

ActuateOne™

One Design
One Server
One User Experience

Designing BIRT Information Objects

Information in this document is subject to change without notice. Examples provided are fictitious. No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, for any purpose, in whole or in part, without the express written permission of Actuate Corporation.

© 1995 - 2013 by Actuate Corporation. All rights reserved. Printed in the United States of America.

Contains information proprietary to:

Actuate Corporation, 951 Mariners Island Boulevard, San Mateo, CA 94404

www.actuate.com

The software described in this manual is provided by Actuate Corporation under an Actuate License agreement. The software may be used only in accordance with the terms of the agreement. Actuate software products are protected by U.S. and International patents and patents pending. For a current list of patents, please see <http://www.actuate.com/patents>.

Actuate Corporation trademarks and registered trademarks include:

Actuate, ActuateOne, the Actuate logo, Archived Data Analytics, BIRT, BIRT 360, BIRT Analytics, The BIRT Company, BIRT Data Analyzer, BIRT iHub, BIRT Performance Analytics, Collaborative Reporting Architecture, e.Analysis, e.Report, e.Reporting, e.Spreadsheet, Encyclopedia, Interactive Viewing, OnPerformance, The people behind BIRT, Performancesoft, Performancesoft Track, Performancesoft Views, Report Encyclopedia, Reportlet, X2BIRT, and XML reports.

Actuate products may contain third-party products or technologies. Third-party trademarks or registered trademarks of their respective owners, companies, or organizations include:

Mark Adler and Jean-loup Gailly (www.zlib.net): zlib. Adobe Systems Incorporated: Flash Player. Amazon Web Services, Incorporated: Amazon Web Services SDK, licensed under the Apache Public License (APL). Apache Software Foundation (www.apache.org): Ant, Axis, Axis2, Batik, Batik SVG library, Commons Command Line Interface (CLI), Commons Codec, Crimson, Derby, Hive driver for Hadoop, Pluto, Portals, Shindig, Struts, Tomcat, Xalan, Xerces, Xerces2 Java Parser, and Xerces-C++ XML Parser. Castor (www.castor.org), ExoLab Project (www.exolab.org), and Intalio, Inc. (www.intalio.org): Castor. Day Management AG: Content Repository for Java. Eclipse Foundation, Inc. (www.eclipse.org): Babel, Data Tools Platform (DTP) ODA, Eclipse SDK, Graphics Editor Framework (GEF), Eclipse Modeling Framework (EMF), and Eclipse Web Tools Platform (WTP), licensed under the Eclipse Public License (EPL). Gargoyle Software Inc.: HtmlUnit, licensed under Apache License Version 2.0. GNU Project: GNU Regular Expression, licensed under the GNU Lesser General Public License (LGPLv3). HighSlide: HighCharts. Jason Hsueh and Kenton Varda (code.google.com): Protocol Buffer. IDAutomation.com, Inc.: IDAutomation. IDRsolutions Ltd.: JBIG2, licensed under the BSD license. InfoSoft Global (P) Ltd.: FusionCharts, FusionMaps, FusionWidgets, PowerCharts. Matt Inger (sourceforge.net): Ant-Contrib, licensed under Apache License Version 2.0. Matt Ingenthron, Eric D. Lambert, and Dustin Sallings (code.google.com): Spymemcached, licensed under the MIT OSI License. International Components for Unicode (ICU): ICU library. jQuery: jQuery, licensed under the MIT License. Yuri Kanivets (code.google.com): Android Wheel gadget, licensed under the Apache Public License (APL). LEAD Technologies, Inc.: LEADTOOLS. The Legion of the Bouncy Castle: Bouncy Castle Crypto APIs. Bruno Lowagie and Paulo Soares: iText, licensed under the Mozilla Public License (MPL). Microsoft Corporation (Microsoft Developer Network): CompoundDocument Library. Mozilla: Mozilla XML Parser, licensed under the Mozilla Public License (MPL). MySQL Americas, Inc.: MySQL Connector. Netscape Communications Corporation, Inc.: Rhino, licensed under the Netscape Public License (NPL). OOPS Consultancy: XMLTask, licensed under the Apache License, Version 2.0. Oracle Corporation: Berkeley DB, Java Advanced Imaging, JAXB, JDK, Jstl. PostgreSQL Global Development Group: pgAdmin, PostgreSQL, PostgreSQL JDBC driver. Progress Software Corporation: DataDirect Connect XE for JDBC Salesforce, DataDirect JDBC, DataDirect ODBC. Rogue Wave Software, Inc.: Rogue Wave Library SourcePro Core, tools.h++. Sam Stephenson (prototype.conio.net): prototype.js, licensed under the MIT license. Sencha Inc.: Ext JS, Sencha Touch. ThimbleWare, Inc.: JMemcached, licensed under the Apache Public License (APL). World Wide Web Consortium (W3C) (MIT, ERCIM, Keio): Flute, JTIty, Simple API for CSS. XFree86 Project, Inc.: (www.xfree86.org): xvfb. ZXing authors (code.google.com): ZXing, licensed under the Apache Public License (APL).

All other brand or product names are trademarks or registered trademarks of their respective owners, companies, or organizations.

Document No. 130131-2-731301 January 23, 2013

Contents

About <i>Designing BIRT Information Objects</i>	ix
--	-----------

Part 1

Creating information objects using the IO Design perspective

Chapter 1

Introducing the IO Design perspective	3
--	----------

About information objects	4
About creating information objects	4
About the IO Design perspective	4
Displaying hidden messages	6

Chapter 2

Creating projects, data connection definitions, and maps	9
---	----------

Creating an Actuate BIRT project	10
Propagating column and parameter renaming and deletion	10
Creating a data connection definition	11
Creating a data connection definition for a database	12
Creating a data connection definition for an ODA data source	15
About connection properties	17
About the IANAAppCodePage property	24
About Informix database connections	27
Specifying a production database schema	28
Encrypting and decrypting data source connection property values	28
Understanding the encryption extension point plug-in	28
Extending the encryption extension point plug-in	30
Troubleshooting an encryption extension	39
Externalizing data source connection property values	39
About the data source connection configuration file	40
Externalizing connection property values for a preconfigured connection type	41
Externalizing connection property values for a configurable connection type	43
Externalizing connection property values for an ODA connection type	44
Creating maps	45
Creating a map of a database table or view	45
Updating a map of a database table or view	48
Creating a map of a native SQL query	51
Creating a map of a stored procedure result set	54
Creating a map of an ODA data source query result set	60

Chapter 3

Creating information objects	65
Creating an information object	66
Creating a graphical information object query	67
Using the expression builder	67
Choosing maps and information objects	68
Defining output columns	69
Creating and displaying column categories	71
Setting column properties	74
About column property inheritance	79
Creating a filter for use in report designs	81
Specifying a join	83
About joins	84
Optimizing joins	86
Using join algorithms	86
Improving the selectivity of a join	88
Creating a Cartesian join	89
Filtering data	90
Creating a filter condition	90
Creating multiple filter conditions	98
Prompting for filter values	101
Grouping data	102
Creating a GROUP BY clause	103
Removing a column from the GROUP BY clause	105
Filtering on an aggregate column	107
Defining parameters	108
Specifying a parameter's prompt properties	110
Setting parameter properties	113
Setting source parameters	115
Synchronizing source parameters	116
Creating a textual information object	117
Displaying output columns	119
Displaying parameters	120
Displaying and testing information object output	120
Displaying a data source query	121
Understanding query execution plan operators	123
Understanding node operators	124
Augment	124
Box	124
CallExecutionUnit	124
DependentJoin	124
Dup	124

Materialize	125
MergeJoin	125
Move	125
MultiAugment	125
Nest	125
NestedLoopJoin	126
Project	126
Select	126
Sort	126
Union	126
Understanding leaf operators	126
FakeData	126
FakeFileData	126
IteratorAsLeaf	126
NoOp	127
ODA	127
SortedOuterUnion	127
SQL	127
Storing a query plan with an information object	127
Saving an information object's query plan	128
Saving query plans for source and dependent information objects	129
Deleting an information object's query plan	130
Localizing an information object	130
Chapter 4	
Building and publishing a project	137
Building a project	138
Propagating column and parameter property values	138
Publishing a project	138
Publishing information object files as resources	140
Publishing information object files as non-resources	142
Downloading files from an Encyclopedia volume	144
Chapter 5	
Assessing the impact of project changes	147
About project dependencies	148
Searching for data connection definitions, maps, and columns	148
Displaying the project model diagram	150
Assessing the impact of a change on files in an Encyclopedia volume	153
Downloading files from an Encyclopedia volume	153
Determining the dependencies between project files	154
Generating an impact report	155

Chapter 6	
Actuate SQL reference	157
About Actuate SQL	158
Differences between Actuate SQL and ANSI SQL-92	158
Limitations compared to ANSI SQL-92	158
Extensions to ANSI SQL-92	159
Database limitations	162
Actuate SQL syntax	162
Actuate SQL grammar	164
Using white space characters	168
Using keywords	168
Using comments	169
Specifying maps and information objects in Actuate SQL queries	169
Using identifiers in Actuate SQL	169
Using column aliases in Actuate SQL	169
Specifying parameter values	170
Using subqueries in Actuate SQL	171
Using derived tables in Actuate SQL	172
Data types and data type casting	172
Facets	172
Casting rules	173
String comparison and ordering	174
Functions and operators	175
Comparison operators: =, <>, >=, >, <=, <	175
Range test operator: BETWEEN	175
Comparison operator: IN	176
Arithmetic operators: +, -, *, /	176
Numeric functions	177
FLOOR, CEILING, MOD	177
ROUND	178
POWER	178
Null test operators: is [not] null	179
Logical operators: and, or, not	179
String functions and operators	179
Case conversion functions: UPPER, LOWER	180
Concatenation operator:	180
Length function: CHAR_LENGTH	180
LIKE operator	181
Substring functions: LEFT, RIGHT, SUBSTRING	181
Trimming functions: LTRIM, RTRIM, TRIM	182
Search function: POSITION	183
Timestamp functions	184
CURRENT_TIMESTAMP	184

CURRENT_DATE	185
DATEADD	185
DATEDIFF	185
DATEPART	186
DATESERIAL	186
Aggregate functions: COUNT, MIN, MAX, SUM, AVG	187
System function: CURRENT_USER	188
Providing query optimization hints	189
Indicating that a table in a join is optional	189
Using the OPTIONAL keyword with a computed field	190
Using the OPTIONAL keyword with parentheses ()	191
Using the OPTIONAL keyword with aggregate functions	193
Specifying the cardinality of a join	194
Using pragmas to tune a query	195
Disabling cost-based optimization	195
Disabling indexing	197
Specifying a threshold value for indexing	197

Part 2

Configuring database types

Chapter 7

Understanding database types 201

About database types	202
About connection types	202
About mappings	203
About preconfigured database types	204
DB2 data type mapping and issues	205
Informix data type mapping and issues	206
Oracle data type mapping and issues	207
SQL Server data type mapping and issues	209
Sybase data type mapping and issues	210
About configurable database types	212
Working with XML files	214

Chapter 8

Configuring connection types 217

About configuring connection types	218
JDBC driver requirements and installation	218
JDBC driver requirements	219
Installing a JDBC driver	220
Working with datasources.xml	220

Configuring connection types: ConnectionTypes element	220
ConnectionType child element: JDBCDriver	221
ConnectionString element	222
ConnectionType child element: CatalogFilter	223
ConnectionType child element: ConnectionParams	223
Configuring database types: DatabaseTypes element	224
 Chapter 9	
Mapping data types	227
About data type mapping	228
DataTypeMapper element	229
MaxSize attribute	230
DataType child element: Aliases	231
 Chapter 10	
Mapping functions and operators	233
About mapping functions and operators	234
About ODBC escape sequences	234
Disabling the default mapping for a function	236
Differences between Actuate SQL functions and database functions	236
About Generic_ODBC mappings.xml	236
Syntax for mapping functions and operators	237
Mapping functions and operators: FunctionMapping element	238
About function templates	239
Example: Mapping the POWER function	239
Example: Mapping the DATEDIFF function with date part yyyy	240
Example: Disabling the POSITION function	241
Mapping Boolean operators: BooleanOpMapper element	241
Example: Mapping the NOT operator	242
Mapping comparison operators: ComparisonOpMapper element	242
Example: Mapping the <> operator	242
Mapping arithmetic operators: ArithOpMapper element	243
Example: Mapping the negation operator	243
Mapping numeric functions: NumericFuncMapper element	244
Example: Mapping the POWER function	244
Mapping string functions: BasicStringFuncMapper element	245
Example: Mapping the CHAR_LENGTH function	245
Mapping substring functions: SubStringFuncMapper element	245
Example: Mapping the POSITION function	246
Mapping the LIKE operator: LikeOpMapper element	246
Example: Mapping the LIKE operator	247
Example: Changing the escape character	248
Example: Disabling the LIKE operator	248

Example: Specifying additional special characters	248
Mapping DATEPART functions: DatePartMapper element	248
Example: Mapping the DATEPART functions	249
Mapping date subtraction functions: DateDiffMapper element	249
Examples: Mapping the DATEDIFF function with date part yyyy	250
Mapping date addition functions: DateAddMapper element	251
Example: Mapping the DATEADD functions	251
Mapping date serialization functions: DateSerialMapper element	252
Example: Disabling the DATESERIAL functions	252
Mapping NULL functions: NullFuncMapper element	252
Example: Disabling the CAST (NULL AS ...) functions	253
Mapping conditional functions: CondFuncMapper element	254
Example: Mapping the CASE statement	254
Mapping aggregate functions: AggrFuncMapper element	254
Example: Mapping the AVG function	255
Mapping multi-row Boolean operators: MultiRowBoolFuncMapper element	255
Mapping cast functions: CastFuncMapper element	256
Example: Mapping the CAST functions	257
Using operators in a mapping	258
Symbolic operators require parentheses	258
Negative sign must be followed by a space	258
Less than (<) and greater than (>) symbols must be escaped	258
Example: Mapping the not-equal-to operator	259
Example: Mapping the CONCAT function	259
Example: Mapping the DATEDIFF function	259
Example: Mapping the CHAR_LENGTH function	259
Example: Mapping the negative sign (-)	260
Using initialization statements	260
Example: Specifying the behavior of concatenation with NULL	260

Chapter 11

Mapping literals and clauses	261
Mapping literals: LiteralMapper element	262
Template format for VARCHAR literals	262
Template format for TIMESTAMP literals	262
Example: Mapping VARCHAR and TIMESTAMP literals	262
Mapping clauses	262
Mapping the ORDER BY clause: OrderByClauseMapper element	263
UseSelectedItemIndexes attribute	263
PushComplexExprs attribute	263
Mapping the GROUP BY clause: GroupByClauseMapper element	263
UseSelectedItemIndexes attribute	264
PushComplexExprs attribute	264

Chapter 12

Working with collations and byte-based strings 265

Working with collations266

 About Integration service collations266

 About database collations267

 About collation implementations268

 Specifying the Integration service and database collations269

Working with byte-based strings269

Index 271

About Designing BIRT Information Objects

Designing BIRT Information Objects provides information about using the IO Design perspective to create information objects and publish them to an Actuate BIRT iHub Encyclopedia volume. This manual also describes how to configure a database type for use with the IO Design perspective.

Designing BIRT Information Objects includes the following chapters:

- *About Designing BIRT Information Objects*. This chapter provides an overview of this guide.
- *Part 1. Creating information objects using the IO Design perspective*. This part explains how to create information objects using the IO Design perspective.
- *Chapter 1. Introducing the IO Design perspective*. This chapter defines the term information object and describes the IO Design perspective.
- *Chapter 2. Creating projects, data connection definitions, and maps*. This chapter describes how to create a project, data connection definitions, and maps prior to creating information objects.
- *Chapter 3. Creating information objects*. This chapter describes how to create information objects.
- *Chapter 4. Building and publishing a project*. This chapter describes how to build and publish an Actuate BIRT project to an Encyclopedia volume.
- *Chapter 5. Assessing the impact of project changes*. This chapter describes how to assess the impact of a change to a file, column, or parameter on other files in an Actuate BIRT project.
- *Chapter 6. Actuate SQL reference*. This chapter describes the differences between Actuate SQL and ANSI SQL-92.
- *Part 2. Configuring database types*. This part explains how to configure a database type for use with the IO Design perspective.

- *Chapter 7. Understanding database types.* This chapter defines the term database type, describes the preconfigured database types, and gives an overview of database type configuration.
- *Chapter 8. Configuring connection types.* This chapter gives the requirements for JDBC drivers and describes how to configure a connection type.
- *Chapter 9. Mapping data types.* This chapter explains how database data types are mapped to Actuate SQL data types at design time.
- *Chapter 10. Mapping functions and operators.* This chapter explains how to map Actuate SQL functions and operators to their database equivalents.
- *Chapter 11. Mapping literals and clauses.* This chapter explains how to map Actuate SQL string and timestamp literals and GROUP BY and ORDER BY clauses to their database equivalents.
- *Chapter 12. Working with collations and byte-based strings.* This chapter explains how to choose Integration service and database collations and how to work with a database that processes strings by byte.

Part One

**Creating information objects using the
IO Design perspective**

Introducing the IO Design perspective

This chapter contains the following topics:

- About information objects
- About creating information objects
- About the IO Design perspective
- Displaying hidden messages

About information objects

Like a view in a relational database, an information object is a named SQL query. An information object can retrieve data using:

- Database tables and views
- Stored procedures
- ODA data sources, including Web Services and XML
- Other information objects

An information object can retrieve data from more than one data source.

Report developers use information objects when they access data with BIRT Designer Professional. Business users use information objects when they access data with BIRT Studio.

About creating information objects

The IO Design perspective in BIRT Designer Professional enables you to create information objects. Before you can create an information object, you must:

- Create data connection definitions for the data sources.
- Create maps:
 - Create maps to represent database tables and views.
 - Create external procedures and maps to represent stored procedures and ODA data source queries and their result sets.

Do not use the following characters in resource names in the IO Design perspective:

- < (less than)
- > (greater than)
- ' (single quote)
- " (double quote)

About the IO Design perspective

The IO Design perspective is a perspective in BIRT Designer Professional, which is built on the Eclipse integrated development environment. For information

about perspectives and other Eclipse features, see the *Workbench User Guide* in the online help.

The default IO Design perspective appears as shown in Figure 1-1.

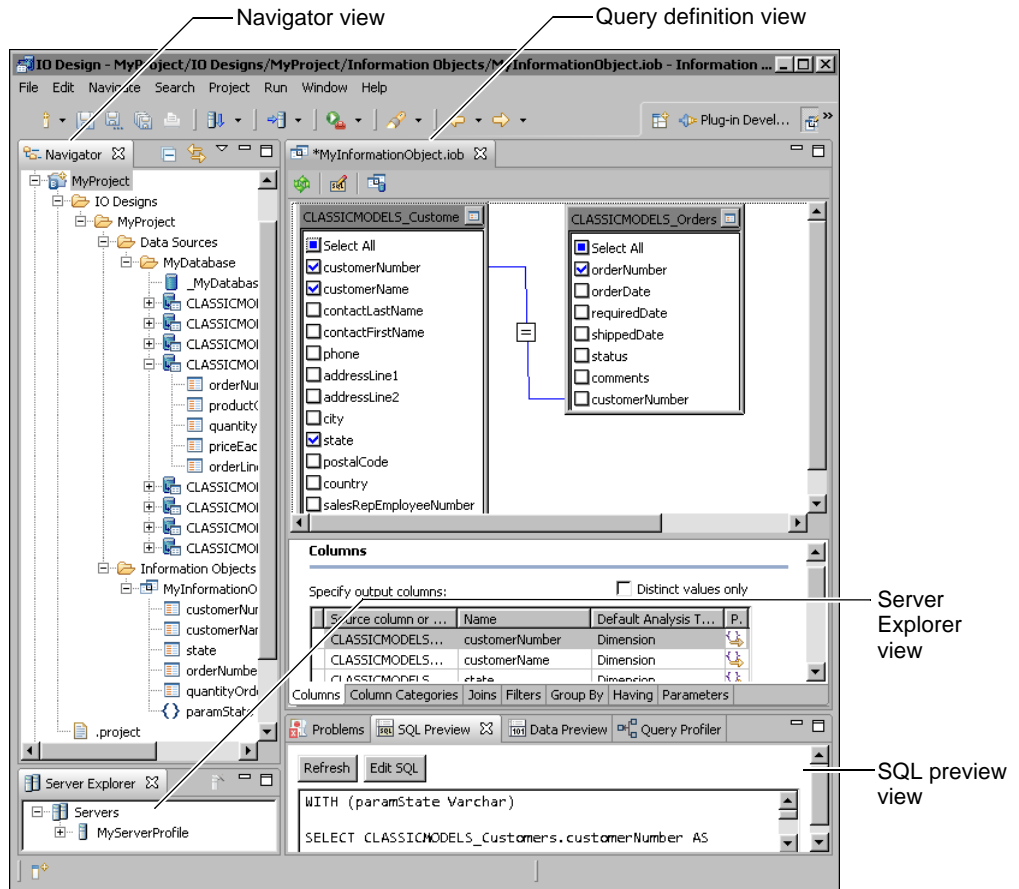


Figure 1-1 The default IO Design perspective

The IO Design perspective displays the following views:

- **Navigator view**
The Navigator view displays the contents of your projects. A project consists of data connection definitions, maps, information objects, and report designs.
- **Query definition view**
You create the information object's SQL query, either graphically or textually, in the query definition view.

- Properties view
The Properties view displays the properties of the selected item, for example a column or table. Figure 1-1 does not show the Properties view.



To toggle the display of properties in categories, choose Show Categories.

- Problems view
The Problems view displays error messages.
- SQL Preview view
The SQL Preview view displays the information object's SQL query.
- Data Preview view
The Data Preview view displays map or information object output.
- Query Profiler view
The Query Profiler view displays the query execution plan.
- Server Explorer view
The Server Explorer view displays your iHub profiles.

Displaying hidden messages

In the IO Design perspective, many dialogs contain a Do not show this message again check box. For example, Figure 1-2 shows the Reminder message for the New Maps dialog. If you check the check box, the message does not appear again.

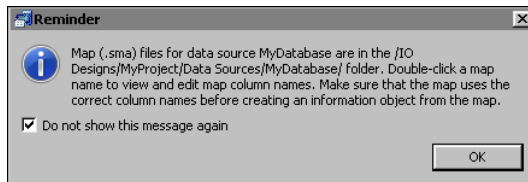


Figure 1-2 Reminder message for New Maps dialog

To display the message again, you must remove it from the Hidden Messages list, shown in Figure 1-3.

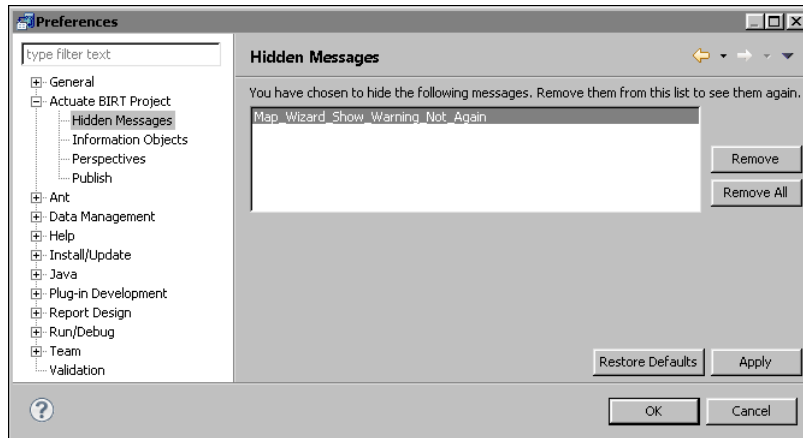


Figure 1-3 Hidden Messages

How to display a hidden message

- 1 Choose Window→Preferences.
- 2 In Preferences, choose Actuate BIRT Project→Hidden Messages.
- 3 Select the appropriate message key. Hover the mouse over a message key to display the message.
- 4 Choose Remove. Choose OK.

Creating projects, data connection definitions, and maps

This chapter contains the following topics:

- Creating an Actuate BIRT project
- Propagating column and parameter renaming and deletion
- Creating a data connection definition
- Encrypting and decrypting data source connection property values
- Externalizing data source connection property values
- Creating maps

Creating an Actuate BIRT project

An Actuate BIRT project is a container for data connection definitions, maps, information objects, and report designs. Figure 2-1 shows the project folder and .project file in Navigator. The .project file is automatically created when you create a project. It contains a description of the project, including the project name, build command, and nature.



Figure 2-1 Project folder with .project file

How to create an Actuate BIRT project

- 1 Choose File→New→Actuate BIRT Project.
- 2 In New Project—Actuate BIRT Project, as shown in Figure 2-2:
 - In Project name, type the name of the project.
 - If you do not want to create the project in the default location, deselect Use default location and type or browse to a different location.
 - Choose Finish.
The project appears in Navigator.

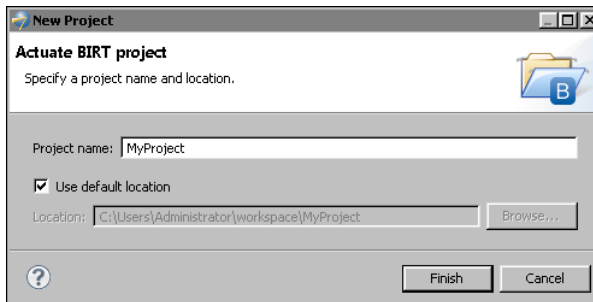


Figure 2-2 Specifying a project's name and location

Propagating column and parameter renaming and deletion

By default, when you rename a column or parameter in a source map or information object, the name change is propagated to any dependent information objects as long as the column name in the dependent information object is not modified. Column names that appear in expressions, including computed

column and filter expressions, are also updated. The IO Design perspective does not check for column name duplication in a dependent information object. When you compile the information object, column name duplication is reported as an error.

By default, when you delete a column or parameter in a source map or information object, the deletion is not propagated to dependent information objects. If you override this behavior, the deletion is propagated to any dependent information objects as long as the column name in the dependent information object is not modified. Filters on a deleted column are deleted. Computed columns and filter expressions that contain the column name are not deleted.

To override the default behavior, choose Window>Preferences>Actuate BIRT Project>Information Objects and make the appropriate selections. Figure 2-3 shows the default settings for column and parameter renaming and deletion.

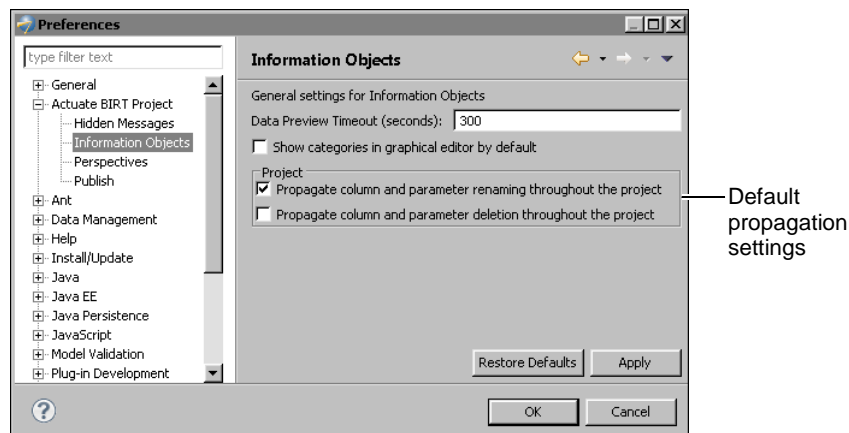


Figure 2-3 Default propagation settings

Creating a data connection definition

A data connection definition defines the connection properties for a data source. Connection properties include connection type, security policy, user name, and password. The first time you create a data connection definition for a project, the IO Design perspective creates the IO Designs folder hierarchy in the project folder. For example, for the project MyProject, the IO Design perspective creates the folder hierarchy /IO Designs/MyProject/Data Sources. The folder hierarchy contains the MyProject subfolder because all information object files are published to the IO Designs subfolder in the Encyclopedia volume's Resources folder using the same relative path used in the workspace, and the project name is required to distinguish one project's information object files from another's.

The IO Design perspective then creates a subfolder in the project's Data Sources folder to contain the data connection definition file. For example, if you create a data connection definition for the database MyDatabase, the IO Design perspective creates the subfolder MyDatabase and the data connection definition file _MyDatabase.dcd, as shown in Figure 2-4.

A data connection definition file name has a .dcd extension. Data connection definition file names are not case-sensitive.

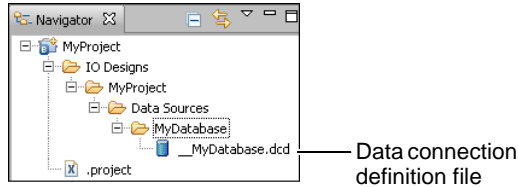


Figure 2-4 Data connection definition file and IO Designs folder hierarchy

Creating a data connection definition for a database

The IO Design perspective uses an ODBC or JDBC driver to connect to a database. The preconfigured database types are:

- DB2
- Informix
- MySQL Enterprise
- Oracle
- PostgreSQL
- SQL Server
- Sybase

How to create a data connection definition for a database

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Data Connection Definition.
- 3 In New Data Connection Definition:
 - In Name, type the name of the database.
 - In Type, choose a type from the drop-down list.
 - In Description, type a description for the database.
 - To retrieve the connection property values from the data source connection configuration file at run time, type the configuration key in Configuration key.

Figure 2-5 shows an example of creating a data connection definition for a SQL Server database.

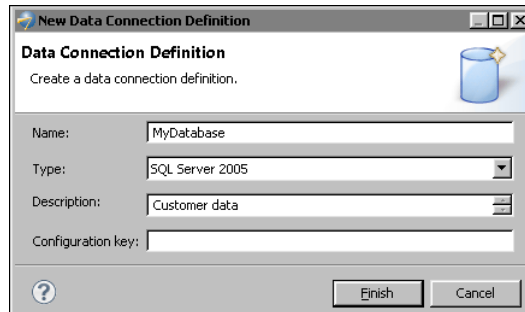


Figure 2-5 Creating a data connection definition for a database

- Choose Finish.


The IO Designs folder hierarchy and data connection definition file name appear in Navigator, and Data source connection properties appears. Connection property values stored in the data source connection configuration file are not displayed.

4 In Data source connection properties:

- If you did not provide a configuration key earlier, you may provide one now.
- In Credentials, choose Proxy or Passthrough.
If you choose Proxy, the IO Design perspective connects to the database using the user name and password you specify in Data source connection properties. If you choose Passthrough, the IO Design perspective connects to the database using the user name and password you specify in User Information. For information about using proxy and passthrough security in an Encyclopedia volume, see *Managing an Encyclopedia Volume*.
- If you chose Proxy, type the user name and password for the database user.
- In Port, type the number of the port that the IO Design perspective uses to connect to the database server.
- Provide values for the remaining properties.

The remaining properties are specific to the connection type.

Figure 2-6 shows an example of specifying the connection properties for a SQL Server database.



Data source connection properties

Type:

Description:

Configuration key:

Credentials:

User name:

Password:

Server:

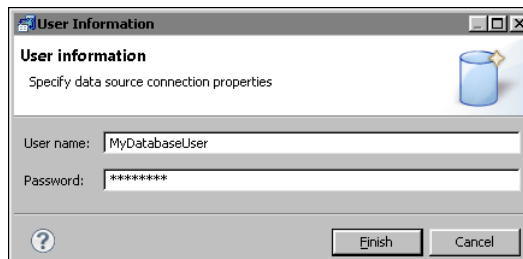
Database:

Port:

Schema (optional):

Figure 2-6 Specifying data source connection properties for a database data connection definition

- Choose **Test Connection** to test the connection to the database. If you chose **Passthrough** in **Credentials**, type the database user name and password in **User Information** and choose **Finish**, as shown in Figure 2-7.



User Information

User information

Specify data source connection properties

User name:

Password:

Figure 2-7 Providing user information to connect to the database

If the IO Design perspective connects to the database, a confirmation message appears. Choose **OK**.

- Choose one of the following:
 - ❑ **Map tables**, to create maps of database tables and views
 - ❑ **Map stored procedures**, to create a map of a stored procedure result set
 - ❑ **Create SQL map**, to create a map of a query written in the database's native SQL

Creating a data connection definition for an ODA data source

The IO Design perspective uses an Open Data Access (ODA) driver to connect to an ODA data source. The preconfigured ODA data source types include:

- e.Reports Data Connector for iHub
- BIRT document
- Flat file
- POJO
- Static
- Web services
- XML

How to create a data connection definition for an ODA data source

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Data Connection Definition.
- 3 In New Data Connection Definition:
 - In Name, type the name of the ODA data source.
 - In Type, choose a type from the drop-down list.
 - In Description, type a description for the ODA data source.
 - To retrieve the connection property values from the data source connection configuration file at run time, type the configuration key in Configuration key.

Figure 2-8 shows an example of creating an ODA data source connection definition.

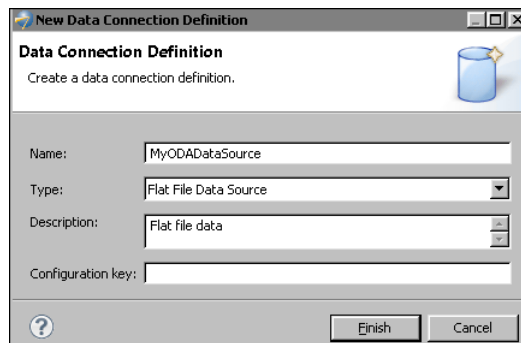


Figure 2-8 Creating an ODA data source connection definition

- Choose Finish.
The data connection definition file name appears in Navigator, and Data source connection properties appears.

