact builds, 131

visualization panes
   definition, 579

W
weight
   specifying for JobStream nodes, 269
wildcards in SQLTXT Designer, 386

wizards
   Date Hierarchy, 112
Index

Dimension Build, 232
Fact Build, 133
Hierarchy, 88
Import wizard, 397
working memory
definition, 579