4 In Data source connection properties:

- If you did not provide a configuration key earlier, you may provide one now.

Figure 2-9 shows an example of beginning to specify the connection properties for an ODA data source connection definition.

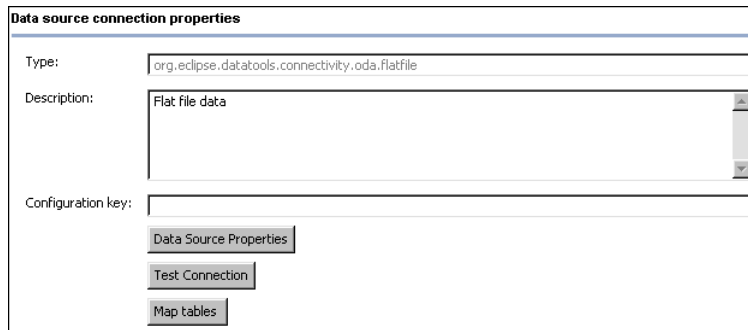


Figure 2-9 Specifying connection properties for an ODA data source connection definition

- Choose Data Source Properties.
- In the data source connection properties dialog:
 - Provide values for the data source connection properties.

Figure 2-10 shows an example of specifying the properties for a flat file data source.

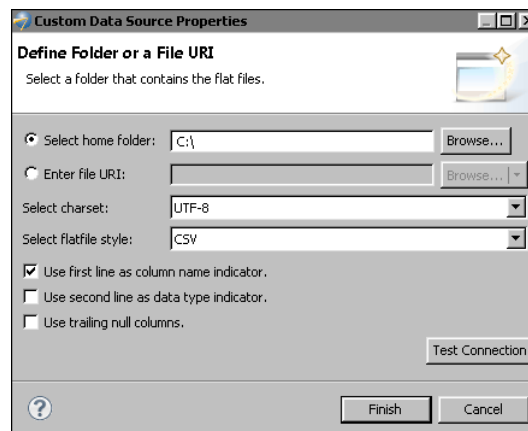


Figure 2-10 Specifying connection properties for a flat file data source

- ❑ Choose Test Connection to test the connection to the data source.
If the IO Design perspective connects to the data source, a confirmation message appears. Choose OK.
- ❑ Choose Finish.
- Choose Map tables to create a map of an ODA data source query result set.

About connection properties

Table 2-1 lists the preconfigured connection types, connection properties, and a description of each property.

Table 2-1 Properties for preconfigured connection types

Connection type	Property	Description
e.Reports Data Connector for iHub Data Source <i>See Using Actuate BIRT Designer Professional.</i>	Password	User's Encyclopedia volume password.
	Server URL	URL for the iServer that manages the Encyclopedia volume in which the ROI file resides, for example <code>http://MyServer:8000</code> .
	Use Trusted Connection	Choose Yes to use a trusted connection. As you edit the data source and the data set, a trusted connection uses the same session to communicate with the iServer. Using a trusted connection improves performance. Choose No to use a non-trusted connection. A non-trusted connection uses the specified credentials to log in to the iServer for each communication.
	User Volume	Encyclopedia volume user name. Name of the Encyclopedia volume in which the ROI file resides.

(continues)

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
BIRT Report Document Data Source <i>See Using Actuate BIRT Designer Professional.</i>	Report document path	Path to the report document to use as a data source. The path must be an absolute path to a location on the iHub computer.
DB2	Collection	Similar to schema. Used only with z/OS and OS/400 operating systems.
	Database	Name of database.
	IANAApp-CodePage	For a description of IANAAppCodePage values, see “About the IANAAppCodePage property,” later in this chapter.
	Password	A password used to connect to your DB2 database.
	Port	The port number that is assigned to the DB2 DRDA listener process on the server host machine. Specify this port’s numeric address or its name. If you specify a port name, the database driver must find this name in the SERVICES file on the iHub computer. Port is optional.
	Server	The IP address of the machine where the catalog tables are stored. Specify the address using the machine’s numeric address (for example, 123.456.78.90) or specify its name. If you specify a name, the database driver must find this name in the HOSTS file on the iHub computer or in a DNS server.
Flat File Data Source <i>See BIRT: A Field Guide.</i>	User name	The login ID used to connect to your DB2 database. For DB2 on Linux, the User name is your Linux user ID.
	Folder	The UNC path for the folder in which the file resides, for example \\MyComputer\MyFolder. The folder must reside on a Windows computer and must be shared.
	Charset	Character set used to encode the file.

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
Flat File Data Source (<i>continued</i>)	Flatfile style	CSV, SSV, PSV, or TSV for a file that uses comma-separated values, semicolon-separated values, pipe-separated values, or tab-separated values, respectively.
Informix	Database	Used to filter catalogs. For example, if you set this property to MyCatalog, the connection recognizes only those catalogs whose name begins with MyCatalog. If you want the connection to recognize all catalogs, do not provide a value for this property.
	Host	The name of the computer on which the Informix database resides.
	IANAAppCodePage	For a description of IANAAppCodePage values, see "About the IANAAppCodePage property," later in this chapter.
	Password	A password.
	Port	The port number of the server listener.
	Service	The name of the server running the Informix database.
	User name	Your user name as specified on the Informix server.
MySQL Enterprise	Database	Used to filter catalogs. For example, if you set this property to MyCatalog, the connection recognizes only those catalogs whose name begins with MyCatalog. If you want the connection to recognize all catalogs, do not provide a value for this property.
	Password	The password for the MySQL Enterprise login account specified by the User name property.
	Port	Port number (optional).

(continues)

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
MySQL Enterprise (continued)	Server	The network address of the server running MySQL Enterprise. This is required and can be an IP address, for example, 199.226.224.34. If your network supports named servers, you can specify an address using the server name, for example, SSserver. To specify a named instance of MySQL Enterprise, use the format server_name \instance_name. If only a server name is specified with no instance name, the database driver connects to the server and uses the default named instance on the server.
	User name	A valid MySQL Enterprise login account.
Oracle	Password	The password that the database driver uses to connect to your Oracle database.
	Port	Identifies the port number of your Oracle listener. The default value is 1521. Check with your database administrator for the correct number.
	Server	Identifies the Oracle server to which you want to connect. If your network supports named servers, you can specify a server name, such as Oracleserver. Otherwise, specify an IP address, such as 199.226.224.34.
	SID	The Oracle System Identifier that refers to the instance of the Oracle database software running on the server.
	TNS names file	Name of the TNS names file, for example, tnsnames.ora. This file must be accessible from the computer running the Integration service. Used only when the Server name property is not set.
	TNS server name	Name of the entry in the TNS names file that contains the configuration information describing the database server. Used only when the Server name property is not set.

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
Oracle (<i>continued</i>)	User name	The user name that the database driver uses to connect to the Oracle database.
POJO Data Source See <i>Using Actuate BIRT Designer Professional</i> .	Runtime properties	Shared location of custom POJO data set classes.
	Design time properties	Shared location of custom POJO data set classes.
PostgreSQL	Database	Used to filter catalogs. For example, if you set this property to MyCatalog, the connection recognizes only those catalogs whose name begins with MyCatalog. If you want the connection to recognize all catalogs, do not provide a value for this property.
	Password	The password for the PostgreSQL login account specified by the User name property.
	Port	Port number (optional).
	Server	The network address of the server running PostgreSQL. This is required and can be an IP address, for example, 199.226.224.34. If your network supports named servers, you can specify an address using the server name, for example, SSserver. To specify a named instance of PostgreSQL, use the format server_name \instance_name. If only a server name is specified with no instance name, the database driver connects to the server and uses the default named instance on the server.
	User name	A valid PostgreSQL login account.

(continues)

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
SQL Server	Database	Used to filter catalogs. For example, if you set this property to MyCatalog, the connection recognizes only those catalogs whose name begins with MyCatalog. If you want the connection to recognize all catalogs, do not provide a value for this property. If you are running multiple instances of SQL Server, precede the database name with the instance name, for example MyInstance/MyDatabase.
	Password	The password for the SQL Server login account specified by the User name property.
	Port	Port number (optional).
	Server	The network address of the server running SQL Server. This is required and can be an IP address, for example, 199.226.224.34. If the network supports named servers, you can specify an address using the server name, for example, SSserver. To specify a named instance of SQL Server, use the format server_name\instance_name. If only a server name is specified with no instance name, the database driver connects to the server and uses the default named instance on the server.
	User name	A valid SQL Server login account.
Static Data Source <i>See Using Actuate BIRT Designer Professional.</i>		Create a data set by typing values in the data set editor. Use a static data source to create sample data for testing purposes.
Sybase	Charset	The name of a character set. This character set must be installed on the Sybase server. The default is the setting on the Sybase server. For the Integration service to return Unicode data, this property must be set to UTF8. Refer to the Sybase server documentation for a list of valid character set names.

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
Sybase (<i>continued</i>)	Database	Used to filter catalogs. For example, if you set this property to MyCatalog, the connection recognizes only those catalogs whose name begins with MyCatalog. If you want the connection to recognize all catalogs, do not provide a value for this property.
	IANAApp-CodePage	For a description of IANAAppCodePage values, see "About the IANAAppCodePage property," later in this chapter.
	Password	A case-sensitive password.
	Port	Port number.
	Server	An IP address, for example, 199.226.224.34. If your network supports named servers, you can specify the address using the server name, for example, Sybaseserver.
	User name	The login ID used to connect to your Sybase database. This ID is case-sensitive.
Web Services Data Source See <i>BIRT: A Field Guide</i> .	WSDL descriptor	The path or URL for the Web Services Description Language file. A well-formed WSDL file defines the available services and, typically, the SOAP end point URL.
	SOAP end point	SOAP end point URL. Omit this value if the end point is defined in the WSDL file, or if you are using a custom connection class that does not require an end point URL.
	Custom driver class	The fully qualified class name. Create and use a custom driver if, for example, the web service does not provide a WSDL file.
	Driver class path	A semicolon-separated list of JAR files and directories to search for the custom driver class.
XML Data Source See <i>BIRT: A Field Guide</i> .	URL of the XML source	The path or URL for the file that contains XML data.

(continues)

Table 2-1 Properties for preconfigured connection types (continued)

Connection type	Property	Description
XML Data Source (continued)	URL of the XML schema	The path or URL for the file that contains a description of the XML file's data structure (optional).
	Encoding for the XML source and schema	Encoding for the XML file and schema. Use the default value, Auto, if you want the data source to detect the encoding for the XML file or schema.

About the IANAAppCodePage property

If the database character set is not Unicode and iHub is running on a Linux platform, you must set the IANAAppCodePage property. The value of IANAAppCodePage must match the database character encoding and the system locale. If you do not set IANAAppCodePage, the database driver searches for a value in the system information file (odbc.ini), first in the Data Source section and then in the ODBC section. If the database driver does not find a value in odbc.ini, it sets IANAAppCodePage to 4 (ISO 8859-1 Latin-1).

Table 2-2 lists values for IANAAppCodePage and the corresponding character encodings.

Table 2-2 IANAAppCodePage values for character encoding

IANAAppCodePage value	Character encoding
3	US_ASCII
4	ISO_8859_1
5	ISO_8859_2
6	ISO_8859_3
7	ISO_8859_4
8	ISO_8859_5
9	ISO_8859_6
10	ISO_8859_7
11	ISO_8859_8
12	ISO_8859_9
13	ISO_8859_10
16	JIS_Encoding
17	Shift_JIS
18	EUC_JP

Table 2-2 IANAAppCodePage values for character encoding (continued)

IANAAppCodePage value	Character encoding
30	ISO_646_IRV
36	KS_C_5601
37	ISO_2022_KR
38	EUC_KR
39	ISO_2022_JP
40	ISO_2022_JP_2
57	GB_2312_80
104	ISO_2022_CN
105	ISO_2022_CN_EXT
109	ISO_8859_13
110	ISO_8859_14
111	ISO_8859_15
113	GBK
2004	HP_ROMAN8
2009	IBM850
2010	IBM852
2011	IBM437
2013	IBM862
2024	Windows_932
2025	GB2312
2026	Big5
2027	macintosh
2028	IBM037
2030	IBM273
2033	IBM277
2034	IBM278
2035	IBM280
2037	IBM284
2038	IBM285
2039	IBM290

(continues)

Table 2-2 IANAAppCodePage values for character encoding (continued)

IANAAppCodePage value	Character encoding
2040	IBM297
2041	IBM420
2043	IBM424
2044	IBM500
2045	IBM851
2046	IBM855
2047	IBM857
2048	IBM860
2049	IBM861
2050	IBM863
2051	IBM864
2052	IBM865
2053	IBM868
2054	IBM869
2055	IBM870
2056	IBM871
2062	IBM918
2063	IBM1026
2084	KOI8_R
2085	HZ_GB_2312
2086	IBM866
2087	IBM775
2089	IBM00858
2091	IBM01140
2092	IBM01141
2093	IBM01142
2094	IBM01143
2095	IBM01144
2096	IBM01145
2097	IBM01146
2098	IBM01147

Table 2-2 IANAAppCodePage values for character encoding (continued)

IANAAppCodePage value	Character encoding
2099	IBM01148
2100	IBM01149
2102	IBM1047
2250	WINDOWS_1250
2251	WINDOWS_1251
2252	WINDOWS_1252
2253	WINDOWS_1253
2254	WINDOWS_1254
2255	WINDOWS_1255
2256	WINDOWS_1256
2257	WINDOWS_1257
2258	WINDOWS_1258
2259	TIS_620

About Informix database connections

Set the DELIMIDENT environment variable to y before starting iHub.

The Integration service cannot communicate with an Informix database if the database character set is Unicode.

If iHub is running on a Windows platform, the computer's application code page must match the DB_LOCALE of the Informix server. For example, if the database character encoding is Japanese (Shift JIS), the application code page of the computer on which iHub is running must be Japanese (Shift JIS). In other words, you must set CLIENT_LOCALE to ja_JP.sjis-s.

If iHub is running on a Linux platform, you must provide a value for the IANAAppCodePage property. If you do not set IANAAppCodePage, the database driver searches for a value in the system information file (odbc.ini), first in the Data Source section and then in the ODBC section. If the database driver does not find a value in odbc.ini, it sets IANAAppCodePage to 4 (ISO 8859-1 Latin-1).

Specifying a production database schema

You can provide the name of the production database schema in a data connection definition. Provide the name of the production database schema in the following cases:

- The development database and the production database are identical except for the schema name.
- You plan to use iHub administrator dashboard reports, so you must provide the schema name for the PostgreSQL database.

On the Data source connection properties page, in Schema (optional), type the schema name in the form MyCatalog.MySchema or browse for the schema. The schema name is applied to maps of database tables and views at runtime, overriding the schema name stored in the maps.

The schema name is not applied to maps of native SQL queries or to maps of stored procedure result sets. The schema name in these maps must be edited manually before you publish the project to the production volume.

Do not provide a schema name if the maps in your project use more than one database schema.

Encrypting and decrypting data source connection property values

You can encrypt and decrypt certain data source connection property values, for example password, using your own encryption and decryption algorithms. You implement encryption and decryption algorithms using an Eclipse-based OSGi extensions framework.

Understanding the encryption extension point plug-in

The encryption extension point plug-in is installed with the following products in the specified locations:

- BIRT Designer Professional in <BDPro_HOME>\eclipse\plugins\com.actuate.ais.encryption_<version>
- BIRT iHub in \$AC_SERVER_HOME/Jar/BIRT/platform/plugins/com.actuate.ais.encryption_<version>

The directory com.actuate.ais.encryption contains the following items:

- The file plugin.xml
- The file encryption.jar
- The directory schema, which contains the file EncryptionProviderID.exsd

To extend the encryption extension point plug-in, you must implement both the encrypt and decrypt methods in the IEncryptionProvider interface, shown in Listing 2-1.

Listing 2-1 The IEncryptionProvider interface

```
package com.actuate.ais.encryption;

/**
 * This interface specifies a couple of functions that need to
 * be implemented in any encryption provider implementation
 */
public interface IEncryptionProvider {

    /**
     * Encrypt function that takes in a string value to be
     * encrypted. The return value is an encrypted text obtained
     * after applying the implementation specific encryption
     * algorithm.
     *
     * @param value
     * @return
     */
    public String encrypt(String value);

    /**
     * Decrypt function that takes in an encrypted text string.
     * The return value is the plain text obtained after applying
     * the implementation decryption algorithm.
     *
     * @param value
     * @return
     */
    public String decrypt(String value);
}
```

The extension JAR file must be installed in the following locations:

- <BDPro_HOME>\eclipse\plugins in the BIRT Designer Professional installation
- \$AC_SERVER_HOME/Jar/BIRT/platform/plugins on the iHub platform

When the IO Design perspective is launched, it detects the encryption extension point plug-in. This plug-in is used for all connection types, for example Oracle and DB2. When the data modeler enters connection property values such as user name, password, host name, and port on the Data source connection properties page, the IO Design perspective determines if the property is tagged as masked. If so, the value entered for that property is passed to the encrypt method. The encrypt method returns the String value you programmed it to return, and this

return value is stored in the data connection definition (.dcd) file. The encrypt method is called only when the value of a masked property is modified. When an information object is executed in the IO Design perspective or on iHub, the values of the connection properties that are tagged as masked are read from the DCD file and passed to the decrypt method. The decrypt method returns the String value you programmed it to return.

You can have the encrypt method return an encrypted version of the string that a data modeler enters on the Data source connection properties page. This encrypted value is then stored in the DCD file and passed to the decrypt method when an information object is executed.

You can also program the encrypt and decrypt methods to implement lookup mechanisms to retrieve the actual property values, such as the user name and password, from an external LDAP source. The values that the data modeler enters on the Data source connection properties page serve as tokens to identify the actual values. This approach can handle multiple data sources.

For example, the encrypt method can simply return any string value the data modeler provides without modification, and this token is stored in the DCD file. So, if a data modeler enters the password for an Oracle connection definition as Password_OracleDevelopment, the encrypt method returns Password_OracleDevelopment, and Password_OracleDevelopment is stored in the DCD file. When the decrypt method receives Password_OracleDevelopment, the decrypt method logic uses this token to query an external data source or to search a local encrypted file to retrieve the actual password.

Extending the encryption extension point plug-in

To extend the encryption extension point plug-in, you must perform the following tasks:

- Make the encryption extension point plug-in available to Eclipse.
- Create a plug-in project.
- Include any required JAR files in the plug-in JAR file.
- Select an extension point.
- Create the plug-in class.
- Implement both the encrypt and decrypt methods in the IEncryptionProvider interface.
- Package the plug-in.
- Deploy the plug-in.
- Set the Type attribute for the appropriate connection parameters to masked.

Detailed instructions for each task are provided in the following topics.

How to make the encryption extension point plug-in available to Eclipse

Copy `com.actuate.ais.encryption_<version>` from one of the locations where it resides to `eclipse\plugins`.

How to create a plug-in project

- 1 In Eclipse, choose **File**→**New**→**Project**.
- 2 In **New Project**—Select a wizard, select **Plug-in Project**. Choose **Next**.
- 3 In **New Plug-in Project**—**Plug-in Project**, in **Project name**, type the name of the project, as shown in Figure 2-11. Choose **Next**.

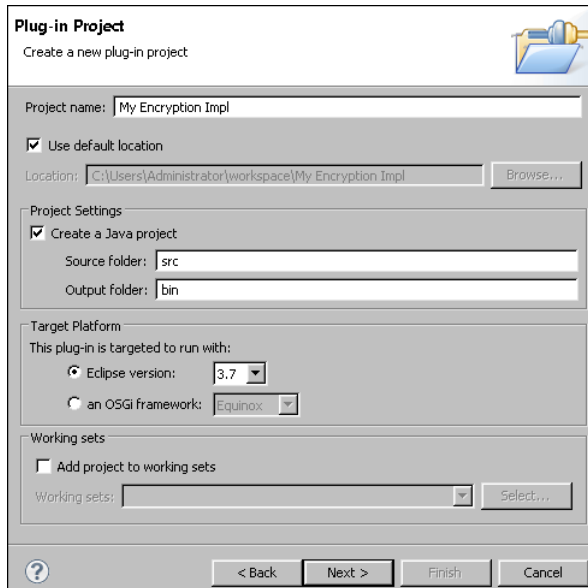


Figure 2-11 Typing the name of the project

- 4 In **New Plug-in Project**—**Content**, do the following, as shown in Figure 2-12. Then, choose **Finish**.
 - In **ID**, type the name of the identifier for the plug-in and the package name for the encryption provider class.
 - In **Version**, type the version number for the code.
 - In **Name** (optional), type a descriptive name for the plug-in.
 - In **Provider** (optional), type the name of the extension's provider.
 - Select **Generate an activator**, a Java class that controls the plug-in's life cycle.

- In Activator, type the full name of the Activator class. The package for the class should be the same as for ID.
- Deselect This plug-in will make contributions to the UI.

Figure 2-12 New Plug-in Project—Content

- 5 Confirm that you want to display the plug-in perspective.

How to include required JAR files in the plug-in JAR file

- 1 In the plug-in perspective, choose Dependencies.
- 2 In Required Plug-ins, choose Add.
- 3 In Plug-in Selection, select com.actuate.ais.encryption (version), as shown in Figure 2-13. Choose OK.

If com.actuate.ais.encryption is not visible, type encryption in Select a Plug-in to filter the list.

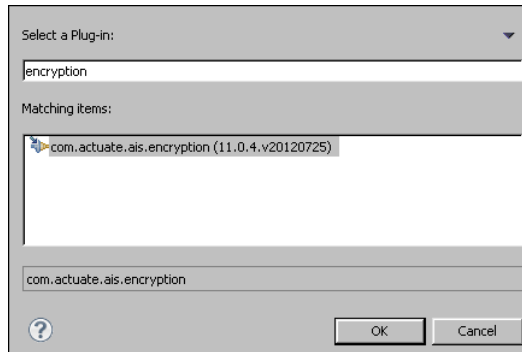


Figure 2-13 Specifying the required plug-in
com.actuate.ais.encryption appears in Required Plug-ins, as shown in Figure 2-14.

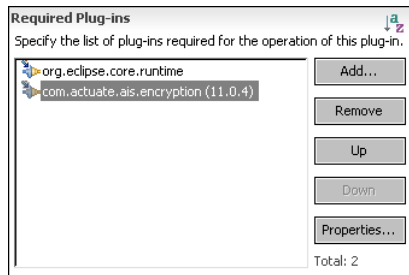


Figure 2-14 Required Plug-ins

How to select an extension point

- 1 Choose Extensions.
- 2 In All Extensions, choose Add.
- 3 In New Extension—Extension Point Selection—Extension Points, select com.actuate.ais.encryption.EncryptionProviderID, as shown in Figure 2-15. Choose Finish.

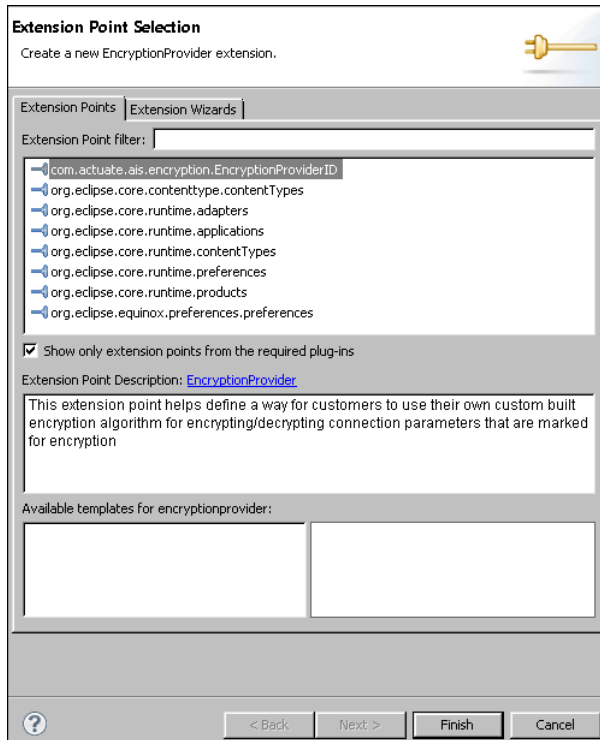


Figure 2-15 Selecting com.actuate.ais.encryption.EncryptionProviderID

- 4 In All Extensions, expand the com.actuate.ais.encryption.EncryptionProviderID node and select the child entry, as shown in Figure 2-16.

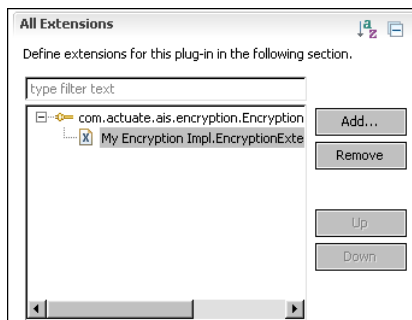
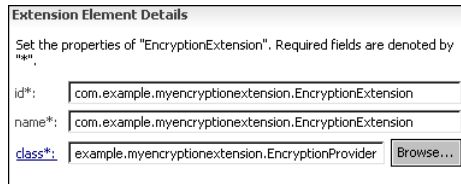


Figure 2-16 Selecting the child entry

- 5 In Extension Element Details, modify the id, name, and class as shown in Figure 2-17.

Note the name of the Java class, EncryptionProvider.



Extension Element Details

Set the properties of "EncryptionExtension". Required fields are denoted by tags.

id*:

name*:

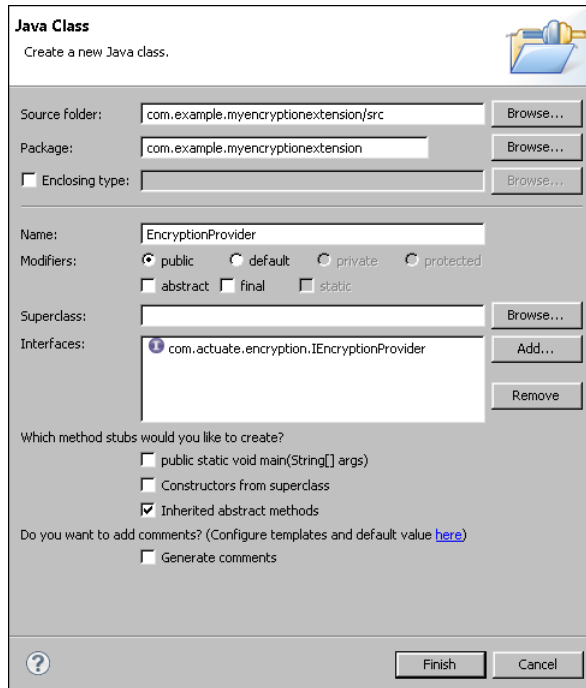
class*:

Figure 2-17 Setting the properties of the extension element

- 6 Choose File→Save to save the plugin.xml file.

How to create the plug-in class

- 1 In Extension Element Details, choose the underlined class link to create the plug-in class.
- 2 On Java Class, shown in Figure 2-18, check that the class name matches the name you provided in Extension Element Details. Choose Finish.



Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected

☐ abstract ☐ final ☐ static

Superclass:

Interfaces: ☒ com.actuate.encryption.IEncryptionProvider

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Figure 2-18 Checking the class name

Eclipse creates a stub class, shown in Figure 2-19, that contains the encrypt and decrypt methods.

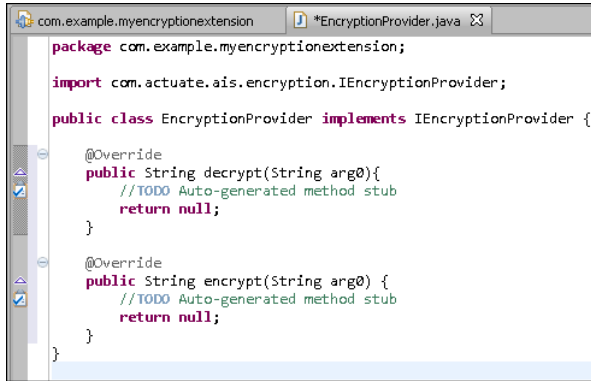


Figure 2-19 Stub class

How to implement the encrypt and decrypt methods

The code in Listing 2-2 implements AES 128-bit encryption. Test the encryption and decryption code in Eclipse. The Activator class, created by Eclipse, must exist in the plug-in.

Listing 2-2 Implementing the encrypt and decrypt methods

```

package com.example.myencryptionextension;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
import com.actuate.ais.encryption.IEncryptionProvider;

public class EncryptionProvider implements IEEncryptionProvider {

    private final byte[] key = {-0x6A, 0x6D, 0x49, -0x05, 0x79,
        0x38, 0x48, -0x0C, 0x6A, 0x19, 0x46, 0x1E, -0x09, -0x5E,
        -0x2F, 0x17};

    private Cipher getCipher(int mode) {
        Cipher cipher = null;
        SecretKeySpec keyspec = new SecretKeySpec(key, "AES");
        try {
            cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(mode, keyspec);
        } catch (Exception ex) {}
        return cipher;
    }

    @Override
    public String decrypt(String encryptedString) {

```



```

        String decryptedString = null;
        try {
            if (encryptedString == null) return null;
            if ("".equals(encryptedString)) return "";

            byte[] encryptedBytes = new
BASE64Decoder().decodeBuffer(encryptedString);
            Cipher cipher = getCipher(Cipher.DECRYPT_MODE);
            byte[] raw = cipher.doFinal(encryptedBytes);
            decryptedString = new String(raw);
        } catch (Exception ex) {}

        return decryptedString;
    }

    @Override
    public String encrypt(String plainText) {
        String encryptedString = null;

        try {
            if (plainText == null) return null;
            if ("".equals(plainText)) return "";

            Cipher cipher = getCipher(Cipher.ENCRYPT_MODE);
            byte[] raw = cipher.doFinal(plainText.getBytes());
            encryptedString = new BASE64Encoder().encode(raw);

        } catch (Exception ex) {}

        return encryptedString;
    }
}

```

How to package the plug-in

- 1 Open plugin.xml.
- 2 Choose Overview.
- 3 In Exporting, shown in Figure 2-20, choose Export Wizard.

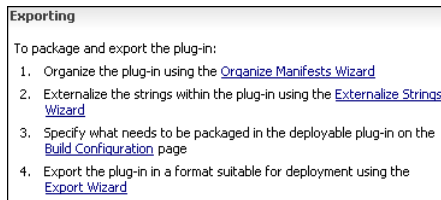


Figure 2-20 Choosing Export Wizard

- 4 In Destination, type or browse to the directory in which to save the plug-in library, as shown in Figure 2-21. Choose Finish. The plug-in is packaged in a JAR file in /plugins in the directory you specify.

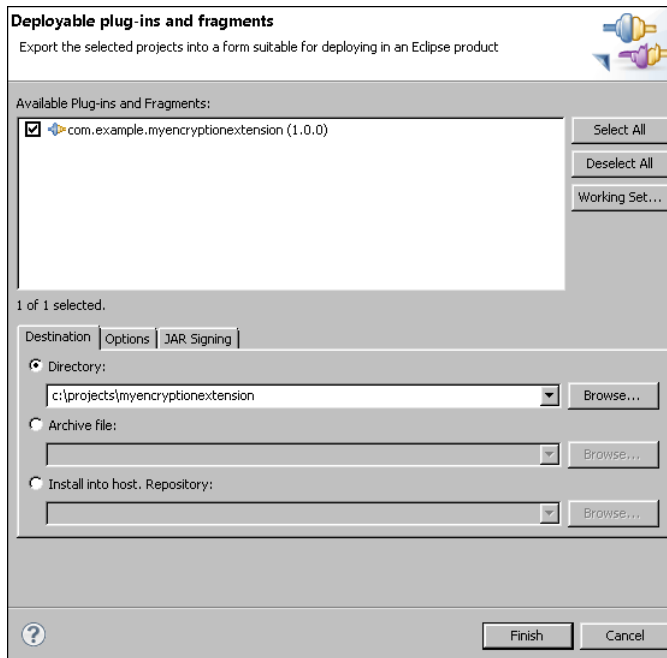


Figure 2-21 Specifying the directory in which to save the plug-in library

How to deploy the plug-in

- 1 Place the plug-in JAR file in \$AC_SERVER_HOME/Jar/BIRT/platform/plugins. If you are using an iHub cluster, repeat this step for each server in the cluster.
- 2 Place the plug-in JAR file in <BDPro_HOME>\eclipse\plugins.
- 3 Restart iHub and BIRT Designer Professional for the plug-in to take effect.

How to set the Type attribute to masked

Set the Type attribute for an encrypted connection property to masked in the appropriate ConnectionParam element in <BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\intsrsvsources.xml and \$AC_SERVER_HOME/etc/intsrsvsources.xml, or <BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\data_integration\datasources.xml and

\$AC_SERVER_HOME/etc/data_integration/datasources.xml. For example, in the following code, the Type attribute for the password property is set to masked:

```
<ConnectionParams>
  <ConnectionParam Name="username"
    Display="User name"
    Type="string">
  </ConnectionParam>
  <ConnectionParam Name="password"
    Display="Password"
    Type="masked">
  </ConnectionParam>
...
</ConnectionParams>
```

Troubleshooting an encryption extension

Plug-in loading errors are logged in the BIRT Designer Professional log file. To display the log file, choose Help→About Actuate BIRT Designer Professional→Installation Details→Configuration→View Error Log.

On the BIRT iHub platform, plug-in loading errors are logged in the following locations:

- On Windows platforms: \$AC_SERVER_HOME\Jar\BIRT\platform\configuration
- On Linux platforms: \$AC_SERVER_HOME/jar/BIRT/platform/configuration

Alternatively, decorate the Activator.start() and Activator.stop() methods with System.out statements to ensure that the plug-in is loaded. When the plug-in is loaded, debug statements are displayed.

Externalizing data source connection property values

You can externalize data source connection property values rather than embed them in the data connection definition (.dcd) file. If you externalize data source connection property values, you can move a project from one environment to another, for example, from a development environment to a test environment, without modifying the .dcd file. There are two ways to externalize data source connection property values:

- Passthrough security
To enable passthrough security:
 - Set the .dcd file's Credentials property to Passthrough.

- Using Management Console, set the data source connection property values.
For more information about setting data source connection property values using passthrough security, see *Managing an Encyclopedia Volume*.

- The data source connection configuration file

Data source connection property values specified using passthrough security take precedence over data source connection property values in the data source connection configuration file. ODA connections do not support passthrough security.

About the data source connection configuration file

You can use the iHub data source connection configuration file to externalize data source connection property values. This file is also used by BIRT report designs. A set of connection property values appears in a `ConnectOptions` element in the configuration file's `Runtime` element, for example:

```
<Runtime>
...
  <ConnectOptions Type="My_DB2_Connection">
    <Property PropName="server">My_DB2_Server</Property>
    <Property PropName="database">My_DB2_Database</Property>
    <Property PropName="username">My_DB2_User</Property>
    <Property PropName="password">My_DB2_Password</Property>
    <Property PropName="port">50000</Property>
    <Property PropName="appcodepage">3</Property>
  </ConnectOptions>
...
</Runtime>
```

The password is not encrypted.

You can create an entry for a new set of connection property values or add connection property values to an existing entry. For information about creating a `ConnectOptions` element, see one of the following topics:

- Externalizing connection property values for a preconfigured connection type
- Externalizing connection property values for a configurable connection type
- Externalizing connection property values for an ODA connection type

In each case, you must first locate the file in which the connection properties are specified.

The configuration key specified in the `.dcd` file must match the `Type` attribute for the `ConnectOptions` element. In Figure 2-22, the configuration key matches the `Type` attribute in the `ConnectOptions` element above.

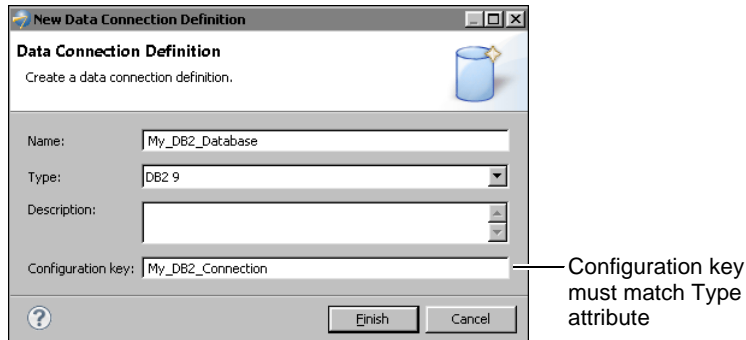


Figure 2-22 Specifying a configuration key

To locate the data source connection configuration file, check the setting of Configuration file for database connections using Configuration Console. For more information about this configuration variable, see *Configuring BIRT iHub*.

Externalizing connection property values for a preconfigured connection type

The connection properties for the following connection types are specified in the files `<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\intrsvrsources.xml` and `$AC_SERVER_HOME/etc/intrsvrsources.xml`:

- DB2
- Informix
- MySQL Enterprise
- Oracle
- PostgreSQL
- SQL Server
- Sybase

For example, the connection properties for the DB2 connection type are:

- server
- database
- username
- password
- port
- appcodepage

The connection properties are specified as follows:

```
<ConnectionType Name="DB2">
...
  <ConnectionParams>
    <ConnectionParam Name="server"
      Display="Server"
      Type="string"
      ValueIsCaseSensitive="false" />
    <ConnectionParam Name="database"
      Display="Database" Type="string"
      ValueIsCaseSensitive="false" />
    <ConnectionParam Name="username"
      Display="User name"
      Type="string" />
    <ConnectionParam Name="password"
      Display="Password"
      Type="masked" />
    <ConnectionParam Name="port"
      Display="Port"
      Type="integer"
      Optional="true"
      DefaultValue="50000" />
    <ConnectionParam Name="appcodepage"
      Display="IANAAppCodePage"
      Type="integer"
      Optional="true"
      DefaultValue="" />
  </ConnectionParams>
</ConnectionType>
```

To externalize the connection property values for a DB2 connection, add a `ConnectOptions` element to the data source connection configuration file's `Runtime` element, for example:

```
<Runtime>
...
  <ConnectOptions Type="My_DB2_Connection">
    <Property PropName="server">My_DB2_Server</Property>
    <Property PropName="database">My_DB2_Database</Property>
    <Property PropName="username">My_DB2_User</Property>
    <Property PropName="password">My_DB2_Password</Property>
    <Property PropName="port">50000</Property>
    <Property PropName="appcodepage">3</Property>
  </ConnectOptions>
...
</Runtime>
```

The property names listed in the `ConnectOptions` element must match the connection parameter names listed in the `ConnectionParams` element in `intsrvrresources.xml`. The match is case-sensitive.

Externalizing connection property values for a configurable connection type

In addition to the preconfigured connection types in `intsrvrresources.xml`, you can specify the connection properties for other connection types in the files `<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\data_integration\datasources.xml` and `$AC_SERVER_HOME/etc/data_integration/datasources.xml`, for example:

```
<ConnectionType Name="My_Database">
  <ConnectionParams>
    <ConnectionParam Name="database"
      Display="Database"
      Type="string"
      ValueIsCaseSensitive="false" />
    <ConnectionParam Name="username"
      Display="User name"
      Type="string" />
    <ConnectionParam Name="password"
      Display="Password"
      Type="masked" />
  </ConnectionParams>
</ConnectionType>
```

To externalize the connection property values for a configurable connection type, add a `ConnectOptions` element to the data source connection configuration file's `Runtime` element, for example:

```
<Runtime>
...
  <ConnectOptions Type="My_Database_Connection">
    <Property PropName="database">My_Database</Property>
    <Property PropName="username">My_Database_User</Property>
    <Property PropName="password">My_Database_Password</Property>
  </ConnectOptions>
...
</Runtime>
```

The property names listed in the `ConnectOptions` element must match the connection parameter names listed in the `ConnectionParams` element in `datasources.xml`. The match is case-sensitive.

Externalizing connection property values for an ODA connection type

The connection properties for an ODA connection type are specified in the plugin.xml file for the appropriate Eclipse plug-in. For example, the connection properties for the XML connection type are specified in the plugin.xml file bundled in org.eclipse.birt.report.data.oda.xml_xxxx.jar. This JAR file is located in <Actuate_HOME>/Jar/BIRT/platform/plugins. To extract plugin.xml, type:

```
jar xvf org.eclipse.birt.report.data.oda.xml_xxxx.jar plugin.xml
```

The JDK must be in your path.

The connection properties for the XML connection type, FILELIST and SCHEMAFILELIST, are specified as follows:

```
<properties>
  <propertyGroup
    defaultDisplayName="Connection Properties"
    name="connectionProperties">
    <property
      type="string"
      defaultDisplayName="%datasource.property.xmlFile"
      canInherit="true"
      name="FILELIST"/>
    <property
      type="string"
      defaultDisplayName="%datasource.property.schemaFile"
      canInherit="true"
      name="SCHEMAFILELIST"/>
    </propertyGroup>
  </properties>
```

To externalize the connection property values for an XML connection, add a ConnectOptions element to the data source connection configuration file's Runtime element, for example:

```
<Runtime>
...
  <ConnectOptions Type="My_XML_Connection">
    <Property PropName="FILELIST">My_XML File</Property>
    <Property PropName="SCHEMAFILELIST">My_XML Schema</Property>
  </ConnectOptions>
...
</Runtime>
```

The property names listed in the ConnectOptions element must match the property names listed in the properties element in plugin.xml. The match is case-sensitive.

Creating maps

A map represents one of the following:

- A database table
- A database view
- A query written in the database's native SQL
- A result set from a stored procedure
- A result set from an ODA data source query

A map file name has an .sma extension. Map file names are not case-sensitive.

You can create column categories for maps of database tables and views. For other map types, build an information object from the map using the graphical information object editor and create column categories for the information object.

Creating a map of a database table or view

When you create a map of a database table or view, the IO Design perspective places the map file in the same folder as the data connection definition (.dcd) file for the database. For example, Figure 2-23 shows that if you create a map of the table MyTable in the database MyDatabase, the IO Design perspective places the file MyTable.sma in the folder MyDatabase.

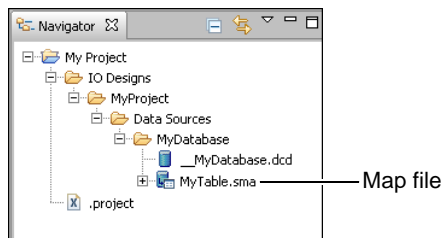


Figure 2-23 Location of a map file for a database table or view

How to create a map of a database table or view

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Map.
- 3 In New Maps—Data Source:
 - Select the appropriate data source.
 - Select Create maps by selecting database tables, as shown in Figure 2-24.

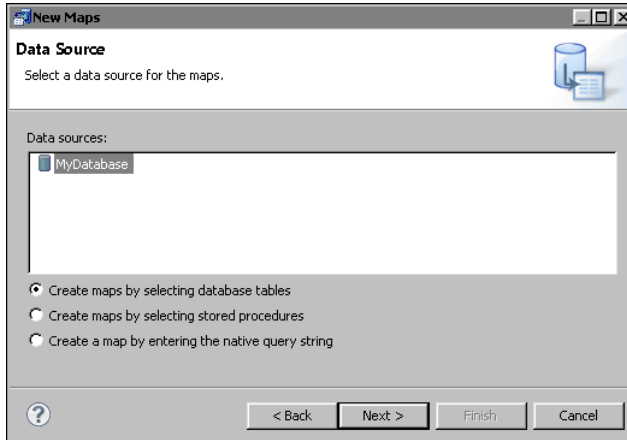


Figure 2-24 Creating maps by selecting database tables

Choose Next.

4 In New Maps—Maps:

- In Catalog, select the appropriate catalog.
- In Filter:
 - To display tables and views from a particular schema, type the first few characters of the schema name in Schema name prefix, for example, dbo. Do not append an asterisk, for example, dbo*. This filter is case-sensitive.
 - To display only tables and views whose names begin with a particular string, type the string in Table/View name prefix, for example, ac. Do not append an asterisk, for example, ac*. This filter is case-sensitive.
 - Select Show tables only, Show views only, or Show all.
 - Choose Apply filter.
- Move the appropriate tables and views from Available to Selected, as shown in Figure 2-25.

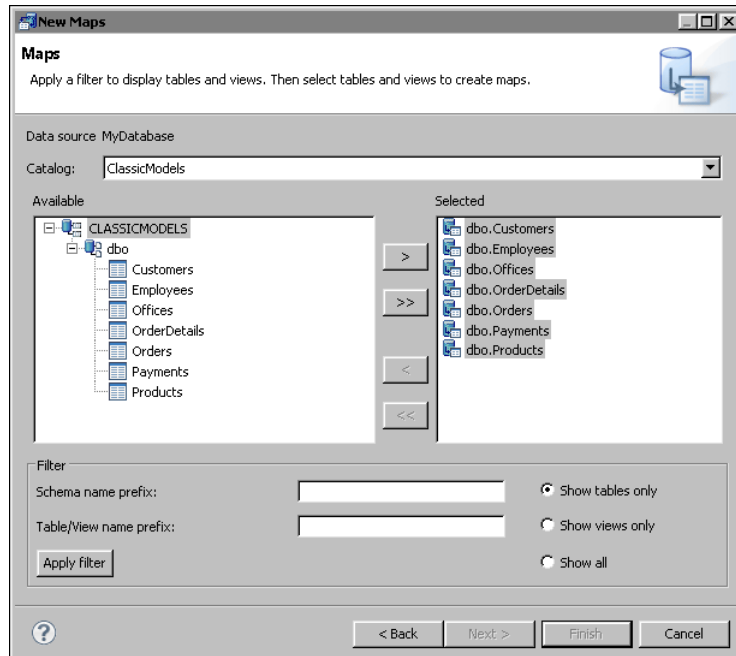


Figure 2-25 Selecting tables

Choose Finish.

A reminder appears. Choose OK.

The map file names appear in Navigator. If you created only one map, Output Columns appears.

- 5 If you created more than one map, double-click the first map file name in Navigator to display Output Columns.
- 6 In Output Columns:
 - Deselect the columns you want to exclude from the map.
 - To rename a column, type the new name in Name.
Decide on column names before you build an information object from the map. Changing a column name after you build an information object results in a compiler error in the information object.
 - To create a filter on a column, set the column's Filter property to Predefined and choose Prompt editor to specify the filter's prompt properties.
Figure 2-26 shows Output Columns and the location of the Prompt editor button.

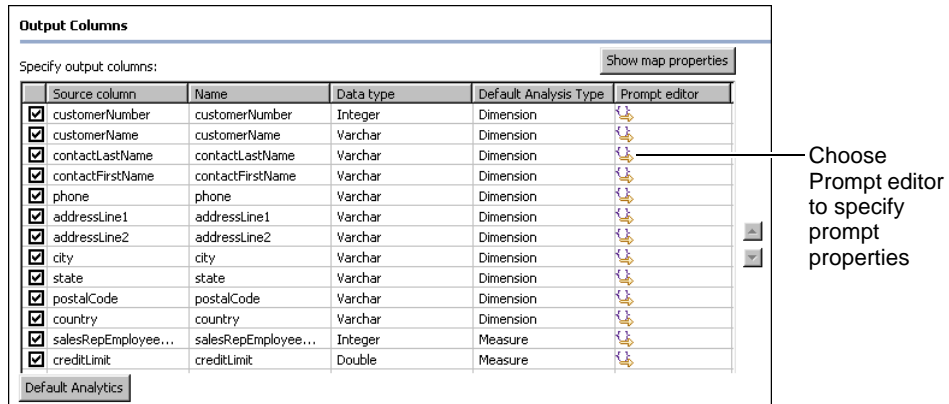


Figure 2-26 Map columns

To define other column properties, select the column and define the properties in Properties.

To specify the default analysis type for a column, choose Default Analytics.

To create column categories, choose Column Categories.

- To change the order of the columns, use Move up and Move down.

7 Repeat the previous two steps for the remaining maps, if any.

Updating a map of a database table or view

In a database, a table or view can change in the following ways:

- A column is renamed.
- A column's data type changes.
- A column is added to the table or view.
- A column is dropped from the table or view.

The IO Design perspective can detect these changes and update the map of the table or view. You can update a single map, or update several maps at once.

Map column name and data type changes are propagated to dependent information objects if the information object column uses the default name. If the name change would result in a duplicate information object column name, a suffix is added, for example creditLimit_1. A column name is updated in the following tabs in the graphical information object editor if it does not appear in an expression:

- Columns
- Column Categories

- Joins
- Filters
- Group By

A column name in an expression, for example SUM(quantityOrdered * priceEach), is not updated. If the information object column name has been modified, changes are not propagated.

A column that is added to a map is also added to dependent information objects as a source column. The column is not added to the information object's SELECT clause. In other words, the column appears in the upper pane of the graphical information object editor, but it is unchecked.

A column that is dropped from a map is not dropped from dependent information objects by default. You must indicate that the column should be dropped. Because this action cannot be undone, back up the project before proceeding.

How to update a map

- 1 In Navigator, right-click the map file name and choose Information Objects➤Update Map.
- 2 In Database Changes:
 - To display the data type for a dropped or added column, hover the cursor over the column name in Dropped Columns or Added Columns.
 - For renamed columns, drag the old column name from Dropped Columns to Old Column Name. Then drag the new column name from Added Columns to New Column Name. For columns whose data type has changed, the Old Column Name may be the same as the New Column Name. To remove a renamed column pair, choose Remove.
In Figure 2-27, the name of the customerID column has changed to customerNumber. The data type of the creditLimit column has changed from Decimal to Double.
 - To remove dropped columns from dependent information objects, select Propagate dropped column changes to dependent files. If the warning shown in Figure 2-28 appears, choose OK.

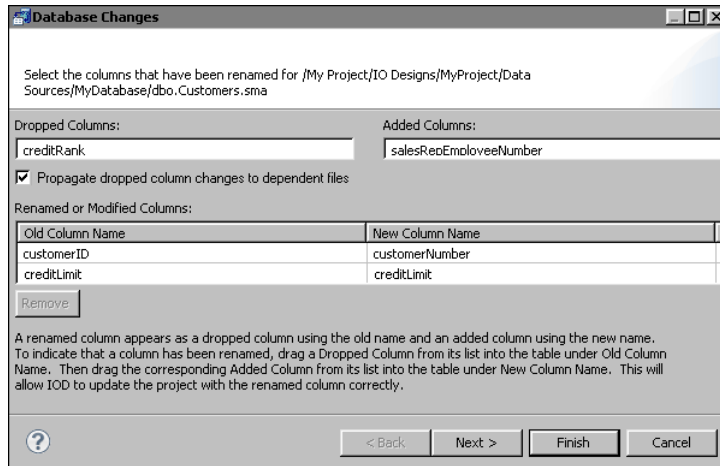


Figure 2-27 Map column changes

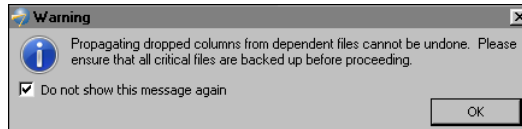


Figure 2-28 Dropped column propagation warning

- Choose Next.
- 3 Because these changes cannot be undone, review them carefully. If the changes are acceptable, choose Finish. If not, choose Back and make the necessary corrections. Figure 2-29 shows a summary of changes for the Customers map and its dependent information object, MyInformationObject.

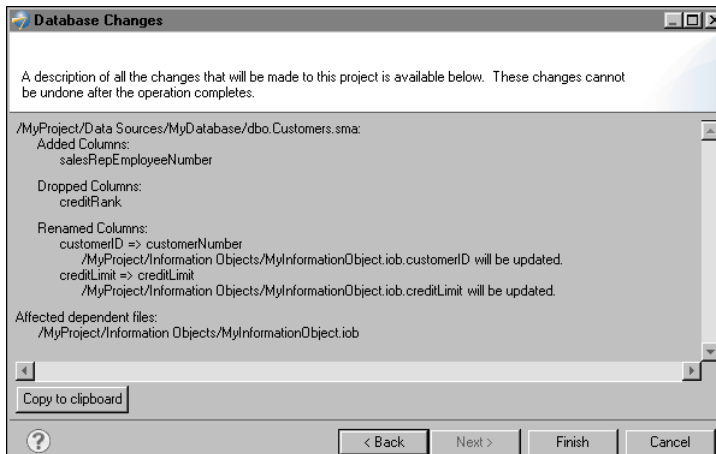


Figure 2-29 Summary of changes for Customers and MyInformationObject

How to update several maps at once

- 1 In Navigator, right-click the folder that contains the maps or any of its parent folders and choose Information Objects>Update Maps.
- 2 In Database Changes, select the maps to update, as shown in Figure 2-30. To select all maps in a folder, select the folder. To have the IO Design perspective automatically detect which maps need updating, choose Auto Detect. The Auto Detect operation may take a few moments. Choose Next.

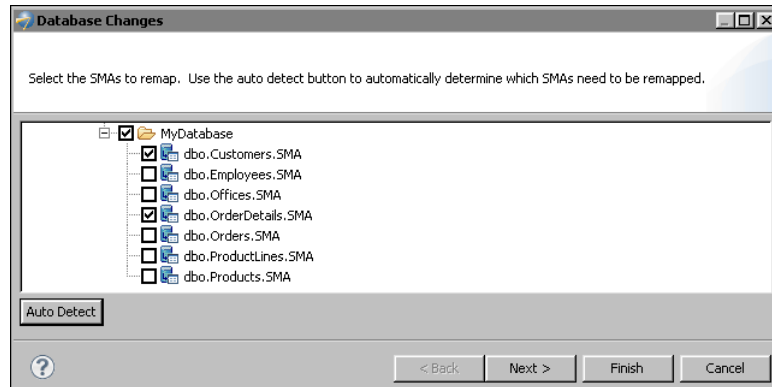


Figure 2-30 Selecting maps to update

- 3 For each map that needs updating, make the necessary changes and choose Next.
- 4 Review your changes carefully. If they are acceptable, choose Finish.

Creating a map of a native SQL query

You can map a query written in the database's native SQL. When writing the query, observe the following rules:

- Write the query using only the database's native SQL; do not use Actuate SQL functions or syntax.
- Do not include an ORDER BY clause in the query. Including an ORDER BY clause adversely affects the performance of information objects built from the map.
- Use unnamed parameters. An unnamed parameter is represented by a question mark (?). Do not use named parameters, for example :BeginDate.

If the query uses a parameter, it may be necessary to cast the parameter to the appropriate data type. For example, if the query queries an Oracle database, you must use the CAST or RPAD functions to ensure that the data type and length for a string parameter match the data type and length for the corresponding column

if the column is of type CHAR or NCHAR. For example, in the following queries the category column is of type CHAR(12):

```
SELECT orderID FROM items WHERE category = CAST ( ? AS  
CHAR (12) )
```

```
SELECT orderID FROM items WHERE category = RPAD ( ?, 12 )
```

How to create a map of a native SQL query

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Map.
- 3 In New Maps—Data Source:
 - Select the appropriate data source.
 - Select Create a map by entering the native query string, as shown in Figure 2-31.

Choose Next.

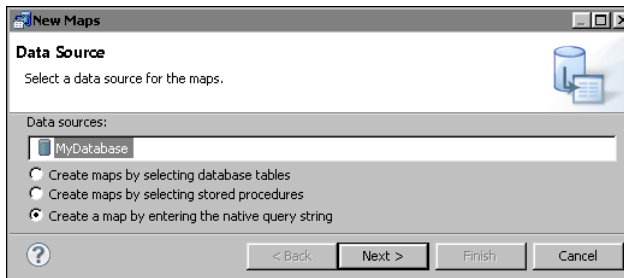


Figure 2-31 Creating a map by entering a native query string

- 4 In New Maps—Map, in Map name, type a map name, as shown in Figure 2-32. Choose Finish.

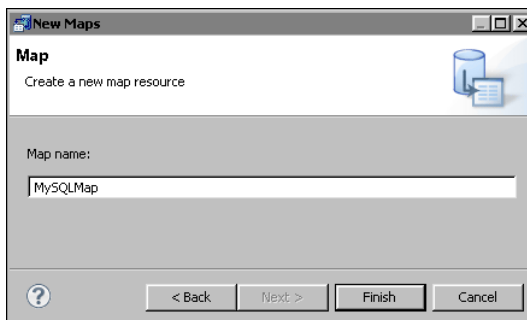
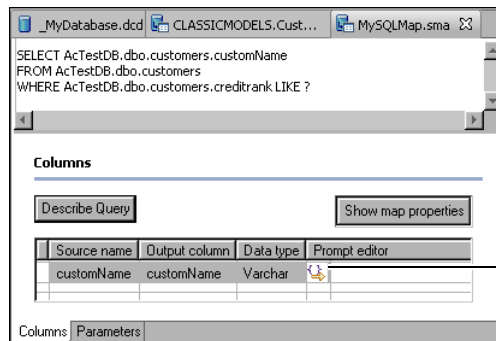


Figure 2-32 Specifying a map name

The map file name appears in Navigator.

5 In the textual query editor:

- In the upper pane, type or paste the native SQL query.
- In the lower pane, choose Describe Query.
The query's output columns appear.
- To rename a column, type the new name in Output column.
Decide on column names before you build an information object from this map. Changing a column name after you build an information object results in a compiler error in the information object.
- If necessary, choose the correct data type from the Data type drop-down list.
The Actuate SQL data type must be compatible with the native SQL data type.
- To create a filter on a column, set the column's Filter property to Predefined and specify the filter's prompt properties. Figure 2-33 shows Columns and the location of the Prompt editor button.
To define other column properties, select the column and define the properties in Properties.



Choose Prompt editor to specify prompt properties

Figure 2-33 A native SQL query and corresponding Columns page

- Choose Parameters.
The query's parameters appear.
- To rename a parameter, type the new name in Parameter.
The IO Design perspective assigns a default name to a parameter based on its position in the query, for example, param_1.
- If necessary, choose the correct data type from the Data type drop-down list.
The Actuate SQL data type must be compatible with the native SQL data type.

- In Default value, type the parameter's default value.
Do not create an expression.
- To specify the parameter's prompt properties, choose Prompt editor.
Figure 2-34 shows Parameters and the location of the Prompt editor button.
To define other parameter properties, select the parameter and define the properties in Properties.

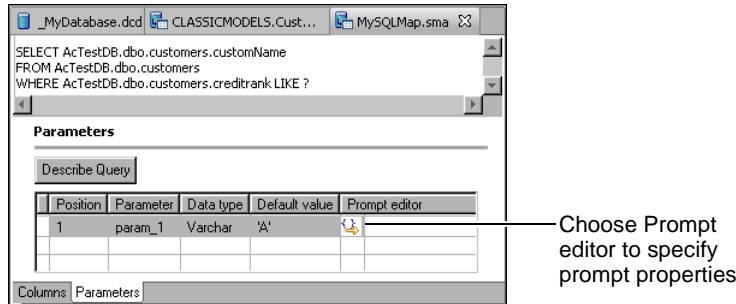


Figure 2-34 A native SQL query and corresponding Parameters page

Creating a map of a stored procedure result set

When you create a map of a stored procedure result set, the IO Design perspective creates a subfolder in the folder that contains the data connection definition (.dcd) file for the database. The subfolder contains an .epf file as well as the .sma file. The .epf file specifies:

- The statement that calls the stored procedure
- The stored procedure's input and input/output parameter values and data types
- The stored procedure's output parameters

The stored procedure's input and input/output parameters are associated with the result set map. In other words, when you build an information object from the result set map, these parameters are source parameters. The parameter values provided by a result set map user must yield the same result set metadata as the parameter values you provide when you create the map. In other words, the result set map must have the same columns and data types at run time as it does at design time.

The names of the subfolder and the .epf file are derived from the name of the stored procedure. For example, you are working with a stored procedure called MyStoredProcedure in a database called MyDatabase. As shown in Figure 2-35, if you create a map of a result set called MyResultSet, the IO Design perspective places the files MyResultSet.sma and _MyStoredProcedure.epf in a subfolder of MyDatabase called MyStoredProcedure.

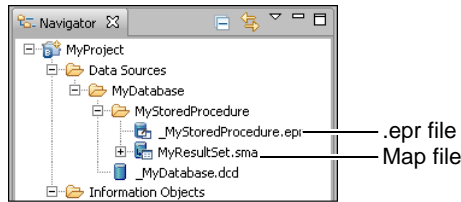


Figure 2-35 Location of the .epr file and map file for a stored procedure

If you create another result set map for MyStoredProcedure, the IO Design perspective creates a subfolder called MyStoredProcedure_1 and an .epr file called _MyStoredProcedure_1.epr. If you create a third result set map for MyStoredProcedure, the IO Design perspective creates a subfolder called MyStoredProcedure_2 and an .epr file called _MyStoredProcedure_2.epr, and so on.

The IO Design perspective does not distinguish between stored procedures that have the same name but different parameters (overloaded stored procedures). DB2 and Informix databases support overloaded stored procedures. To work with overloaded stored procedures, rename each stored procedure so that it has a unique name. If you cannot rename a stored procedure, create another stored procedure with a unique name that calls this stored procedure.

If you are working with a stored procedure in a Sybase database that has an output parameter, change the output parameter to an input/output parameter and provide a dummy value in Parameters For Stored Procedure.

How to create a map of a stored procedure result set

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Map.
- 3 In New Maps—Data Source:
 - Select the appropriate data source.
 - Select Create maps by selecting stored procedures, as shown in Figure 2-36.

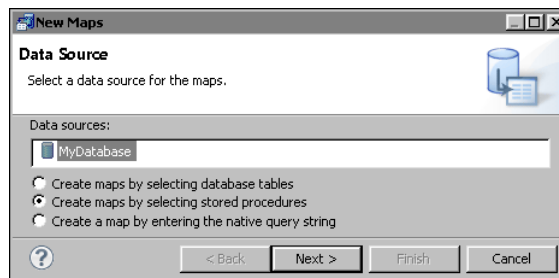


Figure 2-36 Creating a map by selecting a stored procedure

Choose Next.

4 In New Maps—Maps:

- In Catalog, select the appropriate catalog.
- In Filter:
 - To display stored procedures from a particular schema, type the first few characters of the schema name in Schema name prefix, for example, dbo. Do not append an asterisk, for example, dbo*. This filter is case-sensitive.
 - To display only stored procedures whose names begin with a particular string, type the string in Stored procedure name prefix, for example, ac. Do not append an asterisk, for example, ac*. This filter is case-sensitive.
 - Choose Apply filter.
- Move the appropriate stored procedure name from Available to Selected. Parameters For Stored Procedure appears.
- In Parameters For Stored Procedure:
 - In Statement, shown in Figure 2-37, correct the syntax if it is incorrect. If you are using a configurable database type, check your JDBC driver documentation for the correct syntax.

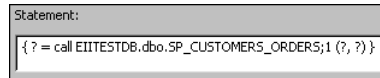


Figure 2-37 Correcting the syntax

- In Parameters, for the stored procedure's input and input/output parameters:
 - If necessary, choose the correct data type for each parameter from the Data type drop-down list. The Actuate SQL data type must be compatible with the native SQL data type.
 - If necessary, choose the correct parameter mode for each parameter from the Parameter mode drop-down list.
 - Type the values, as shown in Figure 2-38.

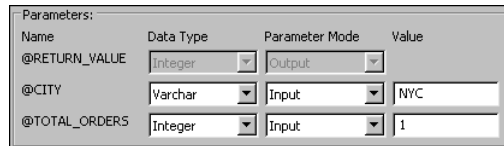


Figure 2-38 Specifying parameter values for a stored procedure

Choose OK. The name of the stored procedure appears in Selected, as shown in Figure 2-39.

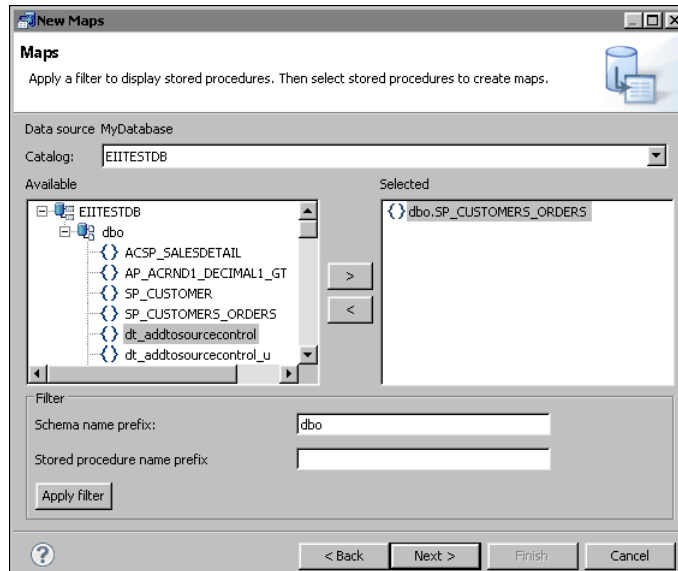


Figure 2-39 Result of specifying input parameters for a stored procedure

Choose Next.

5 In New Maps—Maps:

- Move the appropriate result set from Available to Selected. Result Set Name appears.
- In Name, type the name of the result set, as shown in Figure 2-40.

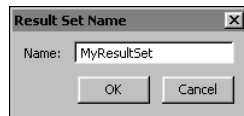


Figure 2-40 Naming the stored procedure's result set

Choose OK. The result set name appears in Selected. The result set columns appear in Data column preview, as shown in Figure 2-41.

Choose Finish. A reminder appears. Choose OK.

The .epr and .sma file names appear in Navigator.

6 In Output Columns:

- To rename a column, type the new name in Name. Decide on column names before you build an information object from this map. Changing a column name after you build an information object results in a compiler error in the information object.

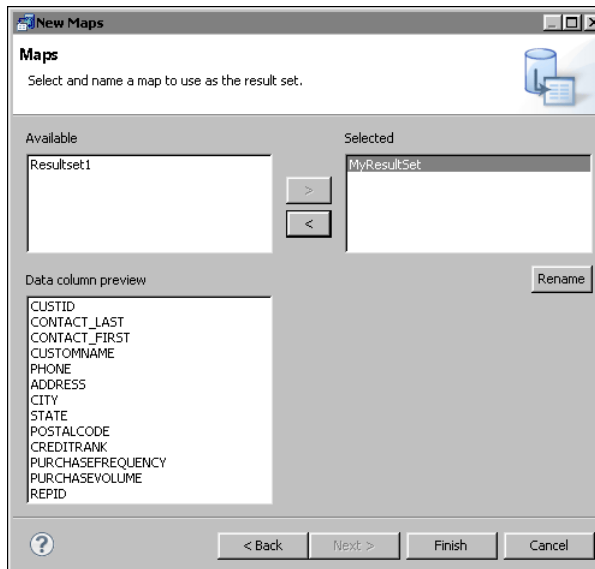


Figure 2-41 Viewing the list of result set columns

- If necessary, choose the correct data type from the Data type drop-down list. The Actuate SQL data type must be compatible with the native SQL data type.
- To create a filter on a column, set the column's Filter property to Predefined and choose Prompt editor to specify the filter's prompt properties. Figure 2-42 shows Output Columns and the location of the Prompt editor button.

To define other column properties, select the column and define the properties in Properties.

Output Columns					
Specify output columns:					Show map properties
	Source column	Name	Data type	Default Analysis T...	Prompt editor
<input checked="" type="checkbox"/>	customerNumber	customerNumber	Integer	Dimension	
<input checked="" type="checkbox"/>	customerName	customerName	Varchar	Dimension	
<input checked="" type="checkbox"/>	contactLastName	contactLastName	Varchar	Dimension	
<input checked="" type="checkbox"/>	contactFirstName	contactFirstName	Varchar	Dimension	
<input checked="" type="checkbox"/>	phone	phone	Varchar	Dimension	
<input checked="" type="checkbox"/>	addressLine1	addressLine1	Varchar	Dimension	
<input checked="" type="checkbox"/>	addressLine2	addressLine2	Varchar	Dimension	
<input checked="" type="checkbox"/>	city	city	Varchar	Dimension	
<input checked="" type="checkbox"/>	state	state	Varchar	Dimension	
<input checked="" type="checkbox"/>	postalCode	postalCode	Varchar	Dimension	
<input checked="" type="checkbox"/>	country	country	Varchar	Dimension	
<input checked="" type="checkbox"/>	salesRepEmployee...	salesRepEmployee...	Integer	Measure	
<input checked="" type="checkbox"/>	creditLimit	creditLimit	Double	Measure	

Choose Prompt editor to specify prompt properties

Figure 2-42 Columns created by the selected result set of a stored procedure

How to modify an .epf file for a stored procedure

- 1 In Navigator, double-click the appropriate .epf file.
- 2 In General:
 - In Query text, correct the syntax if it is incorrect.
If you are using a configurable database type, check your JDBC driver documentation for the correct syntax.
 - In Description, type a description for the stored procedure, as shown in Figure 2-43.

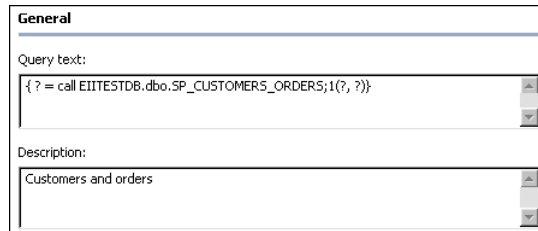




Figure 2-43 Providing a description of a stored procedure

- 3 Choose Parameters.
The stored procedure's parameters appear.
- 4 In Parameters:
 - To rename a parameter, type the new name in Parameter.
The IO Design perspective assigns a default name to a parameter based on its position in the statement, for example param_1.
 - If necessary, choose the correct Actuate SQL data type from the Data type drop-down list. The Actuate SQL data type must be compatible with the native SQL data type.
 - In Default value, type the parameter's default value.
Do not create an expression. You cannot type a default value for an output parameter.
 - To specify the parameter's prompt properties, choose Prompt editor.
Figure 2-44 shows the Parameters pane and the location of the Prompt editor button.

Parameters			
Parameter	Data type	Default value	Prompt editor
param_0	Integer		
param_1	Varchar	'NYC'	
param_2	Integer	1	

Choose Prompt editor to specify prompt properties

Figure 2-44 Specifying prompt properties for a stored procedure's parameters

To define other parameter properties, select the parameter and define the properties in Properties.

Creating a map of an ODA data source query result set

When you create a map of an ODA data source query result set, the IO Design perspective creates a subfolder in the folder that contains the data connection definition (.dcd) file for the ODA data source. The subfolder contains an .epr file as well as the .sma file. The .epr file specifies:

- The ODA data source query
- The query's input and input/output parameter values and data types
- The query's output parameters

The result set map represents the first result set returned by the ODA data source query. The query's input and input/output parameters are associated with the result set map. In other words, when you build an information object from the result set map, these parameters are source parameters. The parameter values provided by a result set map user must yield the same result set metadata as the parameter values you provide when you create the map. In other words, the result set map must have the same columns and data types at run time as it does at design time.

The names of the subfolder and the .epr file are derived from the name you provide for the query. For example, you are working with an ODA data source called MyODADataSource. As shown in Figure 2-45, if you create a map of a query result set called MyResultSet, the IO Design perspective places the files MyResultSet.sma and _MyODADataSourceQuery.epr in a subfolder of MyODADataSource called MyODADataSourceQuery.

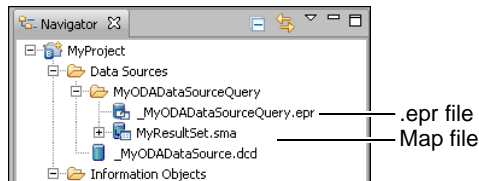


Figure 2-45 Location of the .epr file and map file for an ODA data source

How to create a map of an ODA data source query result set

- 1 In Data source connection properties, choose Map tables.
- 2 In New Maps—Maps:
 - In Name, type a name for the ODA data source query.
 - In Type, select an ODA data source type from the drop-down list, as shown in Figure 2-46.

Choose Next. The ODA data source query builder appears. For example, for a flat file data source, Select Table—Select Columns appears.

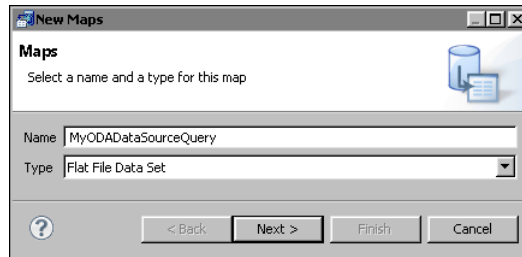


Figure 2-46 Selecting an ODA data source type

- 3 In the ODA data source query builder, build a query. The procedure for building a query varies by data source type. For a flat file data source, you select columns as shown in Figure 2-47. Choose Finish.

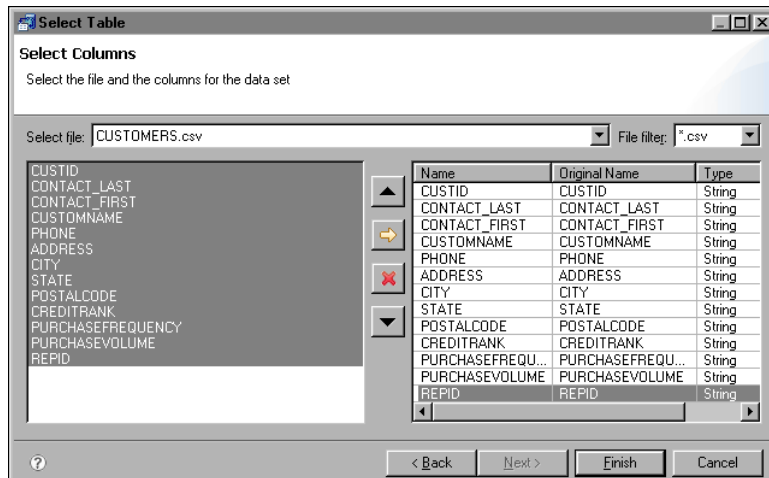


Figure 2-47 Selecting columns from a flat file data source

A reminder appears. Choose OK.

The .epr and .sma file names appear in Navigator.

4 In Output Columns:

- To rename a column, type the new name in Name.
Decide on column names before you build an information object from this map. Changing a column name after you build an information object results in a compiler error in the information object.
- To create a filter on a column, set the column's Filter property to Predefined and choose Prompt editor to specify the filter's prompt properties.
Figure 2-48 shows the Output Columns pane for a map using an ODA data source.

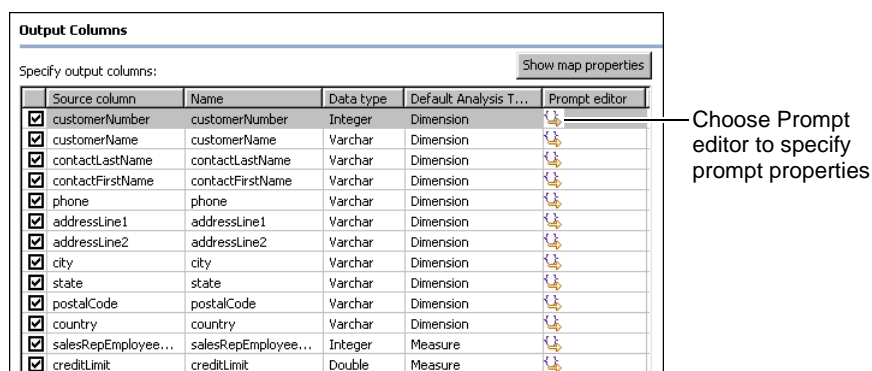


Figure 2-48 Viewing the output columns for a map using an ODA data source

To define other column properties, select the column and define the properties in Properties.

How to modify a .epr file for an ODA data source query

- 1 In Navigator, double-click the appropriate .epr file.
- 2 In General:
 - To modify the query in Query text, choose Query Builder.
 - In Description, type a description for the query.
Figure 2-49 shows General with a description of the query and the Query Builder button.

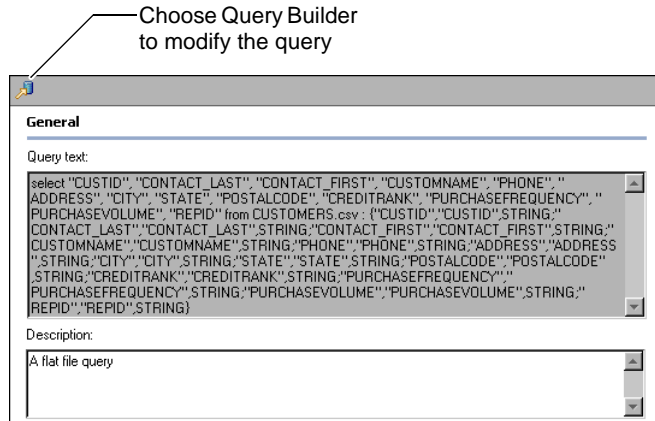


Figure 2-49 Choosing Query Builder to modify the query

3 Choose Parameters.

The query's parameters appear.

4 In Parameters:

- To rename a parameter, type the new name in Parameter.
The IO Design perspective assigns a default name to a parameter based on its position in the query, for example param_1.
- If necessary, choose the correct data type from the Data type drop-down list.
- In Default value, type the parameter's default value.
Do not create an expression. You cannot type a default value for an output parameter.
- To specify the parameter's prompt properties, choose Prompt editor.
Figure 2-50 shows the Parameters pane for a map using an ODA data source.

Parameters			
Parameter	Data type	Default value	Prompt editor
param_0	Integer		
param_1	Varchar	'NYC'	
param_2	Integer	1	

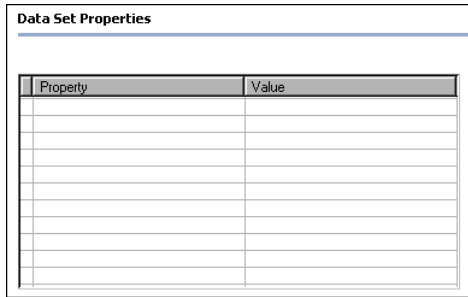
Choose Prompt editor to specify prompt properties

Figure 2-50 Specifying prompt properties for an ODA data source query's parameters

To define other parameter properties, select the parameter and define the properties in Properties.

5 Choose Data Set Properties.

The data set properties appear, if the data set has any. Figure 2-51 shows the Data Set Properties pane.



The figure shows a window titled "Data Set Properties". Inside the window is a table with two columns: "Property" and "Value". The table has a header row with these column names and several empty rows below it for data entry.

Property	Value

Figure 2-51 Data Set Properties pane

6 In Value, type values for the data set properties.

Creating information objects

This chapter contains the following topics:

- Creating an information object
- Creating a graphical information object query
- Creating a textual information object
- Displaying and testing information object output
- Displaying a data source query
- Understanding query execution plan operators
- Storing a query plan with an information object
- Localizing an information object

Creating an information object

You build an information object from maps or other information objects. The IO Design perspective places an information object file in the project's Information Objects folder, as shown in Figure 3-1. An information object file name has an .iob extension. Information object file names are not case-sensitive.

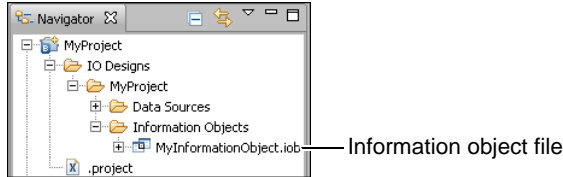


Figure 3-1 Default location for an information object file

How to create an information object

- 1 In Navigator, select the appropriate project.
- 2 Choose File→New→Information Object.
- 3 In New Information Object, accept the default location or deselect Use Default Location and type the path or select a folder.
- 4 In File name, type a name for the information object file, as shown in Figure 3-2. Do not use a name that contains only numbers, for example, 123.

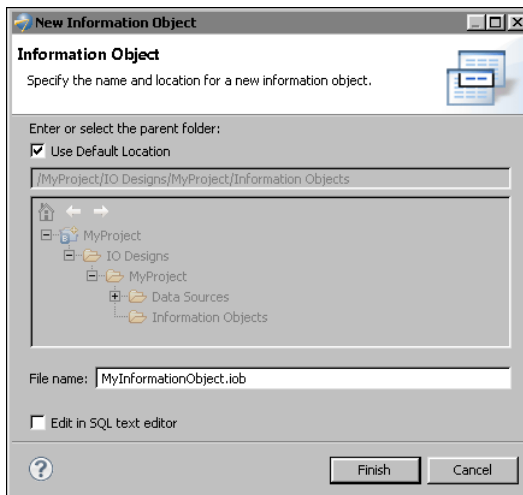


Figure 3-2 Specifying the name and location for a new information object

- 5 To type or paste an Actuate SQL query instead of creating it graphically, select Edit in SQL text editor.
- 6 Choose Finish.
The information object file name appears in Navigator, and the graphical or textual information object editor appears.
- 7 Specify the query for the information object graphically, or using the SQL text editor.

Creating a graphical information object query

In the graphical information object editor, perform the following tasks:

- Choose maps and information objects.
- Specify output columns.
- Specify column categories.
- Specify joins.
- Specify filters.
- Specify the GROUP BY clause.
- Specify the HAVING clause.
- Specify parameters.

Using the expression builder

Many of the steps to create an information object query involve specifying a column or an expression.

When designing a query, you can use Actuate SQL expressions to specify filters or joins, create aggregate data, and so on. For example, you can type expressions, such as `officeID = 101` to specify that the data returned by the query must have 101 in the `officeID` column.

You can type these expressions in either the graphical editor or the textual editor. In the textual editor, you type the expressions as part of the SQL `SELECT` statement. In the graphical query editor, you can type the expressions or use Expression Builder to develop expressions.

Expression Builder helps you create expressions by providing a graphical interface with selections for the available parts of an expression. In Expression Builder, you can build the expressions graphically by selecting constants, operators, functions, column names and so on from a list.

You can use Expression Builder to create Actuate SQL expressions on the following tabs in the graphical information object editor:

- Columns
- Joins
- Filters
- Having
- Parameters

Figure 3-3 shows Expression Builder. You can drag items from the left pane to the right pane or insert items by choosing the appropriate icon. If you select a function in the left pane, the function signature appears in the bottom pane.

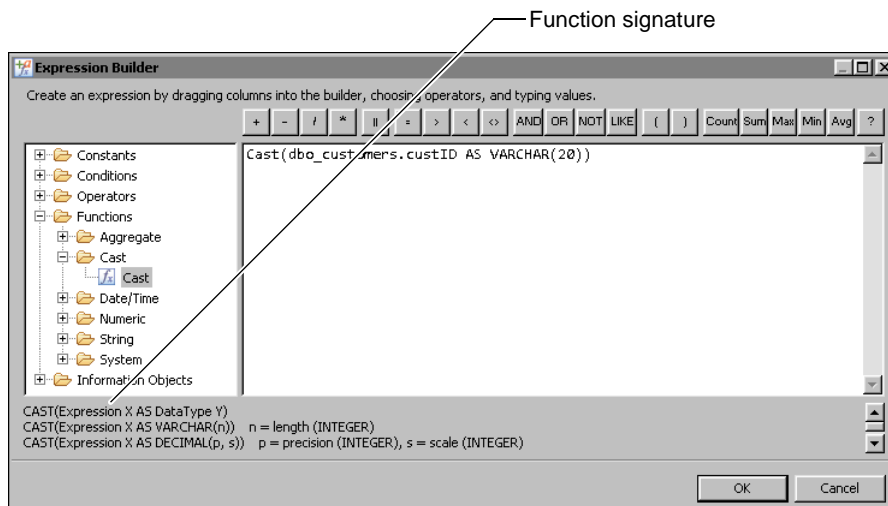


Figure 3-3 Using Expression Builder to create expressions

Choosing maps and information objects

The first step in specifying the query for an information object is to choose the maps and information objects used by the query.

How to choose maps and information objects

- 1 In Navigator, expand the project node and folders to see the maps and information objects.
- 2 Drag the appropriate maps or information objects from Navigator to the upper pane of the graphical information object editor. The columns available in each map or information object appear, as shown in Figure 3-4.

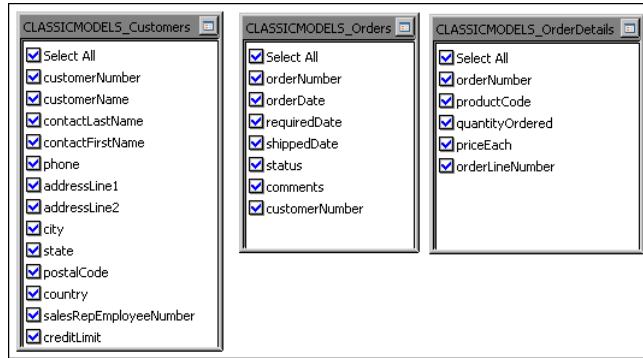


Figure 3-4 Columns available in each selected map or information object

How to open a source map or information object in an editor

If a project contains a large number of maps and information objects, it may be difficult to locate an information object's sources in Navigator. Instead, you can open a source map or information object directly from the graphical information object editor by clicking the button in the source's upper right corner, as shown in Figure 3-5.

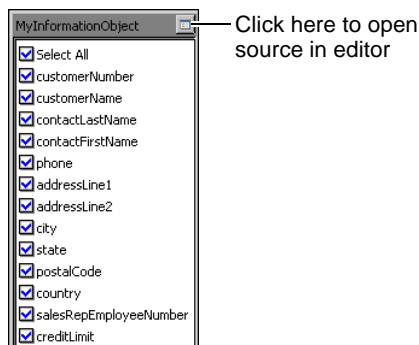


Figure 3-5 Opening a source information object in an editor

Defining output columns

To define the output columns for an information object, use the Columns page. For example, you can create the following SQL fragment:

```
SELECT ename AS employee, (salary * 12) AS annual_comp
FROM Employees
```

How to define output columns

- 1 In the graphical information object editor, choose Columns.

- 2 In the upper pane, select the columns that you want to include and deselect the columns that you want to exclude from the query. To select all columns, select Select All at the top of the listing for that map or information object. By default, all columns in an information object are included in the query. The columns that you select appear in Columns.
- 3 In Columns:
 - To return only distinct rows, select Distinct values only. Some queries return duplicate rows. In a group of duplicate rows, each selected column contains the same value for all the rows in the group. If you want the query to return only one row for each group of duplicate rows, select Distinct values only. This setting affects only rows in which all column values match. The query still returns rows in which only some of the column values match. If the Analysis Type property is set to Dimension or Attribute for all columns in an information object, the DISTINCT keyword is automatically included in the query generated in BIRT Studio when the information object is used as a data source.
 - To change a column alias, type the new alias in Name. Decide on column aliases before you build another information object from this information object. Changing a column alias after you build a dependent information object results in a compiler error in the dependent information object. If a column alias contains a special character, such as a period (.) or a space, enclose the alias in double quotation marks ("). Do not use column aliases that are identical except for case. For example, do not use both status and STATUS as column aliases.
 - To enter an expression, select the source column, and type the expression or choose Ellipsis, as shown in Figure 3-6. Choosing Ellipsis opens Expression Builder.

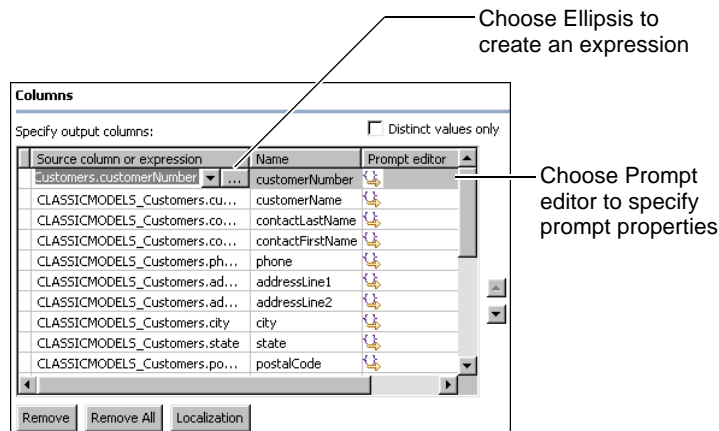


Figure 3-6 Defining output columns



- To create a filter on a column, set the column's Filter property to Predefined, and choose Prompt editor to specify the filter's prompt properties.
 - To change the order of the columns, use the up and down arrows. If the information object uses column categories, you must reorder the columns in Column Categories.
- 4 To define column properties, such as the display name, select the column in Columns, and define the properties in Properties.

How to delete output columns

To delete an output column, select the column in Columns, and choose Remove. To delete all output columns, choose Remove All.

Creating and displaying column categories

If an information object has a large number of output columns, it is difficult for a user to locate a particular column. To help the user locate columns, you can organize them into categories. For example, for an information object that returns customer data, you can create a Customer address category that contains the columns StreetAddress, City, State, and PostalCode.

Creating column categories

Use the Column Categories page to create column categories. In Figure 3-7, Column Categories lists a category with two columns.

How to create a column category or subcategory

- 1 In the graphical information object editor, choose Column Categories.
- 2 On Column Categories, right-click the Root node or a category name and choose Create.
- 3 On New Category, type the category or subcategory name and press Enter. You can provide a description for the category or subcategory in the Properties view.
- 4 Drag-and-drop columns into the category or subcategory. Figure 3-7 shows the result of creating a Contact name category and moving the contact_first and contact_last columns into the category.

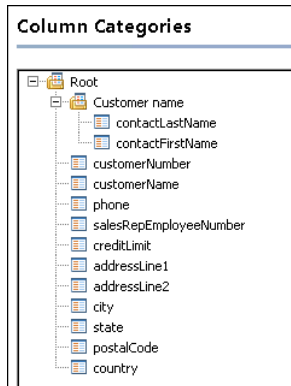


Figure 3-7 Result of moving two columns into a new category

How to rename, move, or remove a category or subcategory

Table 3-1 explains how to work with categories, subcategories, and columns on Column Categories.

Table 3-1 Using Column Categories

To perform this task...	Do the following...
Rename a category or subcategory.	Right-click the category or subcategory name and choose Rename.
Move a category or subcategory.	Drag-and-drop the category or subcategory in the target location.
Remove a category or subcategory.	Right-click the category or subcategory name and choose Remove→Category only.
Remove a category or subcategory and its subcategories.	Right-click the category or subcategory name and choose Remove→Category and subcategories.
Remove all categories and subcategories.	Right-click the Root node and choose Remove→Category and subcategories.
Move a column.	Drag-and-drop the column in the target location. Reordering columns in Column Categories also reorders the columns in Columns.

Displaying column categories

The column categories that you create for an information object appear in the IO Design perspective, Information Object Query Builder, and BIRT Studio.



In the IO Design perspective, column categories appear in the Navigator view and the expression builder. Column categories do not appear in the upper pane of the graphical information object editor. To display column categories in the upper pane of the graphical information object editor, select Toggle categories view in the upper right corner of the information object, as shown in Figure 3-8. The information object on the left does not display column categories. The information object on the right displays the Customer address category.

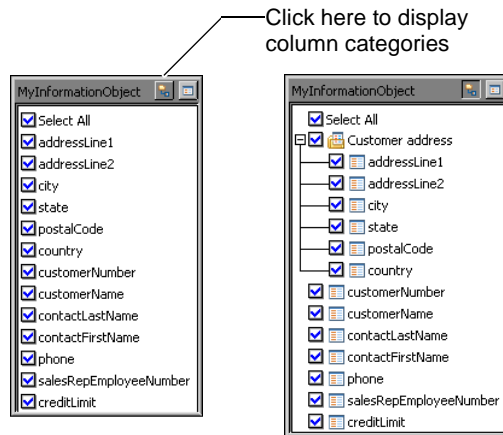


Figure 3-8 Information object with and without categories displayed

If you want column categories to display by default, choose Window>Preferences>Actuate BIRT Project>Information Objects and select Show categories in graphical editor by default, as shown in Figure 3-9.

Column categories do not appear in the expression builder or the upper pane of the graphical information object editor for the information object in which they are defined. Column categories appear for information objects built from this source information object, in other words, for its dependent information objects.

In the Information Object Query Builder, column categories appear in Available Data, the upper pane of the query builder, and the expression builder.

In BIRT Studio, column categories appear in the Available Data pane.

Categories that do not contain columns appear in the Navigator view in the IO Design perspective, but not in the Information Object Query Builder or BIRT Studio.

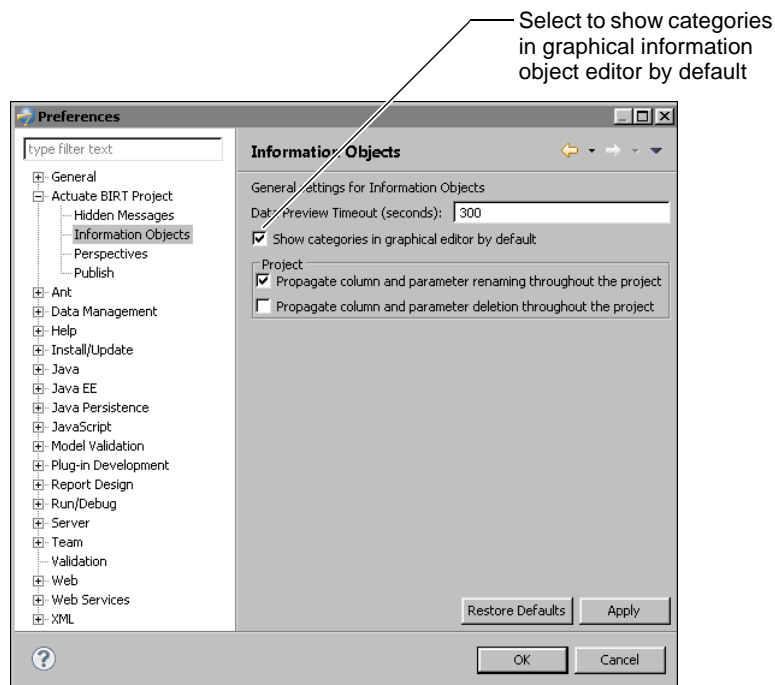


Figure 3-9 Showing categories by default

Setting column properties

You set most column properties in the Properties view. You set default values for analytics properties in the Define Default Column Analytics wizard.

Setting column properties in the Properties view

Table 3-2 lists column properties visible in the Properties view and a description of each property.

Table 3-2 Column properties visible in the Properties view

Column property	Can set?	Description
Aggregate Type	Yes, in Define Default Column Analytics wizard	Default aggregate function for a column in a dashboard or BIRT Studio summary table, for example SUM.
Category Path	No	Path for column category and subcategories.
Conceal Value	Not used	Not used.

Table 3-2 Column properties visible in the Properties view (continued)

Column property	Can set?	Description
Data Type	No	Actuate SQL data type. If the data type is unknown, choose the Compile IO button.
Default Value	Not used	Do not specify a default value.
Description	Yes	Description of the column that appears when the column is selected in BIRT Studio.
Description Key	Yes	Key for Description property in localization properties file.
Display Control Type	Yes, in Prompt editor	Control type for a dynamic filter on this column in the report design. The available values are: text box, read-only drop-down list, editable drop-down list, or radio buttons.
Display Format	Yes	Format to apply to column values in BIRT Studio output. To specify the display format, use an Actuate Basic format pattern or format keyword, such as Short date.
Display Length	Yes	Number of characters to allow for display of column values in report output.
Display Name	Yes	Display name for the column in BIRT Studio.
Display Name Key	Yes	Key for Display Name property in localization properties file.
Do Not Prompt	Not used	Not used.
Expression	Yes, on the Columns tab	Expression for a computed field.
Filter	Yes	To create a dynamic filter on the column in the report design, set to Predefined. To enable a user to create a dynamic filter on this column, set to Optional. To prevent filtering on this column, set to Disabled.
Has Null	Yes	If column contains NULLs, set to True. Otherwise, set to False.
Heading	Yes	The heading for the column in BIRT Studio output.
Heading Key	Yes	Key for Heading property in localization properties file.
Help Text	Yes	Balloon help for the column in BIRT Studio.

(continues)

Table 3-2 Column properties visible in the Properties view (continued)

Column property	Can set?	Description
Help Text Key	Yes	Key for Help Text property in localization properties file.
Horizontal Alignment	Yes	Horizontal alignment of column values in BIRT Studio output. The available values are: left, right, or center.
Indexed	No	Indicates whether the column is indexed in the data source. True indicates that the column is indexed. False indicates that it is not indexed.
Name	Yes, on the Columns tab	The alias for the column in the information object query.
Required	Not used	Not used.
Text Format	Yes	Not used.
Word Wrap	Yes	To display text on multiple lines in BIRT Studio (with fixed width layout preference) output if the length of the text exceeds the width of the column, set to True. To truncate the text, set to False. This property is used only for columns that have the VARCHAR data type.

Setting default values for analytics properties

You can use an information object as a data source in a summary table in a dashboard or a BIRT Studio report. To create a summary table, users select a table's auto-summarize feature, then select the data set column or columns whose data to group and aggregate. Because the grouping and aggregating are performed automatically, you must set default values for the analytics properties for each column. The analytics properties provide the appropriate context for these tasks. For example, it makes sense to group sales data by region or product line, but not by revenue. Conversely, it makes sense to aggregate revenue values, but not region or product line values.

To provide the appropriate information to generate a summary table, set each column's analysis type property to one of the following values:

- **Dimension**
The dimension analysis type supports the grouping of data in the column. For example, to display revenue by region, set the region column as a dimension.
- **Attribute**

An attribute describes the items associated with a dimension. For a product dimension, for example, attributes might include color, size, and price. When you set a column as an attribute, you must also specify the dimension column of which it is an attribute. The summary table cannot group data in an attribute column.

- **Measure**

The measure analysis type supports the aggregation of values in the column. For example, to calculate revenue totals, set the revenue column as a measure. The summary table cannot group data in a measure column.

If you do not set default values for the analysis type property, the following default values are used:

- If the column contains numeric values or the data type is unknown, the default is measure.
- If the column contains data of type `TIMESTAMP`, `VARCHAR`, or `BOOLEAN`, the default is dimension.
- If the column is a primary key, a foreign key, or an indexed column in the database, the default is dimension regardless of the column's data type.

Sometimes, the default values do not provide usable data for a summary table, so you should assign an analysis type for every column in an information object. One problem is that the default analysis type for columns that contain numeric values is measure. In some cases, however, users want to group on numeric values. For example, for a report that shows order numbers and order totals, users want to group on order number, but data in measure columns cannot be grouped.

If analysis type is set to Dimension or Attribute for all columns in an information object, the `DISTINCT` keyword is included in the query generated in BIRT Studio when the information object is used as a data source.

Use the Define Default Column Analytics wizard to specify default values for the following analytics properties for information object columns:

- For each output column, specify the analysis type: dimension, measure, or attribute.
- For each attribute column, specify the dimension of which it is an attribute.
- For each measure column, specify the aggregate function.

How to specify default values for analytics properties

- 1 Open the information object or map in the graphical or textual editor.
- 2 Choose Columns or Output Columns.
- 3 Choose Default Analytics, as shown in Figure 3-10.

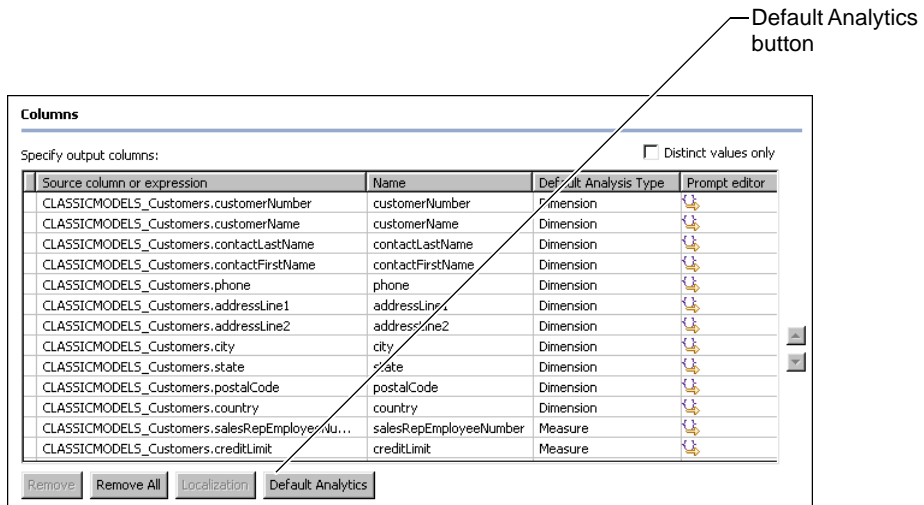


Figure 3-10 Default Analytics button in Columns

- 4 In the first page of the Define Default Column Analytics wizard, specify the analysis type for each column, as shown in Figure 3-11. Choose Next.

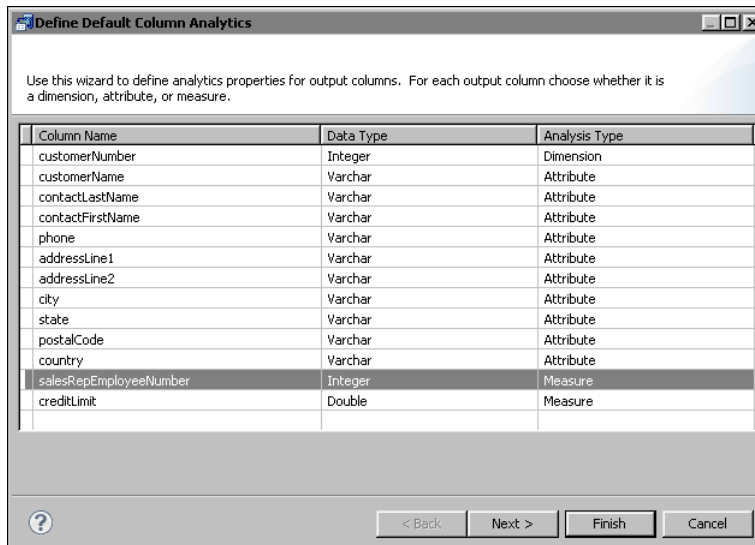


Figure 3-11 Specifying the analysis type for information object columns

- 5 In the second page of the Define Default Column Analytics wizard, specify the dimension with which an attribute is associated, as shown in Figure 3-12. Choose Next.

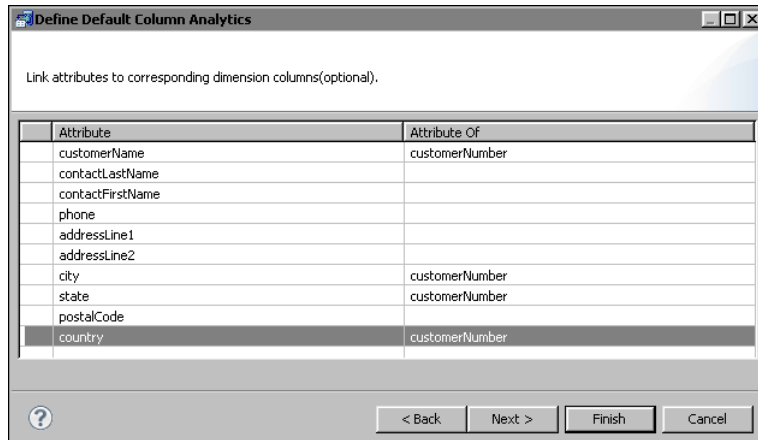


Figure 3-12 Specifying the dimensions with which attributes are associated

- 6 In the third page of the Define Default Column Analytics wizard, specify the aggregate function for measure columns, as shown in Figure 3-13. You can choose a function from the drop-down list or type the name of a function. Choose Finish.

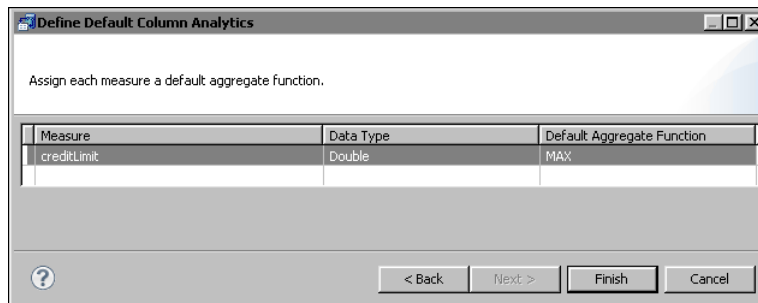


Figure 3-13 Specifying the aggregate function for creditLimit

About column property inheritance

When you build an information object, its output columns inherit property values from the parent maps or information objects. For example, if you use an information object called IO1 to build another information object called IO2, IO2's output columns inherit property values from the corresponding columns in IO1. If a column property value in IO1 changes, the change is propagated to IO2. For example, if the horizontal alignment for IO1.column01 changes from left to right and column01 is an output column in IO2, the horizontal alignment for IO2.column01 also changes from left to right. Changes to a map or information object's Name property are not propagated, however. In Figure 3-14, many of the column's property values are inherited from the parent map.

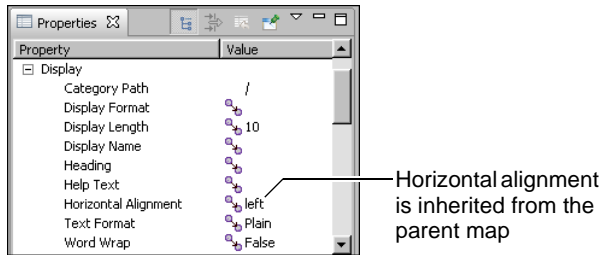


Figure 3-14 Inheritance of property values

If you change a property value for an output column, that property value is no longer inherited from the parent map or information object. For example, if you change the horizontal alignment for IO2.column01 to center and the horizontal alignment for IO1.column01 later changes to left, the change is not propagated to IO2.column01. In Figure 3-15, the column's horizontal alignment is not inherited from the parent map.

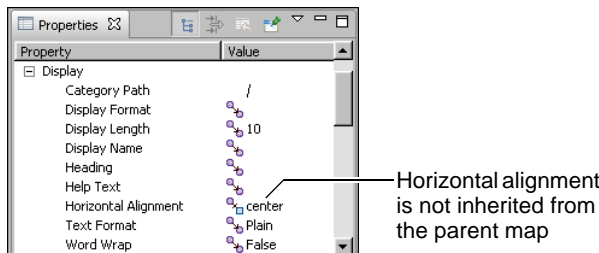


Figure 3-15 Changing property value inheritance

Choosing Reset for the appropriate property in Properties, as shown in Figure 3-16, or in Prompt editor resets the property's value to the inherited value. Any future changes to the property's value in the parent map or information object are propagated.

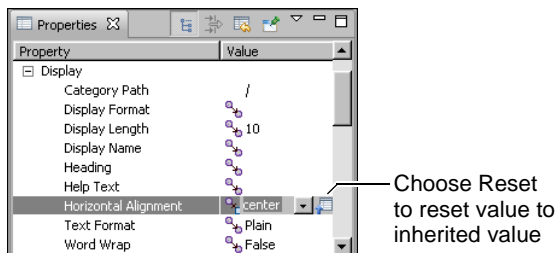


Figure 3-16 Resetting a property's value

Values for the following column properties are inherited from the parent map or information object unless the values in the parent map or information object are blank:

- Description
- Display Name
- Heading
- Help Text

If the values in the parent map or information object are blank, the inheritance rules for these properties are as follows:

- If you do not set the Display Name property, the Display Name property takes the value of the column's Name property.
- If you do not set the Heading property, the Heading property takes the value of the column's Display Name property.
- If you do not set the Description property, the Description property takes the value of the column's Heading property.
- If you do not set the Help Text property, the Help Text property takes the value of the column's Description property.

In other words, if you do not set any of these properties and the values in the parent map or information object are blank, they all take the value of the column's Name property. In this case, propagation of these properties occurs at run time.

Creating a filter for use in report designs

A predefined filter restricts the data returned by a query built from an information object using Information Object Query Builder. Set a column's Filter property to Predefined to create a dynamic filter in a report design in BIRT Designer Professional.

Use Prompt editor to specify the filter's display control type and list of values. Do not provide a default value. You create a list of values by specifying the values or by typing an Actuate SQL query that retrieves the values. You can specify the filter values as well as the values displayed to the user. If you type a query, the query must meet the following requirements:

- The query must retrieve one or two columns from an information object or map, for example:

```
SELECT DISTINCT custID, customName
FROM "MyInformationObject.iob"
ORDER BY 2
```

The first column contains the filter values. The second column contains the values displayed to the user. The information object or map must reside in the

same volume as the IOB used as a data source in the report design. If you use a relative path to reference the information object or map, BIRT Designer Professional interprets the path as relative to the IOB used as a data source. If the information object or map defines a parameter, you must provide a value for the parameter, for example:

```
SELECT DISTINCT custID, customName
FROM "MyInformationObject.iob" ['CA']
ORDER BY 2
```

- The query must not contain a WITH clause.

How to create a filter for use in a report design

1 Select the appropriate column in Columns.

2 In Properties, set Filter to Predefined.



3 In the row for that filter, choose Prompt editor.

4 On Prompt editor, complete the following tasks:

- In Show as, select the display control type. The choices available and appearance of the page depend on the display control type you select.
- If you use a display control type other than Text box, you can specify a list of values for the user to choose by typing the values and, optionally, the display names, as shown in Figure 3-17. Alternatively, you can select from a list of database values by choosing Select Values.

Specify the prompt properties for this parameter. Prompt properties specify the behavior and appearance of filters that appear on requester pages for reports using this information object. Users can specify values for these filters to limit the data in a report.

Show as

☐ Text box ☐ Dynamic list of values

☒ Drop-down list (read only) ☐ Auto suggest

☐ Combo box (editable) Start Auto suggest after 1 character(s)

☐ Radio buttons

Values

Default value: 'CA'

Value	Display name
'CA'	California
'MA'	Massachusetts
'NY'	New York
'PA'	Pennsylvania

Select from a list of database values

Type values and display names

Figure 3-17 Typing values and display names for a filter

To create an Actuate SQL query that retrieves the values, select Dynamic list of values, as shown in Figure 3-18, and do one of the following:

- Type the query.
- Choose Generate Query.
Choosing Generate Query creates a query that retrieves the distinct values from the column for which you are creating a filter, as well as display names. If the information object used in the query has parameters, you must provide parameter values.

If you select Combo box (editable), Dynamic list of values, and Auto suggest, a drop-down list appears after the information object user types the number of characters specified in Start Auto suggest after N character(s). The list contains values that begin with the characters the user typed. For example, if the user typed 'Atel' and N=4, the list contains the value 'Atelier graphique'. In this case, the query that retrieves the values must select two columns, a value column and a display name column.

Choose OK.

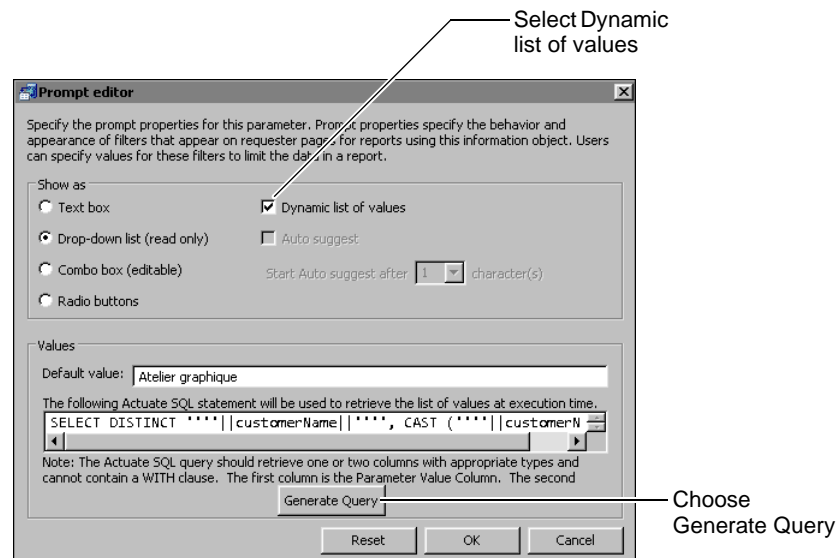


Figure 3-18 Creating an Actuate SQL query to generate values for a filter

Specifying a join

To define the joins for an information object, use the Joins page. For example, you can create the following SQL fragment:

```
FROM Customers INNER JOIN Orders ON (Customers.custID =
Orders.custID)
```

About joins

A join specifies how to combine data from two maps or information objects. The maps or information objects do not have to be based on the same data source. A join consists of one or more conditions that must all be true. In the resulting SQL SELECT statement, join conditions are linked with AND.

A join can consist of multiple conditions in the following form:

```
columnA = columnB
```

A join can have only one condition that uses an operator other than equality (=) or an expression, for example:

```
columnA < columnB
```

The IO Design perspective does not support right outer joins or full outer joins.

How to define a join condition

- 1 In the graphical information object editor, choose Joins.
- 2 In the upper pane, drag the join column from the first information object or map, and drop it on the join column in the second information object or map.

The upper pane shows the join condition, like the one in Figure 3-19, and the join columns and operator are listed in the lower pane.

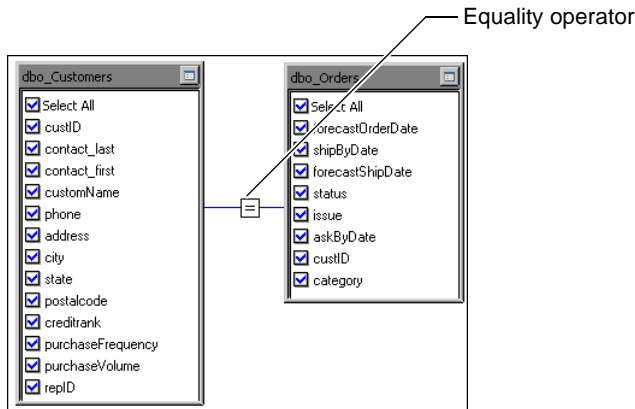


Figure 3-19 Joined columns from two information objects or maps

- 3 In the lower pane, select the row that describes the new join condition.
- 4 If necessary, select a different join condition operator from the drop-down list. By default, the IO Design perspective uses the equality operator (=) to relate two columns.
- 5 To change a column name to an expression, select the column name, and type the expression, or choose Ellipsis to display the expression builder, as shown in Figure 3-20.

If the join has a condition that uses an operator other than equality (=) or an expression, the upper pane marks the join line with the symbol that appears in Figure 3-21.

- 6 If the join consists of more than one condition, repeat this procedure for the other conditions.
- 7 Choose one of the following join types:
 - Inner join
 - Left outer join
- 8 Optimize the join.

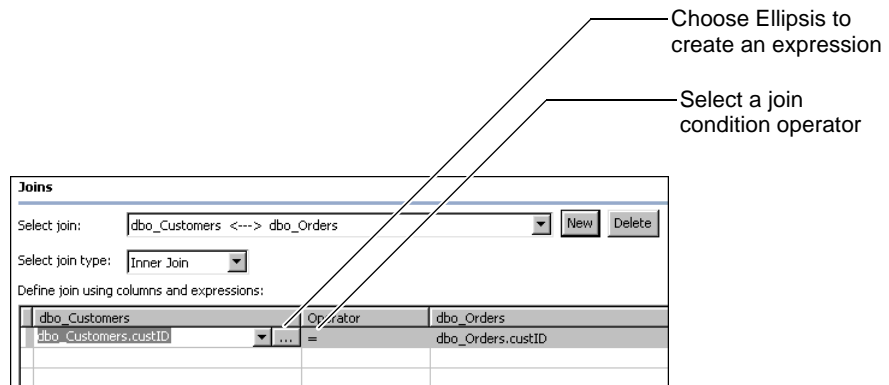


Figure 3-20 Defining a join

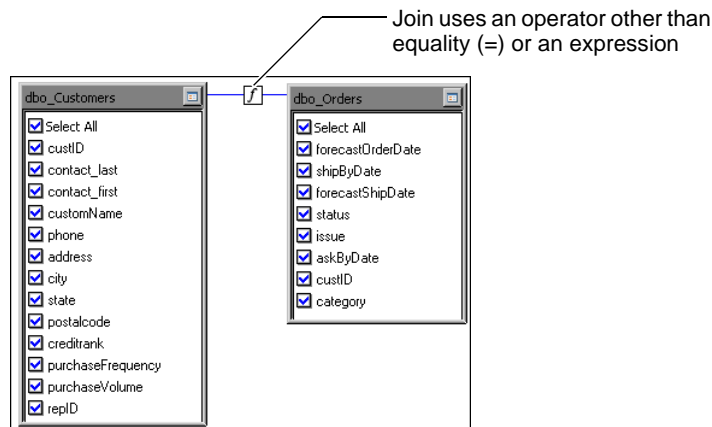


Figure 3-21 A join condition that uses an expression or an operator other than equality

How to delete a join condition

To delete a join condition, select the join condition in the upper pane of the graphical information object editor and press Delete.

Optimizing joins

You can improve a query's performance by optimizing the joins. To optimize a join, use the **CARDINALITY** and **OPTIONAL** keywords in the Actuate SQL query. To optimize a join, you can specify the cardinality of the join. Specifying the cardinality of the join adds the **CARDINALITY** keyword to the Actuate SQL query.

You can also specify whether a table in a join is optional. Specifying that an information object is optional adds the **OPTIONAL** keyword to the Actuate SQL query. If you indicate that a table is optional and none of its columns appear in the query created by a report developer or business user (except in a join condition), the table is dropped from the optimized query.

If the maps or information objects are based on different data sources, there are two additional ways to optimize a join:

- Use a join algorithm.
- Use map and join column properties.

Figure 3-22 shows how to specify the cardinality of an information object or map in a join, whether an information object is optional, and how to specify a join algorithm in Joins.

The screenshot shows the 'Joins' dialog box with the following fields and annotations:

- Specify relationship:** Two dropdown menus. The first is set to '0 or more' and is annotated with 'Applies CARDINALITY keyword'. The second is set to '1'.
- Specify join algorithm:** A dropdown menu set to 'Dependent', annotated with 'Specifies join algorithm'.
- Specify query trimming hint:** A text field containing 'dbo_Customers <---> dbo_Orders'.
- Optional:** Two checkboxes. The first is unchecked, and the second is checked, annotated with 'Applies OPTIONAL keyword'.

Figure 3-22 Optimizing a join

Using join algorithms

When you join maps or information objects that are built from different data sources, the Actuate SQL compiler chooses a join algorithm. If you have a good understanding of the size and distribution of the data, however, you can specify the join algorithm. Choosing the correct join algorithm can significantly reduce information object query execution time. Actuate SQL supports three join algorithms:

- Dependent
- Merge
- Nested Loop

When you join maps or information objects that are built from the same data source, specifying a join algorithm has no effect. The join is processed by the data source.

About dependent joins

A dependent join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results. The results are then processed one at a time (pipelined).
- For each left side result, the right side of the join is executed, parameterized by the values provided by the current left side row.

A dependent join is advantageous when the cardinality of the left side is small, and the selectivity of the join criteria is both high and can be delegated to the data source. When the cardinality of the left side is high, a dependent join is relatively slow because it repeatedly executes the right side of the join.

Dependent joins can be used for any join criteria, although only join expressions that can be delegated to the right side's data source result in improved selectivity performance.

About merge joins

A merge join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results sorted by the left side data source. The results are then processed one at a time (pipelined).
- The right side of the join statement is executed, retrieving all the results sorted by the right side data source. The results are then processed one at a time (pipelined).

A merge join can only be used with an equijoin. A merge join has much lower memory requirements than a nested loop join and can be much faster. A merge join is especially efficient if the data sources sort the rows.

About nested loop joins

A nested loop join is processed in the following way:

- The left side of the join statement is executed, retrieving all the results. The results are then processed one at a time (pipelined).
- The right side of the join statement is executed. The results are materialized in memory. For each row on the left side, the materialized results are scanned to find matches for the join criteria.

A nested loop join is advantageous when the cardinality of the right side is small. A nested loop join performs well when the join expression cannot be delegated to

the data source. A nested loop join can be used for any join criteria, not just an equijoin.

A nested loop join is a poor choice when the cardinality of the right side is large or unknown, because it may encounter memory limitations. Increasing the memory available to the Integration service removes this limitation. The Integration service parameter *Max memory per query* specifies the maximum amount of memory to use for an Integration service query. For more information about this parameter, see *Configuring BIRT iHub*.

How to specify a join algorithm

In Joins, select the appropriate join and choose one of the following from the Specify join algorithm drop-down list shown in Figure 3-23:

- Dependent
- Merge
- Nested loop

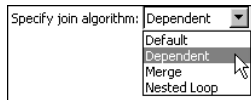


Figure 3-23 Specifying the join algorithm

Improving the selectivity of a join

When you join maps that are based on different data sources, you can optimize the join by providing values for map and join column properties. Providing values for these properties improves the selectivity of the join. You should provide values for:

- The maps' Cardinality property
Cardinality specifies the number of rows returned by the map, or gives an approximation based on the possible parameter values.
- The following join column properties:
 - Distinct Values Count
Distinct Values Count specifies the number of distinct column values.
 - Max Value
Max Value specifies the maximum column value.
 - Min Value
Min Value specifies the minimum column value.

Max Value and Min Value are not used for columns of character data type. When providing values for Max Value and Min Value, use the appropriate

format. For example, if the column is of type `TIMESTAMP`, Max Value must be in the following format:

```
TIMESTAMP '2001-02-03 12:11:10'
```

You should also provide values for these properties for a column used in a `WHERE` clause.

How to provide a value for the Cardinality property

- 1 In Navigator, double-click one of the maps in the join.
- 2 On Output Columns, choose Show map properties, as shown in Figure 3-24.

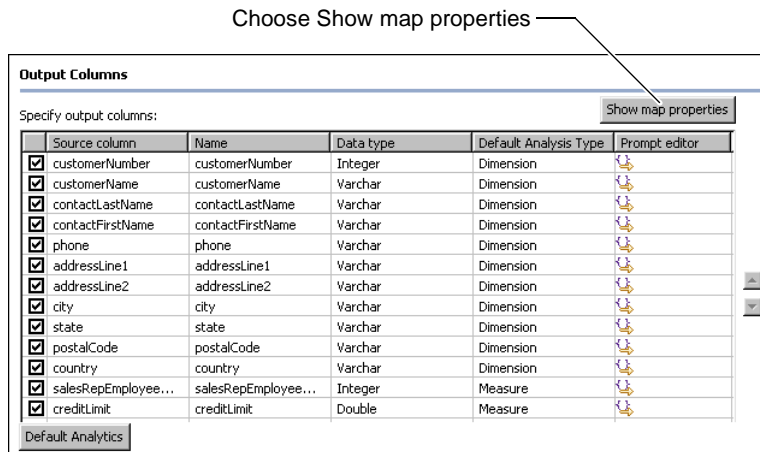


Figure 3-24 Choosing Show map properties

- 3 In Properties, type a value for Cardinality.
- 4 Repeat this procedure for the other map.

How to provide values for join column properties

- 1 In Output Columns, select the join column.
- 2 In Properties, type values for:
 - Distinct Values Count
 - Max Value
 - Min Value
- 3 Repeat this procedure for the join column in the other map.

Creating a Cartesian join

By default, an information object user cannot create a query with two information objects, for example a Customers information object and an Orders information

object, without explicitly joining the information objects. The absence of an explicit join is called a Cartesian join. Cartesian joins can consume database resources and return very large result sets. In some cases, however, it is acceptable to create a Cartesian join. For example, a map of a single-row system information table does not have to be joined to another map or information object. If it is acceptable for a map or information object to be used in a Cartesian join, set Allow this Source to be used in Cartesian Joins to True. To display this property for a map, choose Show map properties as shown in Figure 3-24. To display this property for an information object, click in the white space in the upper pane of the query editor as shown in Figure 3-63.

Filtering data

If an information object returns more data rows than you need, you can restrict the number of data rows by using a filter. For example, rather than list all customer sales, you can create a filter to select only the sales data for a particular week or only the sales data for a particular region.

Filtering data helps you work effectively with large amounts of data. It enables you to find the necessary pieces of information to answer specific business questions, such as which sales representatives generated the top ten sales accounts, which products generated the highest profit in the last quarter, which customers have not made a purchase in the past 90 days, and so on.

Filtering data can also have a positive effect on processing speed. Limiting the number of data rows can reduce the load on the databases because the information object does not need to return all the rows every time it is run.

Creating a filter condition

When you create a filter, you define a condition that specifies which data rows to return. A filter condition is an If expression that must evaluate to true in order for a data row to be returned. For example:

```
If the order total is greater than 10000
If the sales office is San Francisco
If the order date is between 4/1/2008 and 6/30/2008
```

Filter conditions are appended to the information object's WHERE clause, for example:

```
WHERE OrderTotal > 10000 AND SalesOffice LIKE 'San Francisco%' AND
      OrderDate BETWEEN TIMESTAMP '2008-04-01 00:00:00' AND TIMESTAMP
      '2008-06-30 00:00:00'
```

Figure 3-25 shows an example of a condition defined in Filter Conditions.

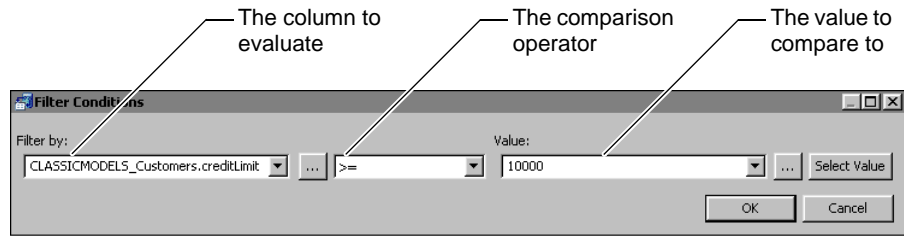


Figure 3-25 Filter Conditions displaying a filter condition

As Figure 3-25 shows, Filter Conditions helps you define the condition by breaking it down into the following parts:

- The column to evaluate, such as credit limit.
- The comparison operator that specifies the type of comparison test, such as > (greater than).
- The value to which all values in the column are compared, such as 10000.

Table 3-3 lists the operators you can use when you create expressions for filter conditions.

Table 3-3 Operators in filter condition expressions

Operator	Use to	Example
BETWEEN	Test if a column value is between two specified values.	Profit BETWEEN 1000 AND 2000
= (Equal to)	Test if a column value is equal to a specified value.	CreditLimit = 100000
> (Greater than)	Test if a column value is greater than a specified value.	Total > 5000
>= (Greater than or equal to)	Test if a column value is greater than or equal to a specified value.	Total >= 5000
IN	Test if a column value is in the specified set of values.	Country IN ('USA', 'Canada', 'Mexico')
IS NOT NULL	Test if a column value is not a null value. A null value means that no value is supplied.	CreditLimit IS NOT NULL
IS NULL	Test if a column value is a null value.	CreditLimit IS NULL
< (Less than)	Test if a column value is less than a specified value.	Total < 5000

(continues)

Table 3-3 Operators in filter condition expressions (continued)

Operator	Use to	Example
<= (Less than or equal to)	Test if a column value is less than or equal to a specified value.	Total <= 5000
LIKE	Test if a column value matches a string pattern.	ProductName LIKE 'Ford%'
NOT BETWEEN	Test if a column value is not between two specified values.	Profit NOT BETWEEN 1000 AND 2000
<> (Not equal to)	Test if a column value is not equal to a specified value.	CreditLimit <> 100000
NOT IN	Test if a column value is not in the specified set of values.	Country NOT IN ('USA', 'Canada', 'Mexico')
NOT LIKE	Test if a column value does not match a string pattern.	ProductName NOT LIKE 'Ford%'

How to create a filter condition

- 1 In the graphical information object editor, choose Filters.
- 2 In Filters, choose New.
- 3 In Filter Conditions, in Filter by, do one of the following:
 - Select a column from the drop-down list. The drop-down list contains the non-aggregate columns that you defined on the Columns page. To create a filter for an aggregate column, use the Having page.
 - Type an expression.
 - Choose Ellipsis to create an expression.
- 4 Select the comparison test, or operator, to apply to the selected column or expression. Depending on the operator you select, Filter Conditions displays one or two additional fields, or a completed filter condition.
- 5 If you selected an operator that requires a comparison value, specify the value in one of the following ways:
 - Type the value or expression.
 - If you selected a column in Filter by, choose Select Value to select from a list of values. Figure 3-26 shows the selection of Boston from a list of possible sales office values.



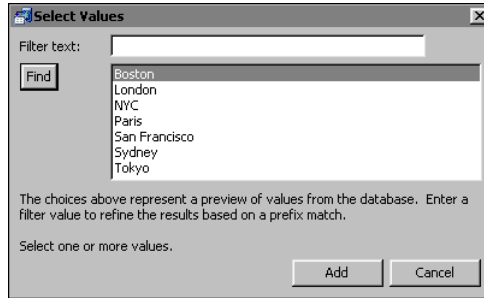


Figure 3-26 Select Value showing the list of values in the selected column

- Select a parameter or column from the drop-down list. You create parameters on the Parameters page.
 - Choose Ellipsis to create an expression.
- Figure 3-27 shows the completed filter condition.

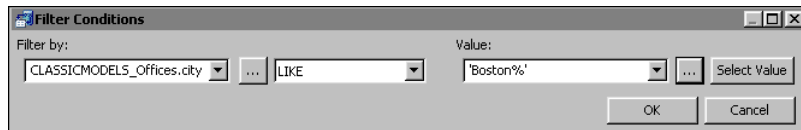


Figure 3-27 Filter Conditions displaying a completed filter condition

Choose OK. The filter condition appears in Filters as shown in Figure 3-28.

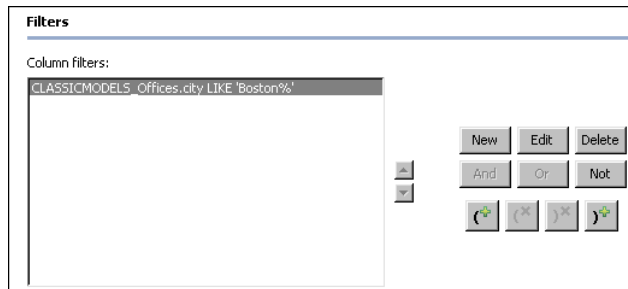


Figure 3-28 Filters page displaying a filter condition

- 6 Display the Actuate SQL query. Verify that the filter condition is appended to the WHERE clause and that the syntax is correct, for example:

```
WHERE SalesOffice LIKE 'Boston%'
```

How to create a filter condition using Actuate SQL

- 1 In the graphical information object editor, choose Filters.
- 2 In Filters, complete the following tasks:
 - Click in the text box.

- Type the filter condition using Actuate SQL, as shown in Figure 3-29. If a table or column identifier contains a special character, such as a space, enclose the identifier in double quotation marks (").

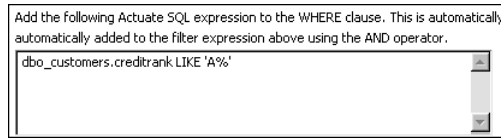


Figure 3-29 Using Actuate SQL to create a filter condition

Selecting multiple values for a filter condition

So far, the filter examples specify one comparison value. Sometimes you need to view more data, for example, sales details for several sales offices, not for only one office. To select more than one comparison value, select the IN operator, choose Select Values, then select the values. To select multiple values, press Ctrl as you select each value. To select contiguous values, select the first value, press Shift, and select the last value. This action selects the first and last values and all the values in between.

Figure 3-30 shows the selection of London and Paris from a list of sales office values.

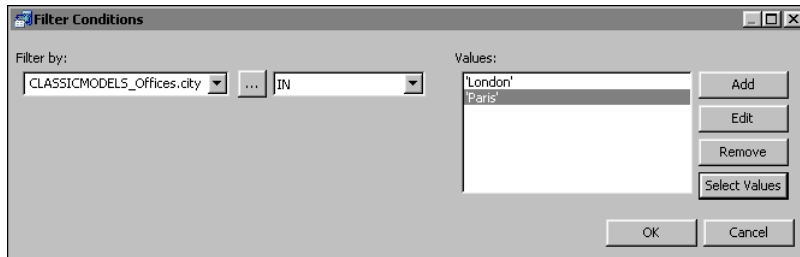


Figure 3-30 Filter Conditions showing the selection of multiple comparison values

Excluding data

You use comparison operators, such as = (equal to), > (greater than), or < (less than), to evaluate the filter condition to determine which data to include. Sometimes it is more efficient to specify a condition that excludes a small set of data. For example, you need sales data for all countries except USA. Instead of selecting all the available countries and listing them in the filter condition, simply use the NOT LIKE operator. Similarly, use NOT BETWEEN to exclude data in a specific range, and <> (not equal to) to exclude data that equals a particular value.

For example, the following filter condition excludes orders with amounts between 1000 and 5000:

```
OrderAmount NOT BETWEEN 1000 AND 5000
```

The filter condition in the next example excludes products with codes that start with MS:

```
ProductCode NOT LIKE 'MS%'
```

Filtering empty or blank values

Sometimes, rows display nothing for a particular column. For example, suppose a customer database table contains an e-mail field. Some customers, however, do not supply an e-mail address. In this case, the e-mail field might contain an empty value or a blank value. An empty value, also called a null value, means no value is supplied. A blank value is entered as " (two single quotes without spaces) in the database table field. Blank values apply to string fields only. Null values apply to all data types.

You can create a filter to exclude data rows where a particular column has null or blank values. You use different operators to filter null and blank values.

When filtering to exclude null values, use the IS NOT NULL operator. If you want to view only rows that have null values in a particular column, use IS NULL. For example, the following filter condition excludes customer data where the e-mail column contains null values:

```
email IS NOT NULL
```

The following filter condition displays only rows where the e-mail column contains null values:

```
email IS NULL
```

When filtering blank values, use the NOT LIKE operator with " (two single quotes without spaces) as the operand. For example, to exclude rows with blank values in an e-mail column, specify the following filter condition:

```
email NOT LIKE ''
```

Conversely, to display only rows where the e-mail column contains blank values, create the following condition:

```
email LIKE ''
```

In a report, you cannot distinguish between an empty value and a blank value in a string column. Both appear as missing values. If you want to filter all missing values whether they are null or blank, specify both filter conditions as follows:

```
email IS NOT NULL AND email NOT LIKE ''
```

Specifying a date as a comparison value

When you create a filter condition that compares the date-and-time values in a column to a specific date, the date value you supply must be in the following format regardless of your locale:

```
TIMESTAMP '2008-04-01 12:34:56'
```

Do not use locale-dependent formats such as 4/1/2008.

Specifying a number as a comparison value

When you create a filter condition that compares the numeric values in a column to a specific number, use a period (.) as the decimal separator regardless of your locale, for example:

```
123456.78
```

Do not use a comma (,).

Comparing to a string pattern

For a column that contains string data, you can create a filter condition that compares each value to a string pattern instead of to a specific value. For example, to display only customers whose names start with M, use the LIKE operator and specify the string pattern, M%, as shown in the following filter condition:

```
Customer LIKE 'M%'
```

You can also use the % character to ensure that the string pattern in the filter condition matches the string in the column even if the string in the column has trailing spaces. For example, use the filter condition:

```
Country LIKE 'USA%'
```

instead of the filter condition:

```
Country = 'USA'
```

The filter condition Country LIKE 'USA%' matches the following values:

```
'USA'  
'USA   '  
'USA      '
```

The filter condition Country = 'USA' matches only one value:

```
'USA'
```

You can use the following special characters in a string pattern:

- % matches zero or more characters. For example, %ace% matches any value that contains the string ace, such as Ace Corporation, Facebook, Kennedy Space Center, and MySpace.
- _ matches exactly one character. For example, t_n matches tan, ten, tin, and ton. It does not match teen or tn.

To match the percent sign (%) or the underscore character (_) in a string, precede those characters with a backslash character (\). For example, to match S_10, use the following string pattern:

```
S\_10
```

To match 50%, use the following string pattern:

50\%

Comparing to a value in another column

Use a filter condition to compare the values in one column with the values of another column. For example, in a report that displays products, sale prices, and MSRP (Manufacturer Suggested Retail Price), you can create a filter condition to compare the sale price and MSRP of each product, and display only rows where the sale price is greater than MSRP.

How to compare to a value in another column

- 1 In the graphical information object editor, choose Filters.
- 2 In Filters, choose New.
- 3 In Filter Conditions, in Filter by, select a column from the drop-down list.
- 4 Select the comparison test, or operator, to apply to the selected column.
- 5 In Value, select a column from the drop-down list.

Figure 3-31 shows an example of a filter condition that compares the values in the priceEach column with the values in the MSRP column.

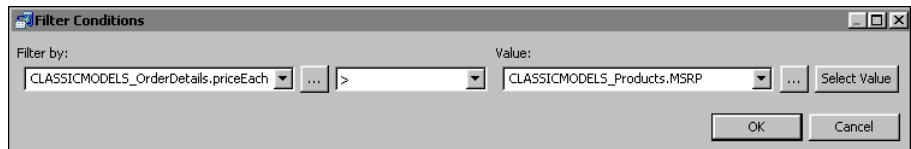


Figure 3-31 Comparing the values in priceEach with the values in MSRP
Choose OK.

Using an expression in a filter condition



An expression is any combination of Actuate SQL constants, operators, functions, and information object columns. When you create a filter condition, you can use an expression in Filter by, Value, or both. You create an expression in the expression builder.

For example, in an information object that returns customer and order data, you want to see which orders shipped less than three days before the customer required them. You can use the DATEDIFF function to calculate the difference between the ship date and the required date:

```
DATEDIFF('d', shippedDate, requiredDate) < 3
```

Figure 3-32 shows this condition in Filter Conditions.

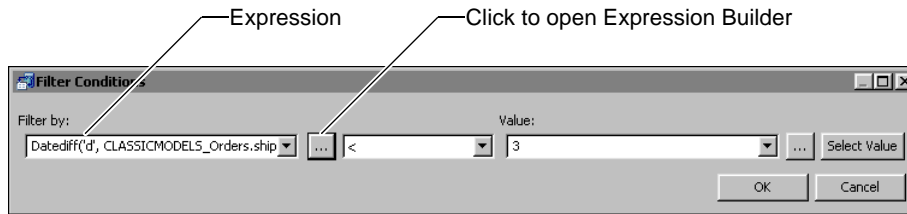


Figure 3-32 Filter Conditions with expression in Filter by

In an information object that returns order data, you want to see which orders were placed today. You can use the `CURRENT_DATE` function to return today's date:

```
orderDate = CURRENT_DATE( )
```

Figure 3-33 shows this condition in Filter Conditions.

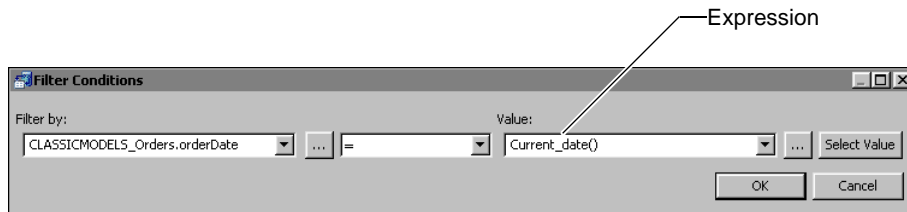


Figure 3-33 Filter Conditions with expression in Value

In an information object that returns employee data, you want the information object to return only data for the user who is currently logged in to the Encyclopedia volume. You can use the `LEFT` function and the concatenation operator (`||`) to construct the employee's user name, and the `CURRENT_USER` function to return the name of the user who is currently logged in:

```
LEFT(firstName, 1) || lastName = CURRENT_USER( )
```

Figure 3-34 shows this condition in Filter Conditions.

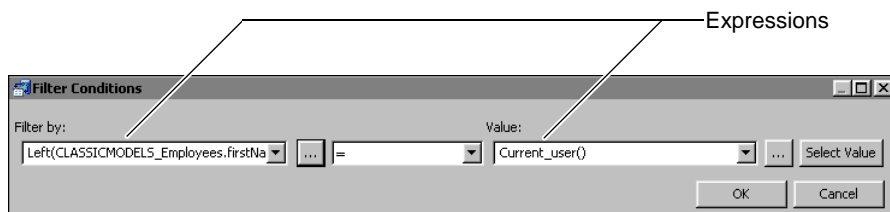


Figure 3-34 Filter Conditions with expressions in Filter by and Value

Creating multiple filter conditions

When you create a filter, you can define one or more filter conditions. Each condition you add narrows the scope of data further. For example, you can create a filter that returns rows where the customer's credit rank is either A or B and

whose open orders total between \$250,000 and \$500,000. Each condition adds complexity to the filter. Design and test filters with multiple conditions carefully. If you create too many filter conditions, the information object returns no data.

Adding a condition

You use the Filters page, shown in Figure 3-28, to create one or more filter conditions. To create a filter condition, you choose New and complete the Filter Conditions dialog, shown in Figure 3-27. When you create multiple filter conditions, the IO Design perspective precedes the second and subsequent conditions with the logical operator AND, for example:

```
SalesOffice LIKE 'San Francisco%' AND
ProductLine LIKE 'Vintage Cars%'
```

This filter returns only data rows that meet both conditions. Sometimes, you want to create a filter to return data rows when either condition is true, or you want to create a more complex filter. To accomplish either task, use the buttons on the right side of the Filters page, shown in Figure 3-35.

If you create more than two filter conditions and you use different logical operators, you can use the parentheses buttons to group conditions to determine the order in which they are evaluated. Display the information object output to verify the results.

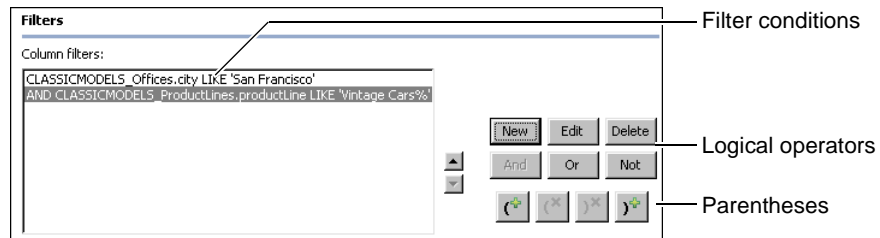


Figure 3-35 Filters page displaying two conditions

Selecting a logical operator

As you add each filter condition, the logical operator AND is inserted between each filter condition. You can change the operator to OR. The AND operator means both filter conditions must be true for a data row to be included in the information object output. The OR operator means only one condition has to be true for a data row to be included. You can also add the NOT operator to either the AND or OR operators to exclude a small set of data.

For example, the following filter conditions return only sales data for classic car items sold by the San Francisco office:

```
SalesOffice LIKE 'San Francisco%' AND
ProductLine LIKE 'Classic Cars%'
```

The following filter conditions return all sales data for the San Francisco and Boston offices:

```
SalesOffice LIKE 'San Francisco%' OR SalesOffice LIKE 'Boston%'
```

The following filter conditions return sales data for all product lines, except classic cars, sold by the San Francisco office:

```
SalesOffice LIKE 'San Francisco%' AND  
NOT (Product Line LIKE 'Classic Cars%')
```



Specifying the evaluation order

The IO Design perspective evaluates filter conditions in the order in which they appear. You can change the order by selecting a filter condition in Filters, shown in Figure 3-28, and moving it up or down using the arrow buttons. Filter conditions that you type in the Actuate SQL text box, shown in Figure 3-29, are preceded by AND and are evaluated last.

If you define more than two conditions, you can use parentheses to group conditions. For example, A AND B OR C is evaluated in that order, so A and B must be true or C must be true for a data row to be included. In A AND (B OR C), B OR C is evaluated first, so A must be true and B or C must be true for a data row to be included.

To illustrate the difference a pair of parentheses makes, compare the following examples.

The following filter contains three conditions and none of the conditions are grouped:

```
Country IN ('Australia', 'France', 'USA') AND  
SalesRepNumber = 1370 OR CreditLimit >= 100000
```

Figure 3-36 shows the first 10 data rows returned by the information object. Although the filter specifies the countries Australia, France, and USA and sales rep 1370, the data rows display data for other countries and sales reps. Without any grouped conditions, the filter includes rows that meet either conditions 1 and 2 or just condition 3.

customerName	country	salesRepEmployeeNumber	creditLimit
Alpha Cognac	France	1370	61100.0
Amica Models & Co.	Italy	1401	113000.0
Anna's Decorations, Ltd	Australia	1611	107800.0
Atelier graphique	France	1370	21000.0
Australian Collectors, Co.	Australia	1611	117300.0
Auto Associés & Cie.	France	1370	77900.0
AV Stores, Co.	UK	1501	136800.0
Collectable Mini Designs Co.	USA	1166	105000.0
Corporate Gift Ideas Co.	USA	1165	105000.0
Corrida Auto Replicas, Ltd	Spain	1702	104600.0

Figure 3-36 Results of a complex filter without parentheses grouping

The following filter contains the same three conditions, but this time the second and third conditions are grouped:

```
Country IN ('Australia', 'France', 'USA') AND
(SalesRepNumber = 1370 OR CreditLimit >= 100000)
```

Figure 3-37 shows the first 10 data rows returned by the information object. The Country IN ('Australia', 'France', 'USA') condition must be true, then either the SalesRepNumber = 1370 condition or the CreditLimit >= 100000 condition is true.

customerName	country	salesRepEmployeeNumber	creditLimit
Alpha Cognac	France	1370	61100.0
Anna's Decorations, Ltd	Australia	1611	107800.0
Atelier graphique	France	1370	21000.0
Australian Collectors, Co.	Australia	1611	117300.0
Auto Associés & Cie.	France	1370	77900.0
Collectable Mini Designs Co.	USA	1166	105000.0
Corporate Gift Ideas Co.	USA	1165	105000.0
Daedalus Designs Imports	France	1370	82900.0
Diecast Classics Inc.	USA	1216	100600.0
Land of Toys Inc.	USA	1323	114900.0

Figure 3-37 Results of a complex filter with parentheses grouping

Changing a condition

You can change any of the conditions in Filters.

How to change a filter condition

- 1 In Filters, shown in Figure 3-28, select the filter condition. Choose Edit.
- 2 In Filter Conditions, shown in Figure 3-27, modify the condition by changing the values in Filter by, Condition, or Value. Choose OK.

Deleting a condition

To delete a filter condition, in Filters, select the condition. Then, choose Delete. Verify that the remaining filter conditions still make sense.

Prompting for filter values

You can use a parameter to prompt an information object user for a filter value. A parameter enables an information object user to restrict the data rows returned by the information object without having to modify the WHERE clause. For example, for an information object that returns sales data by sales office, instead of creating a filter that returns data for a specific office, you can create a parameter called param_SalesOffice to prompt the user to select an office. The WHERE clause is modified as follows:

```
WHERE SalesOffice LIKE :param_SalesOffice
```

You create parameters and define their prompt properties on the Parameters page. Prompt properties include the parameter's default value, a list of values for the user to choose from, and whether the parameter is required or optional. Parameters appear in the Value drop-down list in Filter Conditions with a : (colon) preceding the parameter name, as shown in Figure 3-38.

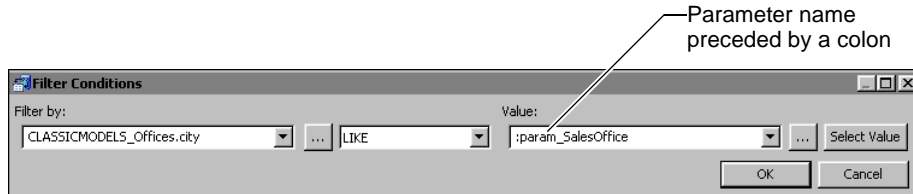


Figure 3-38 Filter Conditions with a parameter in the Value field

Do not use a parameter in a filter condition with the IN operator, for example:

```
Country IN :param_Country
```

Actuate SQL parameters can only accept a single value, but the IN operator takes multiple values. Instead, do one of the following for the appropriate column, for example the Country column:

- Create a predefined filter. The predefined filter becomes a dynamic filter in the report design in BIRT Designer Professional.
- Create a dynamic filter in the report design in BIRT Designer Professional.
- Create a report parameter using the Any Of operator in BIRT Studio.

Grouping data

A GROUP BY clause groups data by column value. For example, consider the following information object:

```
SELECT orderNumber
FROM OrderDetails
```

The first 10 data rows returned by this information object are as follows:

```
orderNumber
10100
10100
10100
10100
10101
10101
10101
10101
10101
10102
10102
```

Each order number appears more than once. For example, order number 10100 appears four times. If you add a GROUP BY clause to the information object, you can group the data by order number so that each order number appears only once:

```
SELECT orderNumber
FROM OrderDetails
GROUP BY orderNumber
```

The first 10 data rows returned by this information object are as follows:

```
orderNumber
10100
10101
10102
10103
10104
10105
10106
10107
10108
10109
```

Typically, you use a GROUP BY clause to perform an aggregation. For example, the following information object returns order numbers and order totals. The Total column is an aggregate column. An aggregate column is a computed column that uses an aggregate function such as AVG, COUNT, MAX, MIN, or SUM.

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

Figure 3-42 shows the first 10 data rows returned by the information object. The data is grouped by order number and the total for each order appears.

Creating a GROUP BY clause

By default, the IO Design perspective creates a GROUP BY clause automatically. If you prefer, you can create a GROUP BY clause manually.

Creating a GROUP BY clause automatically

When an information object's SELECT clause includes an aggregate column and one or more non-aggregate columns, the non-aggregate columns must appear in the GROUP BY clause. If the non-aggregate columns do not appear in the GROUP BY clause, the IO Design perspective displays an error message. For example, consider the following information object:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
```

When you attempt to compile the information object, the error message shown in Figure 3-39 appears in the Problems view.

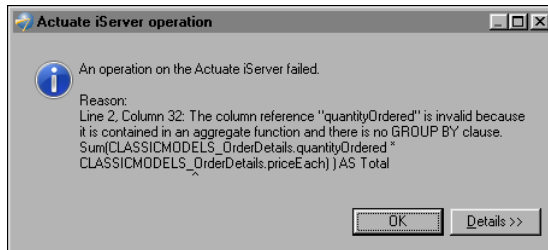


Figure 3-39 Information object requires a GROUP BY clause

To avoid this problem, the IO Design perspective automatically creates a GROUP BY clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

If more than one column appears in the GROUP BY clause, you can change the order of the columns using the up and down arrows in Group By, as shown in Figure 3-40.

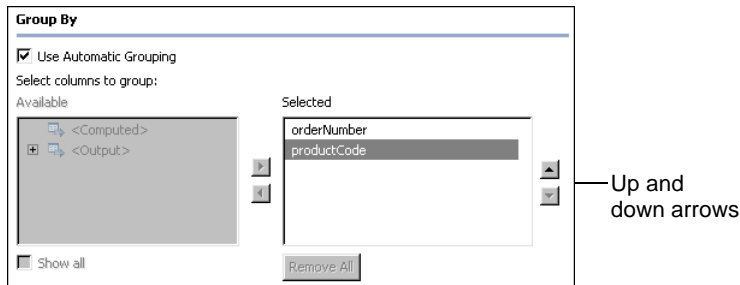


Figure 3-40 Changing the order of GROUP BY columns

Creating a GROUP BY clause manually

If automatic grouping does not generate the desired SQL query, create the GROUP BY clause manually. Create the GROUP BY clause manually if you want to group on a column that does not appear in the SELECT clause, for example:

```
SELECT (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

How to create a GROUP BY clause manually

- 1 In the graphical information object editor, choose Group By.

- 2 In Group By, deselect Use Automatic Grouping.
- 3 In Available, expand the Computed and Output nodes to view the available columns.

By default, the IO Design perspective displays only output columns and non-aggregate computed fields. To group on a column that is not an output column, choose Show all.



- 4 In Available, select the appropriate column, and choose Select. This action moves the column name to Selected, as shown in Figure 3-41.

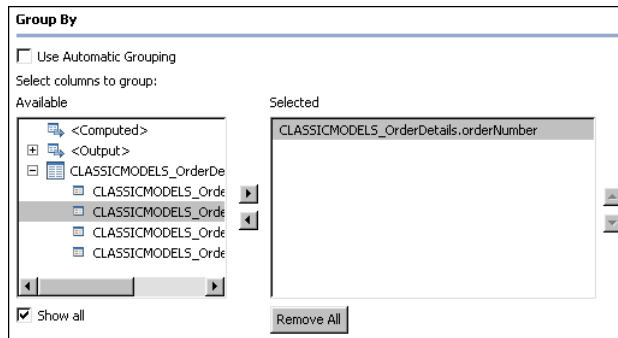


Figure 3-41 Selecting a GROUP BY column



- 5 Repeat the previous step for each GROUP BY column.
- 6 To change the order of the GROUP BY columns, select a column in Selected, and use the up or down arrow.

Removing a column from the GROUP BY clause

By default, the IO Design perspective removes GROUP BY columns automatically. If you disable automatic grouping, you must remove GROUP BY columns manually.

Removing a GROUP BY column automatically

The IO Design perspective automatically removes a column from the GROUP BY clause when:

- You remove the column from the SELECT clause.

For example, consider the following information object:

```
SELECT orderNumber, productCode, (SUM(quantityOrdered *
    priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber, productCode
```

You remove the `productCode` column from the `SELECT` clause. The IO Design perspective automatically removes `productCode` from the `GROUP BY` clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

- You manually add a column to the `GROUP BY` clause that does not appear in the `SELECT` clause and then enable automatic grouping.

For example, consider the following information object:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber, productCode
```

The `productCode` column appears in the `GROUP BY` clause but not in the `SELECT` clause. You enable automatic grouping. Information Object Designer automatically removes `productCode` from the `GROUP BY` clause:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

The IO Design perspective automatically removes the `GROUP BY` clause when:

- You remove all aggregate columns from the `SELECT` clause.

For example, consider the following information object:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

You remove the aggregate column `SUM(quantityOrdered * priceEach)` from the `SELECT` clause. The IO Design perspective automatically removes the `GROUP BY` clause:

```
SELECT orderNumber
FROM OrderDetails
```

- You remove all non-aggregate columns from the `SELECT` clause.

For example, consider the following information object:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
GROUP BY orderNumber
```

You remove the `orderNumber` column from the `SELECT` clause. The IO Design perspective automatically removes the `GROUP BY` clause:

```
SELECT (SUM(quantityOrdered * priceEach)) AS Total
FROM OrderDetails
```

Removing a GROUP BY column manually

If you disable automatic grouping, you must remove GROUP BY columns manually.

How to remove a GROUP BY column manually

- 1 In the graphical information object editor, choose Group By.
- 2 In Group By, complete one of the following tasks:
 - Select the column in Selected, and choose Deselect.
 - To remove all Group By columns, choose Remove All.



Filtering on an aggregate column

If an information object includes a GROUP BY clause, you can restrict the data rows the information object returns by adding a HAVING clause. The HAVING clause places a filter condition on one or more aggregate columns. An aggregate column is a computed column that uses an aggregate function such as AVG, COUNT, MAX, MIN, or SUM, for example SUM(quantityOrdered * priceEach).

For example, the following information object returns order numbers and order totals. The Total column is an aggregate column. The data is grouped by order number and no filter condition is placed on the Total column.

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS  
       Total  
FROM OrderDetails  
GROUP BY orderNumber
```

Figure 3-42 shows the first 10 data rows returned by the information object.

You can add a HAVING clause to this information object to place a filter condition on the Total column. The following information object returns only rows for which the order total is greater than or equal to 50000:

```
SELECT orderNumber, (SUM(quantityOrdered * priceEach)) AS Total  
FROM OrderDetails  
GROUP BY orderNumber  
HAVING SUM(quantityOrdered * priceEach) >= 50000
```

orderNumber	Total
10100	10223.83
10101	10549.01
10102	5494.78
10103	50218.950000000004
10104	40206.2
10105	53959.209999999999
10106	52151.810000000001
10107	22292.620000000003
10108	51001.219999999994
10109	25833.14

Figure 3-42 Data rows returned by information object with GROUP BY clause

Figure 3-43 shows the first 10 data rows returned by the information object.

The procedures for creating filter conditions for aggregate columns are identical to the procedures for creating filter conditions for other columns, except that you use the Having page instead of the Filters page. Filter conditions that you create using the Filters page are evaluated before filter conditions that you create using the Having page. In other words, filter conditions in the WHERE clause are applied before filter conditions in the HAVING clause.

orderNumber	Total
10103	50218.950000000004
10105	53959.209999999999
10106	52151.810000000001
10108	51001.219999999994
10122	50824.659999999996
10126	57131.92
10127	58841.35
10135	55601.840000000004
10142	56052.560000000001
10145	50342.74

Figure 3-43 Data rows returned by information object with GROUP BY and HAVING clauses

Defining parameters

An Actuate SQL parameter is a variable that is used in an information object. The information object user provides a value for this variable in BIRT Studio, Information Object Query Builder, or the IO Design perspective.

For example, the following Actuate SQL query uses the parameters lastname and firstname in the WHERE clause:


```
WITH ( lastname VARCHAR, firstname VARCHAR )
SELECT lname, fname, address, city, state, zip
FROM customerstable
WHERE (lname = :lastname) AND (fname = :firstname)
```

If an Actuate SQL query defines a parameter in a WITH clause but does not use the parameter, the query does not return any rows if no value is provided for the

parameter when the report runs. For example, the following query does not return any rows if no values are provided for the lastname and firstname parameters when the report runs:

```
WITH ( lastname VARCHAR, firstname VARCHAR )  
SELECT lname, fname, address, city, state, zip  
FROM customerstable
```

How to define a parameter

- 1 In the graphical information object editor, choose Parameters.
 - 2 In Parameters, click the top empty line, and complete the following tasks:
 - In Parameter, type the name of the parameter. If a parameter name contains a special character, such as a period (.) or a space, enclose the name in double quotation marks ("").
 - In Data type, select a data type from the drop-down list.
 - In Default value, type the default value:
 - If Default value is a string, enclose the string in single quotation marks, as shown in the following example:
`'New York City'`
 - If Default value is a timestamp, it must be of the following form:
`TIMESTAMP '2001-02-03 12:11:10'`
 - If Default value is a number, use a period (.) as the decimal separator, as shown in the following example:
`123456.78`
- NULL is not a valid parameter value.
- 
- To change the order of the parameters, use the up or down arrow.
 - To use Prompt editor to specify the parameter's prompt properties, choose Prompt editor, as shown in Figure 3-44.
 - To define other parameter properties, such as display name, select the parameter in Parameters, and define the properties in Properties.

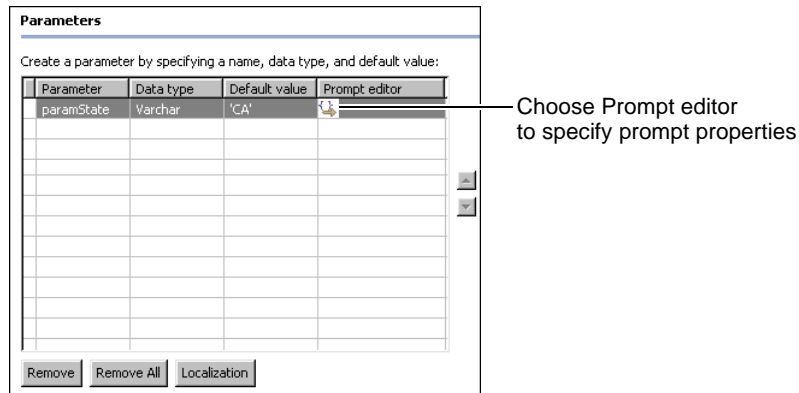


Figure 3-44 Choosing Prompt editor to specify a parameter's prompt properties

How to delete a parameter

- 1 In the graphical information object editor, choose Parameters.
- 2 In Parameters, complete one of the following tasks:
 - To delete an individual parameter, select the parameter, and choose Remove.
 - To delete all parameters, choose Remove All.

Specifying a parameter's prompt properties

Use Prompt editor to specify a parameter prompt's properties, including display control type, list of values, and default value. You can specify the parameter values and, if desired, a corresponding set of display values that the users choose. You create a list of values by typing the values or by typing an Actuate SQL query that retrieves the values.

The query must meet the following requirements:

- The query must retrieve one or two columns from an information object or map, as shown in the following example:

```
SELECT DISTINCT custID, customName
FROM "MyInformationObject.iob"
ORDER BY 2
```

The first column contains the parameter values. The second column contains the values that are displayed to the user. The information object or map must reside in the same volume as the IOB used as a data source in the report design. If you use a relative path to reference the information object or map, BIRT Designer Professional interprets the path as relative to the IOB used as a

data source. If the information object or map defines a parameter, you must provide a value for the parameter, as shown in the following example:

```
SELECT DISTINCT custID, customName
FROM "MyInformationObject.iob" ['CA']
ORDER BY 2
```

- The first column's data type must match the parameter's data type.
- The query must not contain a WITH clause.

The IO Design perspective does not validate the query. The values returned by the query appear when a user specifies a value for the parameter in BIRT Studio. The values do not appear, however, when you specify a value for the parameter in the IO Design perspective or when a report developer specifies a value for the parameter in Information Object Query Builder.

How to specify a parameter prompt's properties



- 1 Locate the appropriate parameter in Parameters and choose Prompt editor.
- 2 On Prompt editor, in Show as, select the method of prompting the user, as shown in Figure 3-45. If you use a type of display other than a text box, you can specify a list of values for the user to choose.

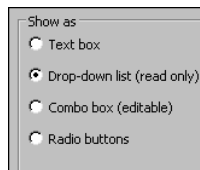


Figure 3-45 Selecting the method of prompting the user

You can create a list of values by typing the values and, optionally, the display names, as shown in Figure 3-46. If you do not provide display names, the values are displayed to the user.

You can create an Actuate SQL query that retrieves the values or both the values and the corresponding display names. If the query has two columns, the values in the second column are used as the display names. To use a query to create the list of values, select Dynamic list of values, as shown in Figure 3-47, and type the query.

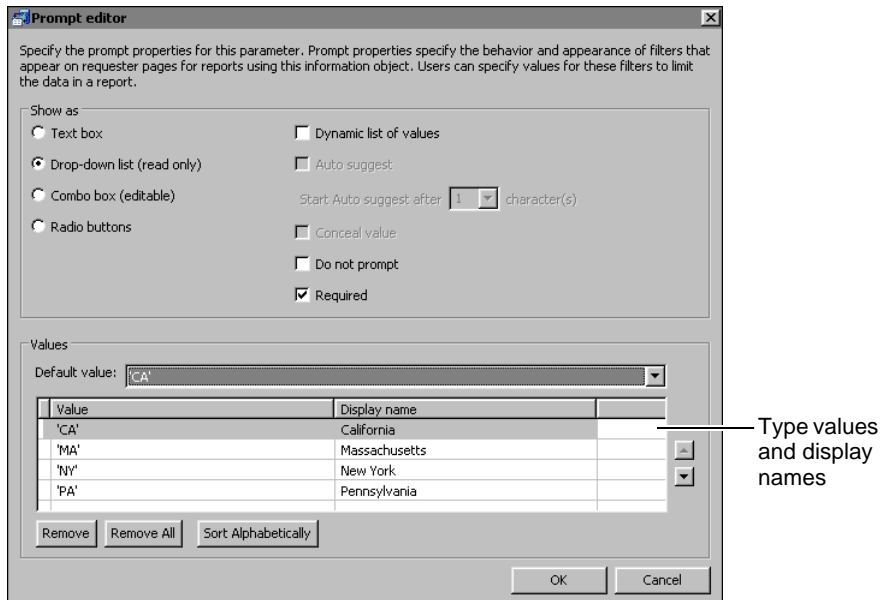


Figure 3-46 Typing a list of values and display names

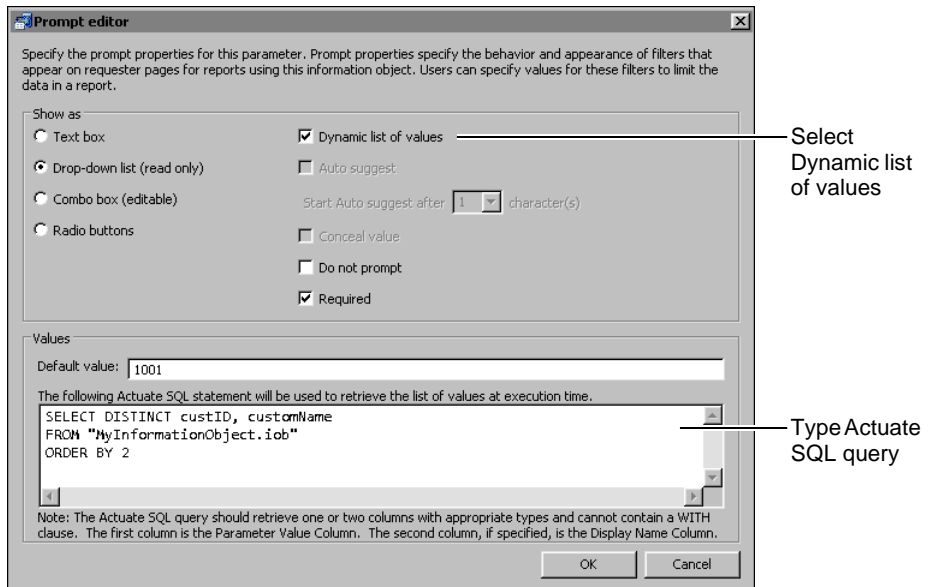


Figure 3-47 Specifying an Actuate SQL query to provide a dynamic list of values

If you select Combo box (editable), Dynamic list of values, and Auto suggest, a drop-down list appears after the information object user types the number of characters specified in Start Auto suggest after N character(s). The list contains values that begin with the characters the user typed. For example, if the user typed 'Atel' and N=4, the list contains the value 'Atelier graphique'. In this case, the query that retrieves the values must select two columns, a value column and a display name column.

- 3 In Default value, you specify the default value.
- 4 You also can specify values for the following additional properties:
 - Conceal value
 - Do not prompt
 - Required

When you finish specifying the property values for the prompt, choose OK.

Setting parameter properties

Table 3-4 lists parameter properties and provides a description of each property.

Table 3-4 Parameter properties

Parameter property	Can set?	Description
Conceal Value	Yes, in Prompt editor	Visibility of the value that the user provides for this parameter. To conceal the value, set to True. To display the value, set to False. This parameter property applies only to parameters with the varchar data type and the text box display type.
Data Type	Yes, on the Parameters tab	Parameter's data type.
Default Value	Yes, in Prompt editor	Parameter's default value. If a parameter does not have a default value, and the Required property is set to False, the parameter takes one of the following values if the user does not provide a value: <ul style="list-style-type: none"> ■ 0 if the parameter is of type decimal, double, or integer. ■ Empty string if the parameter is in the varchar data type. ■ Current date and time if the parameter is in the timestamp data type.

(continues)

Table 3-4 Parameter properties (continued)

Parameter property	Can set?	Description
Description	Not used	Not used.
Description Key	Not used	Not used.
Display Control Type	Yes, in Prompt editor	Control type for the parameter. The options are text box, read-only drop-down list, editable drop-down list, or radio buttons.
Display Format	Not used	Not used.
Display Length	Not used	Not used.
Display Name	Yes	Parameter prompt in BIRT Studio.
Display Name Key	Yes	Key for Display Name property in localization properties file.
Do Not Prompt	Yes, in Prompt editor	Visibility of the parameter to the user. To hide the parameter, set to True. To display the parameter, set to False.
Heading	Not used	Not used.
Heading Key	Not used	Not used.
Help Text	Not used	Not used.
Help Text Key	Not used	Not used.
Horizontal Alignment	Not used	Not used.
Name	Yes, on the Parameters tab	Parameter name.
Parameter Mode	Yes	Setting for parameters in stored procedures and ODA data source queries to specify the input or output type of the parameter. The options are Input, Output, InputAndOutput, or ReturnValue. ReturnValue is used only for stored procedures and is equivalent to Output.
Required	Yes, in Prompt editor	Indicator of whether the parameter is required. To require a value for this parameter, set to True. Otherwise, set to False.
Size	Yes	The size of the parameter if the parameter data type is varchar. Otherwise, not used. Must be set if size is greater than 1300.

Setting source parameters

A source parameter is a parameter that is defined in a map or information object from which you are building another information object.

You can set a source parameter to one of the following types of values:

- A single scalar value
- A local parameter in the information object that you are creating

You cannot set a source parameter to a column reference, such as `ORDERS.ORDERID`, or an Actuate SQL expression.

When you set a source parameter to a local parameter, you can indicate that the local parameter inherits the values of its prompt properties from the source parameter. The available prompt properties are Conceal Value, Default Value, Display Control Type, Do Not Prompt, and Required. If you specify that the local parameter inherits its prompt property values from the source parameter, and prompt property values for the source parameter change, the changes are propagated to the local parameter. For example, if the display control type for the source parameter changes from text box to read-only drop-down list, the display control type for the local parameter also changes from text box to read-only drop-down list.

If you change a prompt property value for a local parameter, its prompt property values are no longer inherited from the source parameter. For example, if you change the display control type for the local parameter to editable drop-down list, and the display control type for the source parameter later changes to text box, the change is not propagated to the local parameter. To reinstate inheritance, choose Reset in Prompt editor. Choosing Reset returns all property values in the local parameter to inherited values, and the local parameter inherits any future changes to property values in the source parameter.

To set source parameters, use the Parameters page. To define a local parameter and set a source parameter to the local parameter in one step, drag the source parameter from Source parameter, and drop it in Parameter, as shown in Figure 3-48.

How to set a source parameter

- 1 In the graphical information object editor, choose Parameters.
- 2 In Parameters, complete the following tasks:
 - In Source parameter, select the appropriate parameter.
 - In Value, complete one of the following tasks:
 - Choose a parameter from the drop-down list. The drop-down list contains the local parameters for the information object you are building.

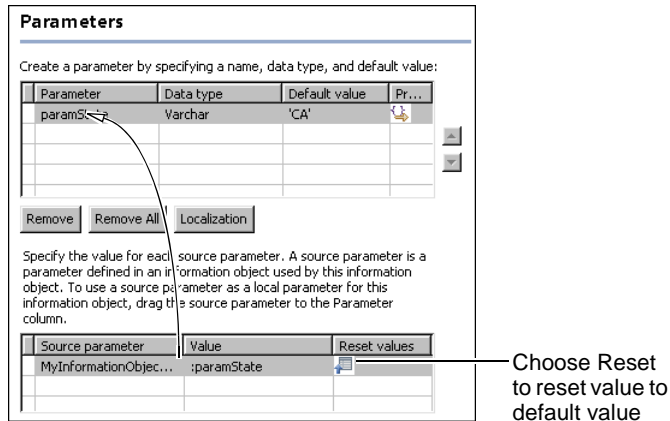


Figure 3-48 Setting a source parameter for a new, local parameter

- ❑ Type a value, as shown in Figure 3-49:
- ❑ If Value is a string, enclose the string in single quotation marks, as shown in the following example:
`'New York City'`
- ❑ If Value is a timestamp, it must be in the following form:
`TIMESTAMP '2001-02-03 12:11:10'`
- ❑ If Value is a number, use a period (.) as the decimal separator, as shown in the following example:
`123456.78`
- ❑ If Value is a parameter, precede the parameter name with a colon (:).

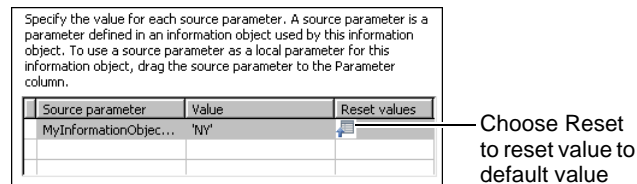


Figure 3-49 Providing a value for a source parameter

Synchronizing source parameters



You must synchronize source parameters when parameters in a source map or information object are added, removed, or reordered, or their data types or other properties change. To synchronize source parameters, choose Compile IO in the graphical information object editor, as shown in Figure 3-50. Synchronizing source parameters refreshes the list of source parameters on the Parameters page.

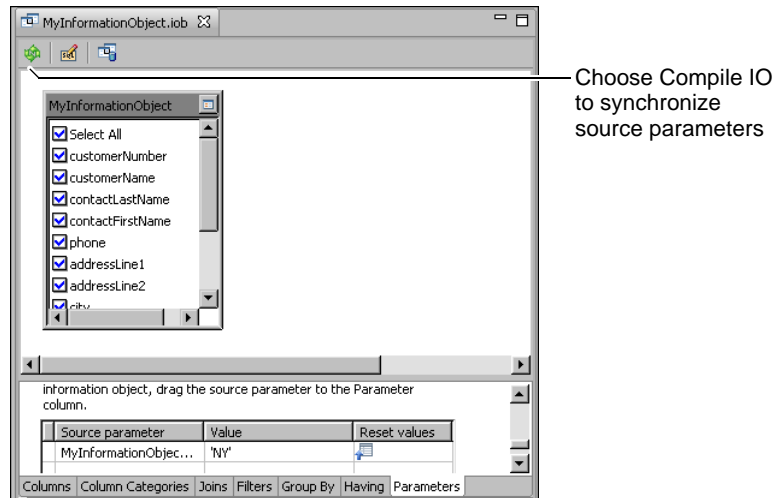


Figure 3-50 Synchronizing source parameters

Creating a textual information object

Use the Actuate SQL text editor if either of the following conditions is true:

- The graphical information object editor does not generate the desired Actuate SQL query, so you must edit the query. For example, if the query includes OR or UNION, you must use the Actuate SQL text editor to edit the query.
- You want to type or paste an Actuate SQL query instead of creating it graphically.

If you save a query in the Actuate SQL text editor, you cannot modify the query in the graphical information object editor.

To display the Actuate SQL text editor, complete one of the following tasks:



- In the graphical information object editor, choose SQL editor, as shown in Figure 3-51.
- In SQL Preview, choose Edit SQL.
- On New Information Object, select Edit in SQL text editor, as shown in Figure 3-52.

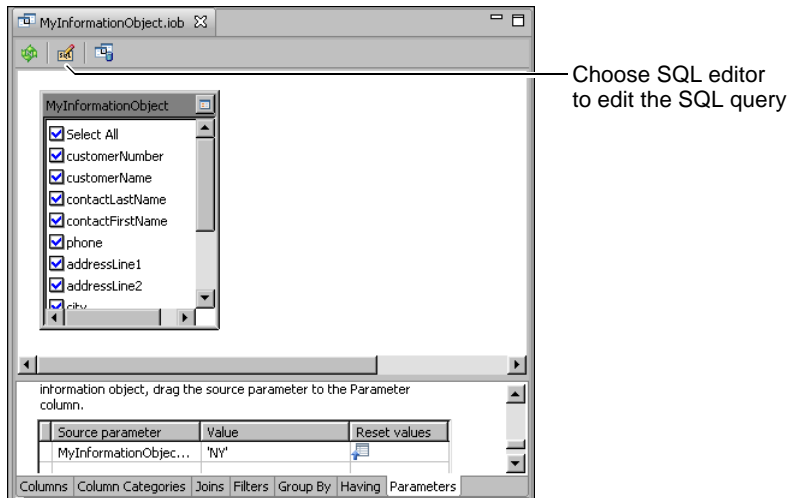


Figure 3-51 Choosing SQL editor to edit an Actuate SQL query

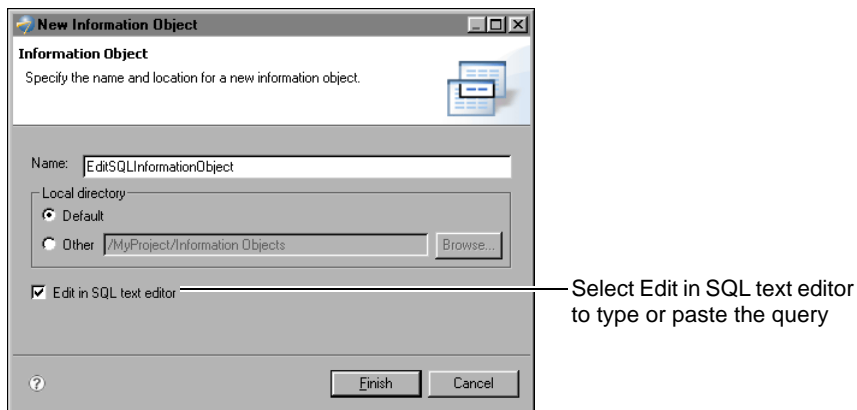


Figure 3-52 Choosing to provide the Actuate SQL query in the SQL text editor

You edit the query in the upper pane of the Actuate SQL text editor, as shown in Figure 3-53. The lower pane displays output columns or parameters.

When you edit a query in the SQL text editor, do not use table and column aliases that are identical except for case. For example, do not use both status and STATUS as column aliases.

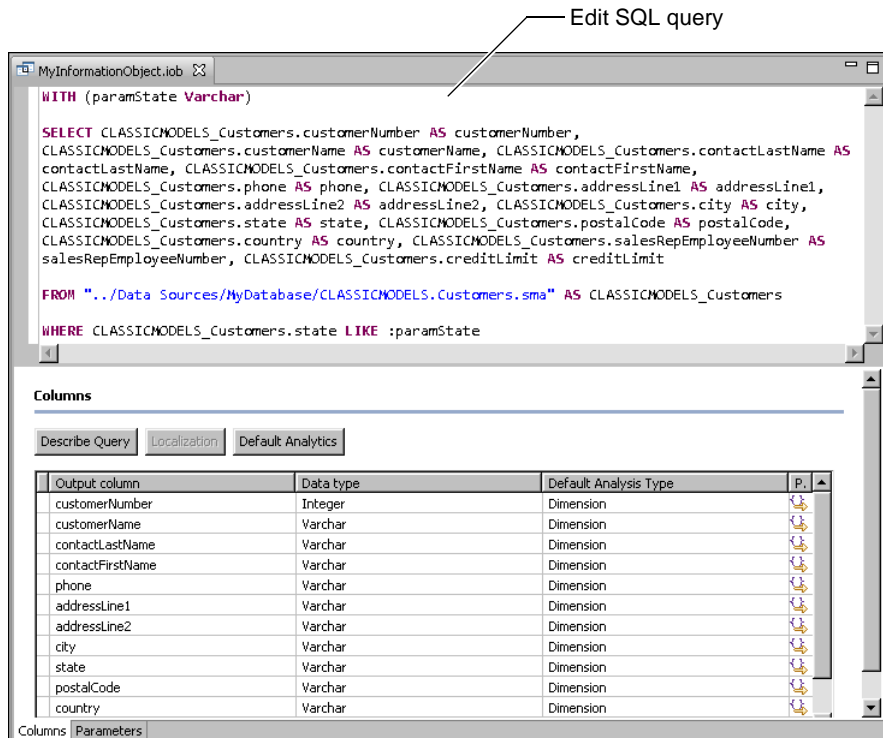


Figure 3-53 Editing the SQL query in the Actuate SQL text editor

The following rules also apply:

- Do not include an ORDER BY clause in the query.
- Paths that do not begin with a forward slash (/) are relative to the IOB file, as shown in the following example:

```
../Data Sources/MyDatabase/dbo.customers.sma
```
- Absolute paths must begin with a forward slash. Using absolute paths is not recommended.

Figure 3-53 shows the Actuate SQL text editor.

Displaying output columns

In SQL Text Editor—Columns, to display the query's output columns and the data type for each column, choose Describe Query, as shown in Figure 3-54.

To create a filter on a column, set the column's Filter property to Predefined, and specify the filter's prompt properties.

To specify other column property values, select the column, and specify the property values in Properties.

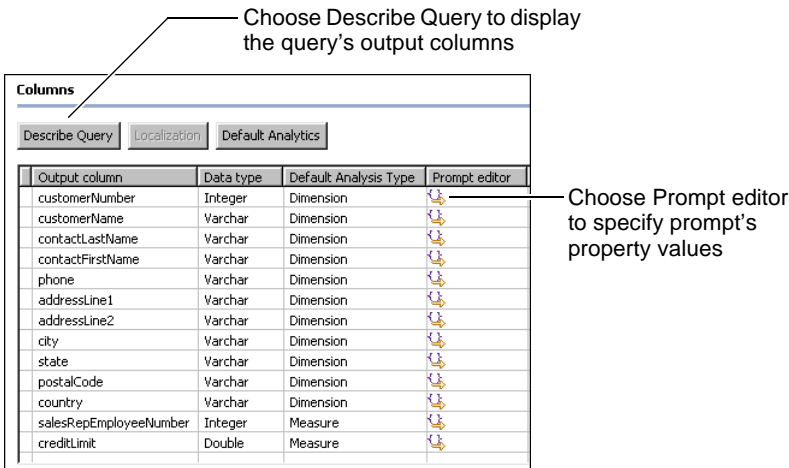


Figure 3-54 Using Describe Query to display the query's output columns

Displaying parameters

On SQL Text Editor—Parameters, choose Describe Query to display the query's parameters and the data type for each parameter. You can type a default value for a parameter in Default value, as shown in Figure 3-55.

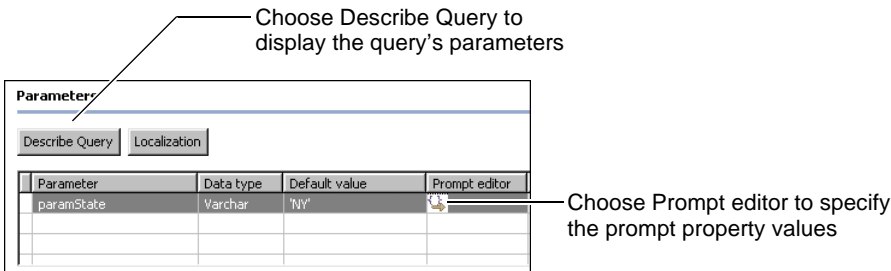


Figure 3-55 Using Describe Query to display the query's parameters

You can choose Prompt editor to set the prompt property values.

Displaying and testing information object output

To preview output, you can display information object output in Data Preview.

How to display and test information object output

- 1 Choose Data Preview.
- 2 In Data Preview, choose Refresh. As shown in Figure 3-56, Parameter Values appears if the information object defines parameters.

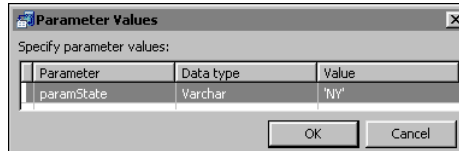


Figure 3-56 Specifying parameter values

- 3 On Parameter Values, type the parameter values. A parameter value must be a single value, not a list of values. When you finish, choose OK, and information object output appears, as shown in Figure 3-57.

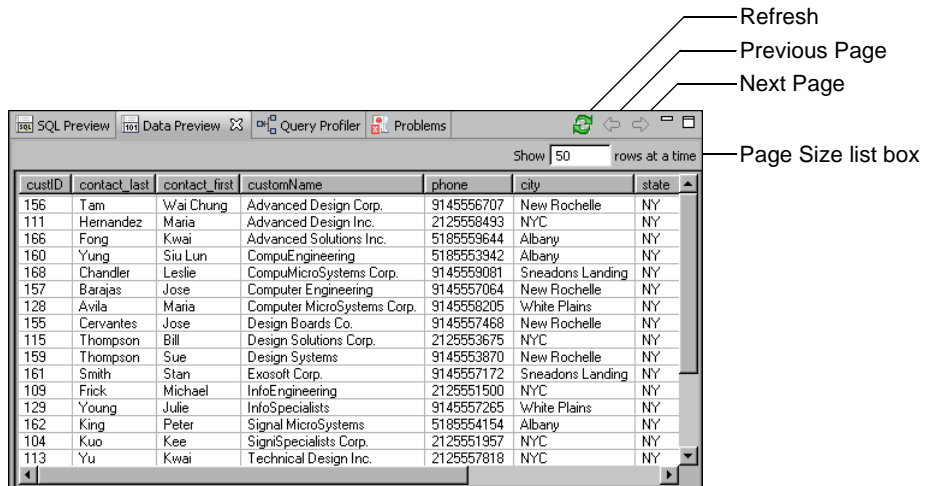


Figure 3-57 Viewing the information object's output

- 4 Use the scroll bars to view all columns and displayed rows. Use the Page Size list box to change the number of rows displayed on each page. Use the Next Page and Previous Page icons to navigate through the data preview one page at a time.

Displaying a data source query

When the Actuate SQL compiler compiles an information object, the compiler creates one query for each data source. The query is written in the data source's

native query language. You can display the query that is sent to a data source, as well as the following information about the query:

- The number of rows in the query's output
- The amount of time it takes to execute the query in milliseconds
- The name and path for the data connection definition file

How to display a data source query and information about the query

- 1 Choose Query Profiler.
- 2 In Query Profiler, choose Start.
- 3 On Parameter Values, type the parameter values, as shown in Figure 3-58. Choose OK.

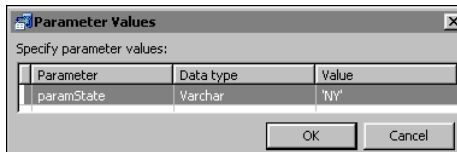


Figure 3-58 Providing the parameter values

A schematic representation of the query execution plan appears in the upper pane of Query Profiler.

- 4 Select a SQL or ODA node to display the data source query.

The data source query appears in the lower pane, as shown in Figure 3-59.

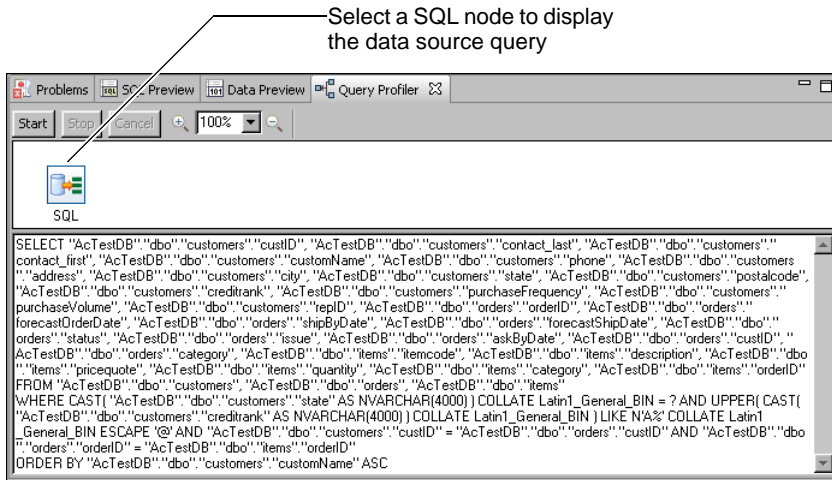


Figure 3-59 Displaying the data source query

- 5 Hover the cursor over the SQL or ODA node to display information about the query. Figure 3-60 shows an example of the information displayed for a query.

SQL	
Input Tuples Exhausted	0
Output Tuples	157
Cumulative Time	12340
Exhaust Calls	0
Reset Calls	0
Pages Allocated	0
Physical Store	/MyProject/Data Sources/MyDatabase/_MyDatabase.dcd
Press F2 for focus.	

Number of output rows
Query execution time
Connection definition file

Figure 3-60 Display of information about a query

Understanding query execution plan operators

If an information object retrieves data from only one data source, the query execution plan consists of a single node: a SQL or ODA node that displays the query sent to the data source. If an information object retrieves data from more than one data source, the query execution plan consists of several nodes, including the following:

- One SQL or ODA node for each data source query
- Nodes that represent the joins between data source queries

For example, the query execution plan in Figure 3-61 displays two SQL nodes and a join node. The SQL nodes represent the native SQL queries that are sent to two different databases. The join node indicates that the join uses the Nested Loop join algorithm.

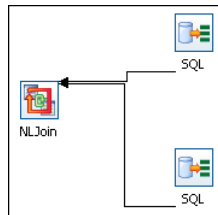


Figure 3-61 An execution plan for a query using a join of two SQL queries

You can experiment with different join algorithms and determine which one performs best by doing the following:

- Displaying the execution time for each node
- Calculating the total execution time for the information object by adding the execution times for the individual nodes

Understanding node operators

Node operators process the output of other operators to produce rows.

Augment

Augment adds the result of an expression as a new column to each row in the target relation.

Box

Box appends a nested relation containing a single row whose columns are a subset of the columns in the input row. The boxed columns are removed from the input row. Both the inner and outer relations are projections of the input relation. In other words, columns can be rearranged.

CallExecutionUnit

CallExecutionUnit executes the equivalent of a subroutine call. CallExecutionUnit augments each incoming row with an iterator that iterates over the product of a dependent execution unit. The dependent execution unit can be parameterized with data from the input relation or from the calling execution unit. The parameters are either scalar types or iterators over nested relations.

DependentJoin

DependentJoin joins an input relation with the product of a dependent execution unit. The dependent execution unit can be parameterized with data from the input relation or from the calling execution unit. The join itself is unconditional. Because the dependent execution unit is parameterized, however, the contents of the dependent relation can be different for every input row. The join is either nesting or flat:

- Nesting
Each input row is augmented by a single iterator column that contains the dependent execution unit's output rows.
- Flat
Each input row is augmented by all the columns of the dependent execution unit's output rows.

Dup

Dup creates a second independent iterator over a materialized relation. In effect, Dup duplicates the relation. The duplicated relation can come from any row in the target path. It is possible to take a relation that is nested in an outer row and nest a copy of it into every row in a deeper relation. The output row is a copy of the input row with the additional iterator on the end.

Materialize

Materialize caches an entire relation, including its descendants in the executor's memory. Iterators over the relation and its descendants can then be reset repeatedly. This is necessary if the relation is duplicated or if the relation is to be aggregated over as well as detailed. Materialize does not change any data. Materialize simply accumulates rows from its input and does not release the rows until they are no longer needed. For more information about controlling how a relation is materialized, see *Configuring BIRT iHub*.

MergeJoin

MergeJoin performs an equijoin between left and right relations. Both relations must be ordered by join condition. The join can be nesting or flat:

- Nesting
Each left-side row is augmented by a single iterator column that contains the selected right-side rows.
- Flat
Each left-side row is augmented by all the columns of the selected right-side rows.

Move

Move copies an iterator. The copied iterator can come from any row in the target path. It is possible to take a relation that is nested in an outer row and nest a copy of it into every row in a deeper relation. Unlike Dup, the copied iterator shares state with the original one. It is illegal to iterate over one iterator in an inner loop while maintaining the context of the other in an outer loop.

MultiAugment

MultiAugment augments a row with an iterator column. This iterator produces a relation whose contents are the result of evaluating an expression. One row is produced for each expression, and then the inner iterator is exhausted until the outer iterator is advanced again. The operator has the effect of augmenting a row with a sequence.

Nest

Nest groups adjacent rows that match an equality constraint. The group of rows is then replaced by a single outer row with a nested relation containing the same number of rows that were in the group. The columns in the input rows that are projected into the inner relation are independent of the columns that are compared to determine grouping.

NestedLoopJoin

NestedLoopJoin joins two input relations by evaluating an expression for every pair of rows. NestedLoopJoin materializes the right-side relation and then, for every left-side row, iterates over all rows in the right-side materialization, evaluating the expression each time. If the result of the expression is True, the rows match.

Project

Project reorders or removes columns within a relation.

Select

Select removes rows from the target relation. For every input row, an expression is evaluated. If the result of the expression is False, the row is rejected. When Select is advanced, it advances its input iterator repeatedly until it finds a row that is accepted. Then it passes that row on.

Sort

Sort sorts a relation using one or more sort keys. The relation is first materialized.

Union

Union combines the rows from two relations without eliminating any duplicates. The left-side rows are output first, and then the right-side rows.

Understanding leaf operators

Leaf operators produce rows by communicating with a data source.

FakeData

FakeData generates a flat relation using synthesized data. FakeData is used for testing.

FakeFileData

FakeFileData reproduces a flat relation stored in tab-delimited format in a text file. FakeFileData is used for testing.

IteratorAsLeaf

IteratorAsLeaf is used in dependent execution units. IteratorAsLeaf takes an iterator that is specified in the execution unit's parameter list and treats the iterator as a leaf operator. IteratorAsLeaf is used when a binary operator, for example a join, is applied to sibling relations within the same containing relation.

NoOp

NoOp returns an empty relation. NoOp is used by the compiler to represent queries when the compiler determines that there are no results.

ODA

For more information about the ODA operator, see “Displaying a data source query,” earlier in this chapter.

SortedOuterUnion

SortedOuterUnion allows multiple SELECT statements to be evaluated by a data source in a single round-trip.

SQL

For more information about the SQL operator, see “Displaying a data source query,” earlier in this chapter.

Storing a query plan with an information object

An information object can be used as a data source in BIRT Studio. The information object’s query plan is compiled the first time a BIRT Studio user drags-and-drops a column from the Available Data pane to the report display area. If the information object is built from a large number of maps or information objects, the first drag-and-drop operation may take a long time. To avoid this problem, you can store a precompiled query plan with the information object.

Storing a precompiled query plan with an information object increases the size of the information object. For this reason, do not store a precompiled query plan with an information object unless the first drag-and-drop operation in BIRT Studio is unacceptably slow.

Store a precompiled query plan with an information object when either of the following conditions is true:

- The information object is built from more than 100 tables consisting of more than 1000 columns.
- The information object has more than four levels in its hierarchy.

If an information object query’s compile time is very large, it may be advisable to store a precompiled query plan with the information object.

How to display an information object query’s compile time

- 1 Choose Query Profiler.
- 2 In Query Profiler, choose Start.

- 3 On Parameter Values, type the parameter values, as shown in Figure 3-58. Choose OK.

A schematic representation of the query execution plan appears in the upper pane of Query Profiler, as well as Profiler Statistics such as compile time. In Figure 3-62, Profiler Statistics displays a compile time of three seconds.

Profiler Statistics
Compile Time : 3 s
Execution Time : 31 s

Figure 3-62 Profiler Statistics

Saving an information object's query plan

To store a query plan with an information object, set the Use Precompiled Query Plan at runtime property to True and save the information object. The Precompiled query plan saved on property displays the date and time at which the query plan is saved.

How to save an information object's query plan

- 1 Open the information object in the graphical or textual query editor.
- 2 Click in the white space in the upper pane of the query editor.

Figure 3-63 shows the upper pane of the graphical information object query editor.

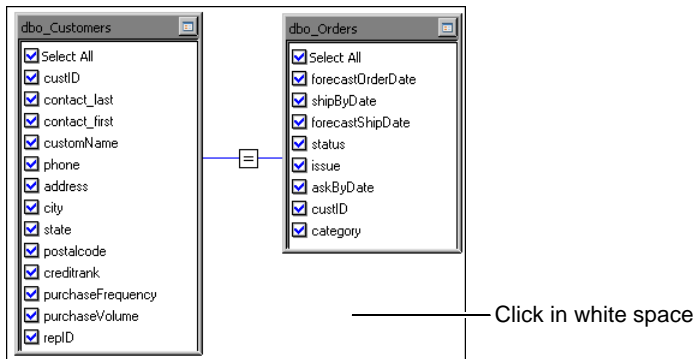


Figure 3-63 Upper pane of the graphical information object editor

- 3 In Properties, set Use Precompiled Query Plan at runtime to True.
- 4 Choose File→Save.

Saving query plans for source and dependent information objects

To store query plans for a source information object and its dependent information objects, set the Use Precompiled Query Plan at runtime property to True for each information object and save the information objects. If you modify the source information object, you can refresh the query plans in one step by compiling the information objects.

How to refresh the query plans for source and dependent information objects

- 1 In Navigator, select the source information object.
- 2 Choose File→Information Objects→Compile IO and dependents.

If any of the information objects are open, the prompt shown in Figure 3-64 appears. Choose Yes.

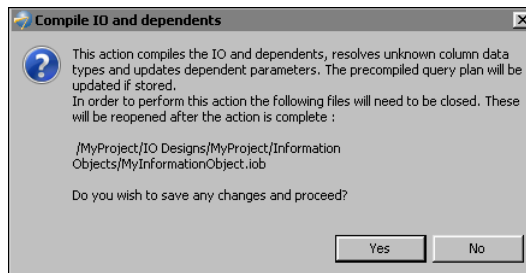


Figure 3-64 File close prompt

The Progress dialog may appear, as shown in Figure 3-65. To run the compile operation in the background, choose Run in Background.

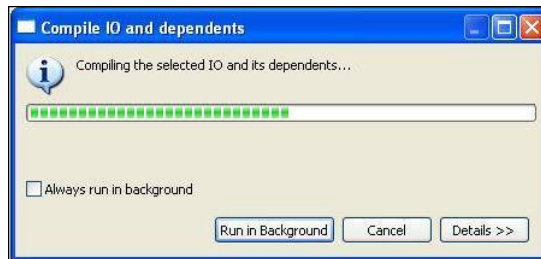


Figure 3-65 Progress dialog



The progress indicator appears in the lower right corner of the IO design perspective.

- 3 To display the Progress view, choose the progress indicator.

The Progress view, shown in Figure 3-66, displays the progress of the compile operation.

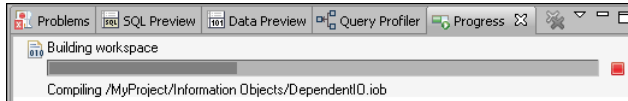


Figure 3-66 Progress view

Deleting an information object's query plan

To delete an information object's query plan, set Use Precompiled Query Plan at runtime to False and save the information object.

Localizing an information object

If an information object is used in more than one locale, you should provide translated strings for the following column and parameter properties:

- Description
- Display Name
- Heading
- Help Text

The translated strings are used when:

- The information object is used as a data source in a report design in BIRT Studio.
- A report with an information object data source is viewed in the Viewer or Interactive Viewer.

For each locale, you create a properties file that contains a key and a translated string for each column or parameter and property. For example, the following entries contain keys and French strings for the customerNumber and customerName columns:

```
customerNumber_DescriptionKey="Numéro de client"
customerNumber_DisplayNameKey="Numéro de client"
customerNumber_HeadingKey="Numéro de client"
customerNumber_HelpTextKey="Numéro de client"
customerName_DescriptionKey="Nom du client"
customerName_DisplayNameKey="Nom du client"
customerName_HeadingKey="Nom du client"
customerName_HelpTextKey="Nom du client"
```

Localization properties files reside in the project's Localization folder. The file name is constructed from the project name and the locale code. In Figure 3-67, the Localization folder in the project MyLocalizedProject contains properties files for

Spanish and French. Localization properties files are shared among all information objects in a project.

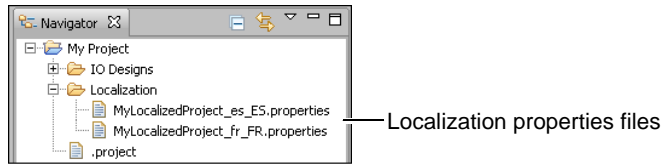


Figure 3-67 Localization properties files

The locale code consists of a two-letter language code, an underscore, and a two-letter country code. For example, the locale code for French (France) is `fr_FR`. Use the following URL to display a list of languages and the corresponding ISO 639 language codes:

<http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt>

Use the following URL to display a list of countries and the corresponding ISO 3166 country codes:

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

Localization properties files are published to the Encyclopedia volume's resource folder. When you publish localization properties files, the folder structure in the project is preserved. For projects created with Release 11 Service Pack 3 and earlier, localization properties files are published to the folder you specify in the Publish Information Objects dialog. To determine whether localization properties files are published as resources or not, select the appropriate project in Navigator and choose **File**→**Properties**→**Actuate BIRT**, as shown in Figure 3-68.

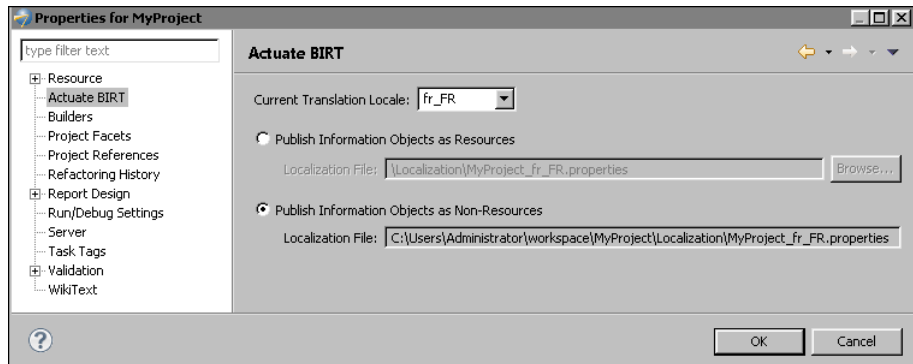


Figure 3-68 Publishing localization properties files as non-resources

How to localize an information object

- 1 Choose the translation locale for the project.
 - 1 Select the project in Navigator.
 - 2 Choose File>Properties>Actuate BIRT.
 - 3 In Current Translation Locale, choose the translation locale from the drop-down list.
 - 4 If the default location for the localization properties file is not correct, browse to the correct location. The path is relative to the resources root. Choose OK.

In Figure 3-69, the translation locale is French (France).

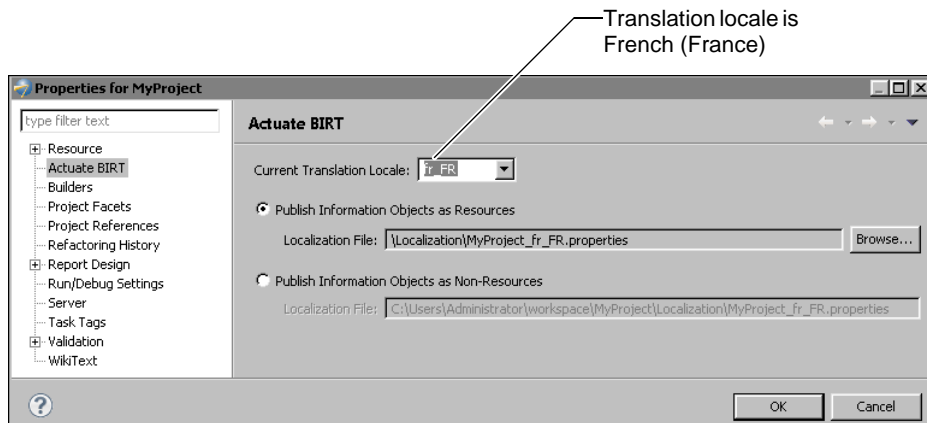


Figure 3-69 Specifying the translation locale

- 2 Open the information object in the graphical or textual information object editor.
- 3 Choose Columns or Parameters.
- 4 In Columns or Parameters, select the column or parameter whose properties you want to localize.

In Figure 3-70, the customerName column is selected in the graphical information object editor.

- 5 Choose Localization, as shown in Figure 3-70.

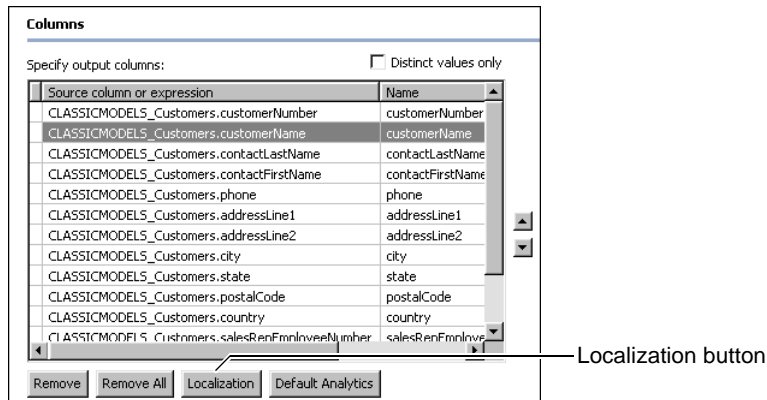


Figure 3-70 Localization button in Columns

- 6 In Localization, choose Browse for the appropriate key.

For example, to localize the Display Name property, choose the Browse button to the right of the Display Name Text Key field.

- 7 In Select Key, in Quick Add:

- 1 In Key, type the translation key.
- 2 In Value, type the translation string.
- 3 Choose Add.

- 8 Select the Key-Value pair. Choose OK.

In Figure 3-71, the Key-Value pair customerName_DisplayNameKey-"Nom du client" appears.

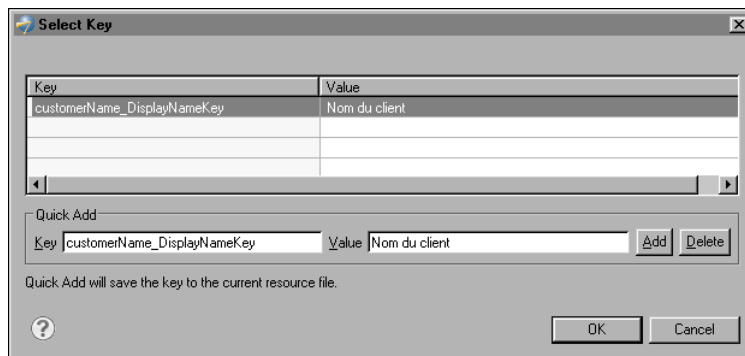


Figure 3-71 Creating a Key-Value pair

- 9 Repeat steps 6 through 8 for the remaining translation keys. Choose OK.

Figure 3-72 shows the translation keys for the customerName column.

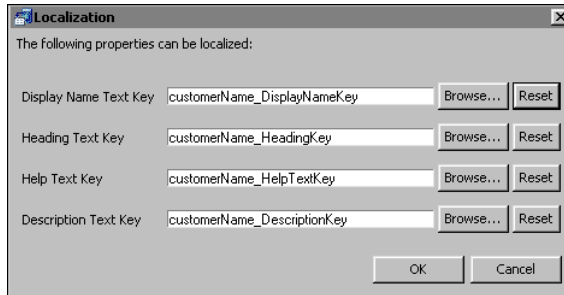


Figure 3-72 Translation keys for customerName

- 10 Repeat steps 4 through 9 for the remaining columns or parameters.
- 11 Test the localized information object with the appropriate locale. For example:
- 1 Log in to Information Console with the French (France) locale.
 - 2 Use the localized information object as a data source in a report design in BIRT Studio.
 - 3 Verify that the French display name, help text, and heading appear.

How to modify a translation string

- 1 In Localization, shown in Figure 3-72, choose Browse for the appropriate key.
- 2 In Select Key, shown in Figure 3-71, select the appropriate Key-Value pair.
- 3 In Value, modify the translation string. Choose Add.
- 4 Select the Key-Value pair. Choose OK.

How to modify a translation key

- 1 In Localization, shown in Figure 3-72, choose Browse for the appropriate key.
- 2 In Select Key, shown in Figure 3-71, select the appropriate Key-Value pair. Choose Delete.
- 3 In Key, modify the translation key. Choose Add.
- 4 Select the Key-Value pair. Choose OK.

How to disable localization for a column or parameter property

In Localization, shown in Figure 3-72, choose Reset for the appropriate key. The information object displays the untranslated string for the column or parameter property, for example Display Name, instead of the translated string.

How to restore localization for a column or parameter property

- 1** In Localization, shown in Figure 3-72, choose Browse for the appropriate key.
- 2** In Select Key, shown in Figure 3-71, select the appropriate Key-Value pair. Choose OK.

4

Building and publishing a project

This chapter contains the following topics:

- Building a project
- Propagating column and parameter property values
- Publishing a project
- Downloading files from an Encyclopedia volume

Building a project

Building a project compiles the resources in the project. By default, the IO Design perspective builds a project whenever you modify and save a resource. If you disable automatic building, choose Project>Build Project to build a project.

Build error messages appear in Problems, as shown in Figure 4-1. To locate an error, double-click the error message. To filter the error messages, choose View Menu>Show and select the appropriate menu item. For example, to show all errors, choose View Menu>Show>All Errors.

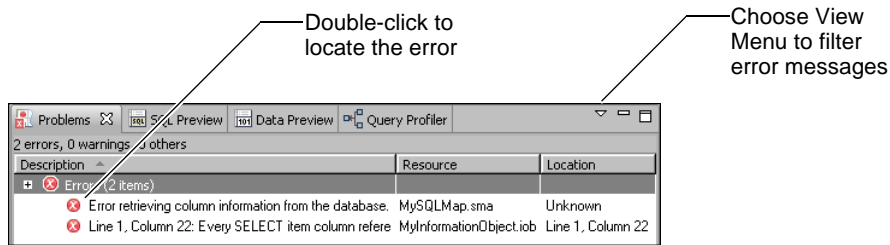


Figure 4-1 The Problems view



If the description is truncated, select the error message and choose Ellipsis to display the complete description.

For more information about building a project and displaying error and warning messages, see the *Workbench User Guide* in the IO Design perspective online help.

Propagating column and parameter property values

Before you publish a project, make sure the information objects in the project inherit the correct column and parameter property values by selecting the project in Navigator and choosing Project>Propagate Property Values. Propagation of property values may take several minutes. A map's or information object's column order and category order are not propagated to its dependent information objects.

Publishing a project

When you are ready to test or deploy a project, you must publish the project to the appropriate Encyclopedia volume. You specify the Encyclopedia volume by selecting an iHub profile. The iHub profile specifies the iHub, port number, volume, user name, and password.

Report design files and information object files are published to different locations:

- Report design files (RPTDESIGNs) are published to the folder you specify in the Publish to Server dialog. When you publish report design files, you have the option to preserve the folder structure in the project.
- Information object files such as DCDs, SMAs, and IOBs are published to the Encyclopedia volume's resource folder. When you publish information object files, the folder structure in the project is preserved.

You can publish an entire project or individual files. You can replace the latest version of a file or create a new version.

How to create an iHub profile

- 1 In Server Explorer, right-click and choose New Server Profile.
- 2 In New Server Profile:
 - In Profile Type, choose Server from the drop-down list.
 - In Profile name, type the name of the iHub profile, for example MyProfile.
 - In Server, type the name or IP address of the computer on which the Message Distribution Service is running.
 - In Port number, type the number of the port on which the Message Distribution Service listens for requests.

The port number appears in the Port for the Message distribution service endpoint on the Server Configuration Templates—Properties—Message Distribution Service—Process Management—Communication page of Configuration Console. The default port number is 8000.
 - In Volume, choose the Encyclopedia volume in which you want to store Actuate BIRT project files.
 - In User name, type your Encyclopedia volume user name. You type this user name when you log in to Management Console.
 - In Password, type your Encyclopedia volume password. You type this password when you log in to Management Console.

If you select Remember Password, the IO Design perspective stores the password in encrypted format in acserverprofile.xml. You are not required to provide the password again when you next launch the IO Design perspective. Storing the password in this way may pose a security risk.

If you deselect Remember Password, the IO Design perspective stores the password in memory for the duration of the session. When you next launch the IO Design perspective, you must provide the password again in order to connect to iHub. Remember Password is selected by default.

Figure 4-2 shows an example of an iHub profile.

- Choose Finish.

The profile appears in Server Explorer, as shown in Figure 4-3.

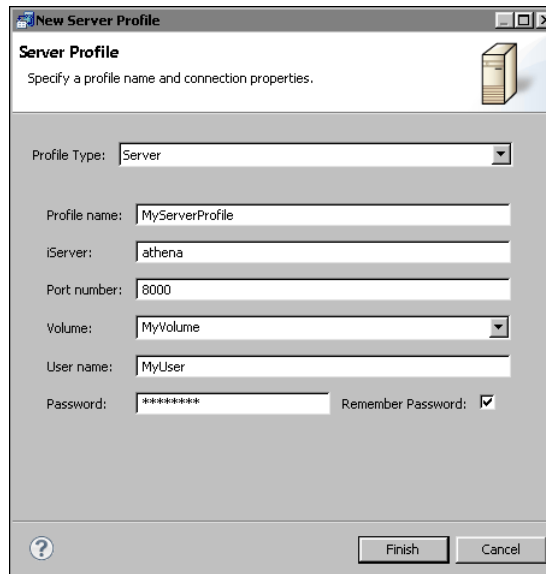


Figure 4-2 Specifying an iHub profile

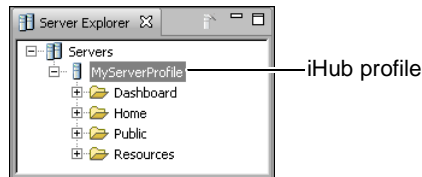


Figure 4-3 The Server Explorer view, showing an iHub profile

Publishing information object files as resources

Information object files must reside in a project's Shared Resources folder. By default, a project's Shared Resources folder is the project folder. If the Shared Resources folder is not the project folder, you must copy information object files to the Shared Resources folder before publishing. To check the location of the Shared Resources folder, choose Window→Preferences→Report Design→Resource.

When you publish information object files to an Encyclopedia volume, the files are published to the IO Designs folder in the Encyclopedia volume's resource folder. The resource folder's default location is /Resources. You must have write privilege on the resource folder.

How to copy information object files to the shared resources folder

- 1 In Navigator, select the appropriate project.
- 2 Choose File→Copy to Resources→Copy Information Objects to Shared Resources Folder.
- 3 In Share Information Objects, shown in Figure 4-4, select the appropriate files and folders. Choose Finish.

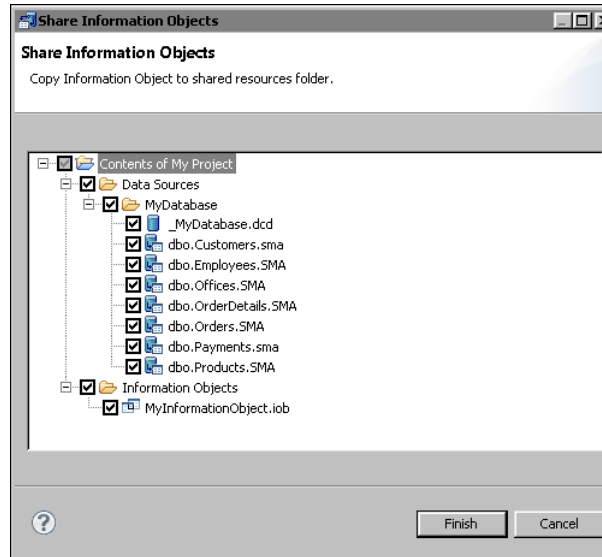


Figure 4-4 Copying information object files and folders to Shared Resources

How to publish information object files as resources

- 1 Choose File→Publish→Publish to Server.
- 2 In Publish to Server, in Server profile, choose an iHub profile from the drop-down list.
- 3 In Project, select the appropriate project from the drop-down list.
- 4 Select Publish Resources.
- 5 Select the appropriate files and folders, as shown in Figure 4-5.
- 6 In Version:
 - 1 Select Replace the latest version to replace the latest version of each file, or Create a new version to create a new version of each file.
 - 2 To copy permissions from the last version of each file, select Copy permissions from last version. If you do not select Copy permissions from

last version, you must set the permissions for each file using Management Console.

7 Choose Publish Files.

A confirmation dialog, shown in Figure 4-6, appears.

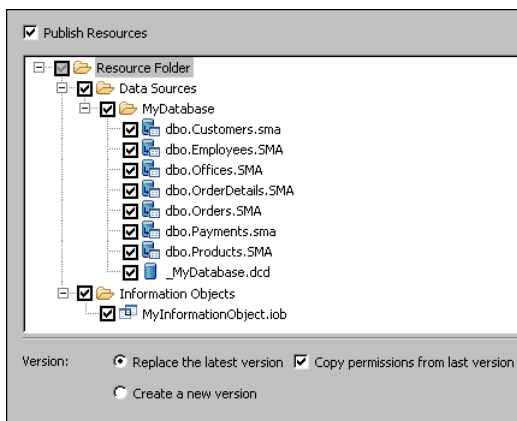


Figure 4-5 Publishing information object files as resources

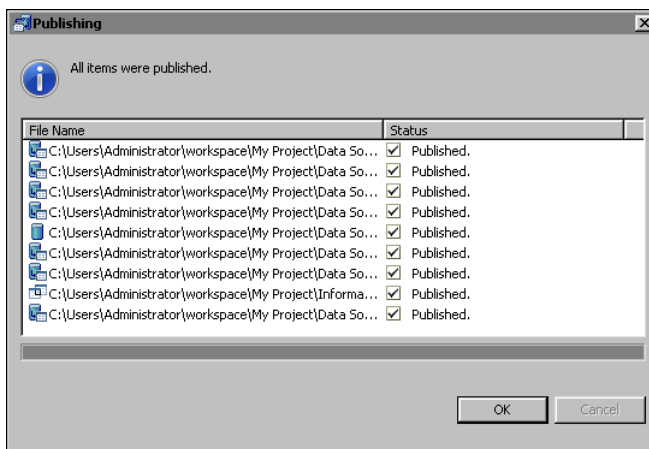


Figure 4-6 Publishing confirmation dialog

8 In Publishing, choose OK.

9 In Publish to Server, choose Close.

Publishing information object files as non-resources

For projects created with Release 11 Service Pack 4 and later, information object files are published to the volume's resource folder. For projects created with

Release 11 Service Pack 3 and earlier, information object files are published to the folder you specify in the Publish Information Objects dialog. To determine whether information object files are published as resources or not, select the appropriate project in Navigator and choose File→Properties→Actuate BIRT, as shown in Figure 4-7.

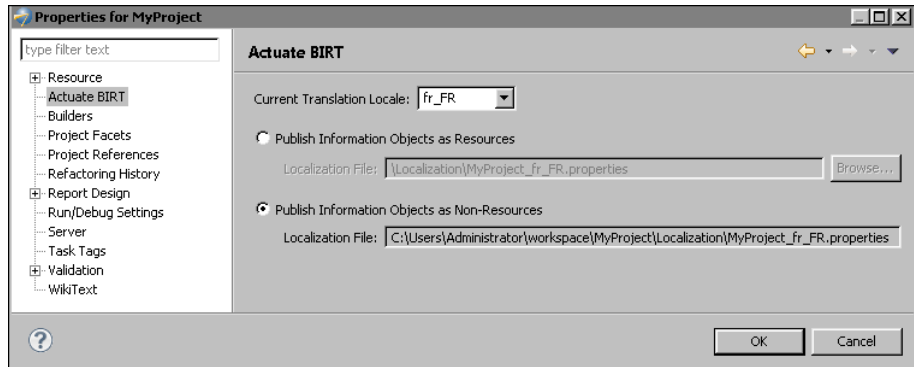


Figure 4-7 Publishing information object files as non-resources

How to publish information object files as non-resources

- 1 In Navigator, right-click the appropriate project and choose File→Publish→Publish Information Objects.
- 2 In Publish Information Objects:
 - 1 Select the project, a folder, or individual files.
 - 2 In Server profile, select a profile from the drop-down list.
 - 3 In Publish location, browse for the appropriate folder or accept the default folder, as shown in Figure 4-8.
 - 4 If you want the IO Design perspective to remember this location the next time you publish a project, select Remember this location.

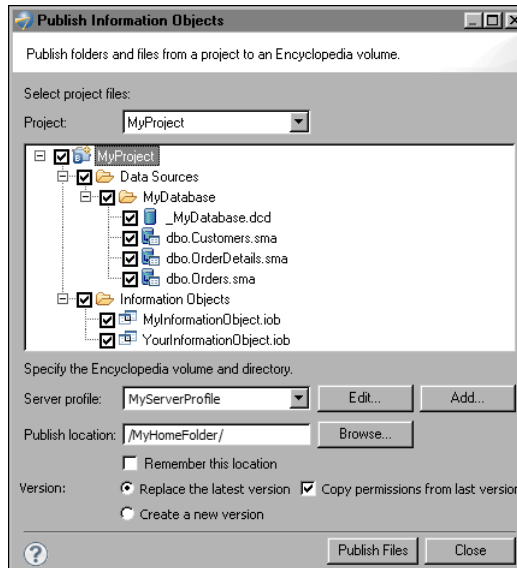


Figure 4-8 Selecting files and folders to publish

5 Choose Publish Files.

A confirmation dialog, shown in Figure 4-9, appears.

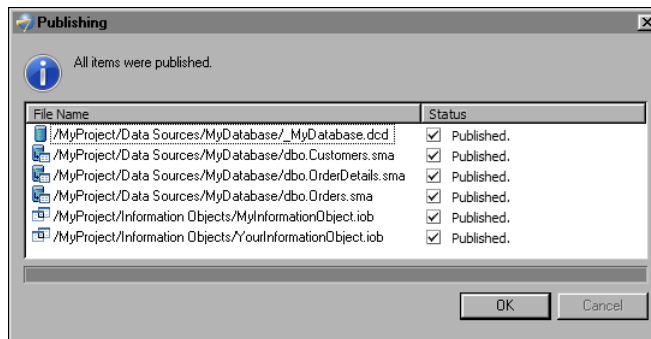


Figure 4-9 Publishing confirmation dialog

3 In Publishing, choose OK.

4 In Publish Information Objects, choose Close.

Downloading files from an Encyclopedia volume

You can download report design files and information object files from an Encyclopedia volume to an Actuate BIRT project. For example, you can download

the example files that install with iHub and examine them as a way to familiarize yourself with BIRT Designer Professional. Report design files and information object files are downloaded from different locations:

- Report design files (RPTDESIGNs) are downloaded from the folder in which they reside in the Encyclopedia volume to the project folder you specify. When you download report design files, you have the option to preserve the folder structure in the Encyclopedia volume.
- Information object files such as DCDs, SMAs, and IOBs are downloaded from the Encyclopedia volume's resource folder to the project's Shared Resources folder. When you download information object files, the folder structure in the resource folder is preserved.

To download a file from an Encyclopedia volume, you must have read privilege on the file.

How to download information object files from an Encyclopedia volume

- 1 Choose File→Download→Download from Server.
- 2 In Download from Server, choose an iHub profile from the drop-down list.
- 3 In Download to project, select the appropriate project from the drop-down list.
- 4 Select Download Resources.
- 5 Select the appropriate files and folders, as shown in Figure 4-10.

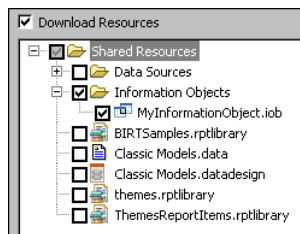


Figure 4-10 Downloading information object files

- 6 Choose Download.

The files and folders you download appear in Navigator.

Assessing the impact of project changes

This chapter contains the following topics:

- About project dependencies
- Searching for data connection definitions, maps, and columns
- Displaying the project model diagram
- Assessing the impact of a change on files in an Encyclopedia volume

About project dependencies

Most files in an Actuate BIRT project depend on other files in the project. For example, the project model diagram on the left in Figure 5-4 shows the dependencies between files in a very simple project consisting of a data connection definition, a map, an information object, and a report design. A change to a file impacts the files that depend on it. In our example, let's assume that a column's data type changes in the database table from which the map is built and you update the map. A change to the map obviously impacts the information object and the report design. In a typical project, however, the impact of a change is not so obvious, as most files depend on several other files and there are several layers of information objects.

The IO Design perspective provides tools to make it easier to assess the impact of project changes. You can search for specific data connection definitions, maps, and columns used in a project. You can then display a project model diagram and use the search results to show the impact of a change to a file or column. Figure 5-3 shows the search results for the `paymentDate` column in the project shown on the left in Figure 5-4. The project model diagram on the right in Figure 5-4 shows the impact of a change to the map.

To assess the impact of a change on files in an Encyclopedia volume such as report designs created in BIRT Studio and dashboard designs, you must use command-line scripts.

Searching for data connection definitions, maps, and columns

You can search for data connection definitions, maps, and columns used in a project.

How to search for a data connection definition, map, or column

- 1 Choose Search➤Database.
- 2 In Search—Database Search:
 - 1 Do one of the following:
 - ❑ In Databases, select a host name and type from the list. The list includes only database hosts found in the current project, regardless of the scope of the search.
 - ❑ In Search for database host name or select from the list below, type the name of the database host.

- In Search for database host name or select from the list below, make a selection from the drop-down list.
- 2 If you want to specify a table name and, optionally, a column name and have not already done so, choose Refine.
- 3 In Refine Database:
 - 1 In Catalog, select the appropriate catalog.
 - 2 In Filter:
 - To display tables and views from a particular schema, type the first few characters of the schema name in Schema name prefix, for example, dbo. Do not append an asterisk, for example, dbo*. This filter is case-sensitive.
 - To display only tables and views whose names begin with a particular string, type the string in Table/View name prefix, for example, ac. Do not append an asterisk, for example, ac*. This filter is case-sensitive.
 - Select Show tables only, Show views only, or Show all.
 - Choose Apply filter.
 - 3 In Available, select a table or expand the appropriate table and select a column. In Figure 5-1, the paymentDate column in the Payments table is selected. Choose OK.

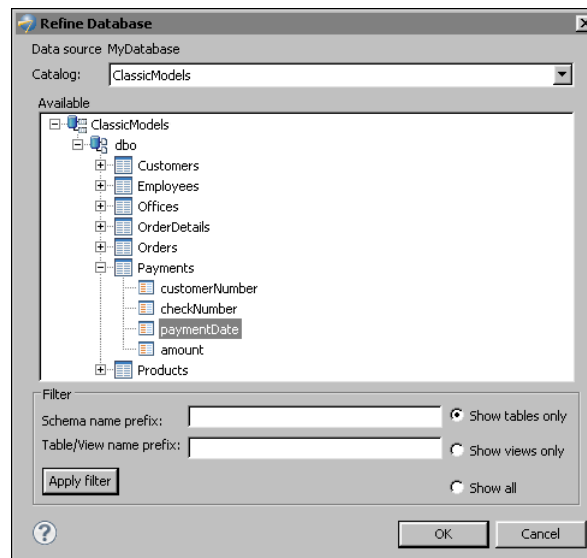


Figure 5-1 Selecting a database column to search for

- 4 In Scope, specify the scope of the search. You can search the entire workspace, the resources selected in Navigator, the projects that enclose the selected resources, or predefined working sets. The Search dialog looks similar to Figure 5-2. Choose Search.

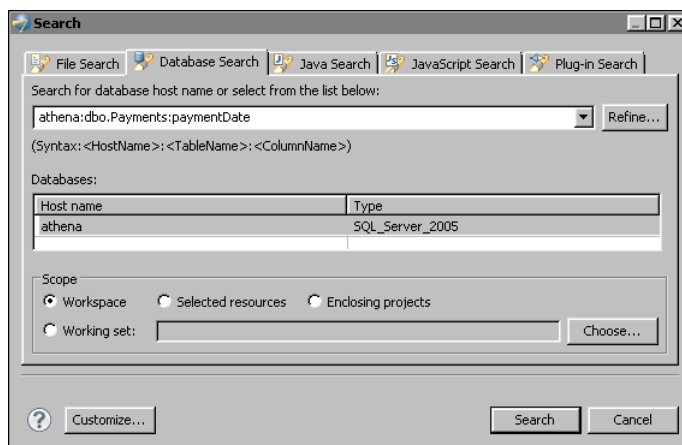


Figure 5-2 Searching for a database column

The search results appear in the Search view. If you searched only for a database host, the DCD path and file name appear. If you also searched for a table and a column, the SMA path and file name and column name appear as well, as shown in Figure 5-3.

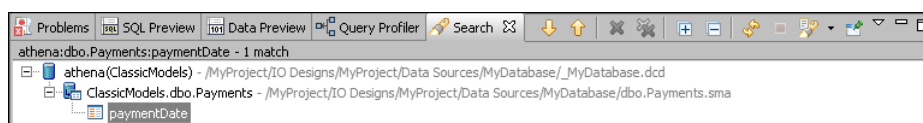


Figure 5-3 Displaying database search results

Displaying the project model diagram

The Navigator, shown in Figure 3-1, shows the physical relationships between project files and folders on disk. The Navigator does not show the logical relationships between files in a project. To display the logical relationships, you must use the project model diagram. The project model diagram shows, from top to bottom:

- Report designs
- The top layer of information objects
- Any intermediate layers of information objects

- The base layer of information objects
- Maps
- Data connection definitions

The project model diagram also shows data sets, data designs, report libraries, and external procedures. Downward-pointing arrows show the dependencies between files.

For example, the project model diagram on the left in Figure 5-4 shows the following dependencies:

- The report design `MyReport.rptdesign` is dependent on the information object `MyInformationObject.iob`.
- `MyInformationObject.iob` is dependent on the map `ClassicModels.dbo.Payments`.
- `ClassicModels.dbo.Payments` is dependent on the data connection definition `_MyDatabase.dcd`.

The project model diagram can help you assess the impact of a change to a file, column, or parameter by highlighting the impact path in red. For example, the project model diagram on the right in Figure 5-4 shows the impact of a change to `ClassicModels.dbo.Payments`. `MyReport.rptdesign`, `MyInformationObject.iob`, `ClassicModels.dbo.Payments`, and the arrows connecting them are highlighted in red to indicate that a change to the map impacts the information object and the report design.

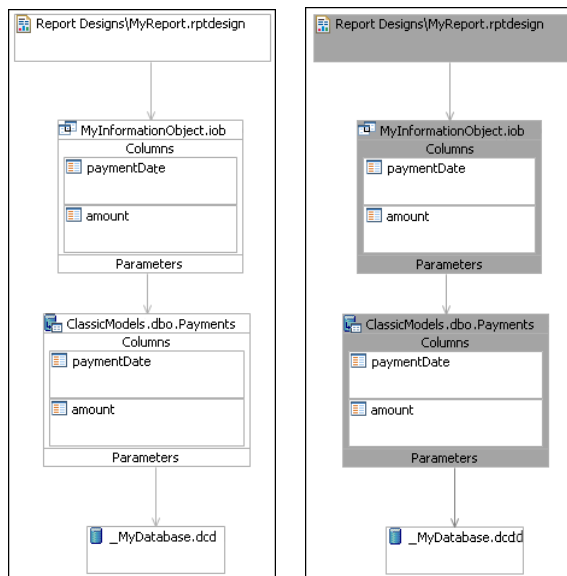


Figure 5-4 Project model diagram with (right) and without (left) impact path

You can display a report that lists the files impacted by a change to a file or column. For example, Figure 5-5 shows a report that lists the files impacted by a change to `dbo.Payments.sma`.

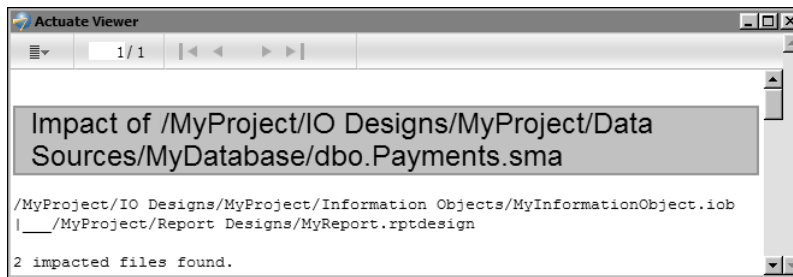


Figure 5-5 Impact report for `dbo.Payments.sma`

How to display the project model diagram

In the Navigator view or the Search view, right-click and choose Show Relationship Overview.

How to display a specific part of the project model diagram

In the Report Design perspective, in the Outline view, drag the gray box to the part of the project model diagram you want to display.

How to reinstate the project model diagram's initial layout

You can change the project model diagram's layout by dragging the shapes in the diagram to a new location. To reinstate the initial layout, right-click in the white space and choose Arrange All.

How to collapse or expand column and parameter lists

In the project model diagram, double-click the word Columns or the word Parameters for the appropriate column or parameter list. To collapse or expand both columns and parameters, right-click the map or information object file name and choose Collapse or Expand.

How to display an impact path

In the Navigator view, the Search view, or the project model diagram, right-click a file name, column name, or parameter name and choose Show Impact.

How to clear an impact path

In the white space in the project model diagram, right-click and choose Clear Impact.

How to refresh the project model diagram

In the white space in the project model diagram, right-click and choose Regenerate Diagram.

How to display an impact report

In the Navigator view or the Search view, right-click a file name or column name and choose Generate Impact Report.

How to open the editor associated with a file

In the project model diagram, right-click the file name and choose Open Editor.

Assessing the impact of a change on files in an Encyclopedia volume

The project model diagram can help you assess the impact of a change within an Actuate BIRT project. The project model diagram cannot, however, show the impact of a change on files in an Encyclopedia volume such as report designs created in BIRT Studio and dashboard designs. To assess the impact of a change on files in an Encyclopedia volume, you must use the following command-line scripts:

- `encycDownload.bat`
`encycDownload.bat` downloads files and folders from an Encyclopedia volume to a project in the local file system.
- `encycAnalyze.bat`
`encycAnalyze.bat` determines the dependencies between files in the project.
- `encycFind.bat`
`encycFind.bat` assesses the impact of a change to a file, data set, column, or parameter and generates an impact report.

These scripts reside in `<BDPro_HOME>\eclipse` and are only supported on platforms that support BIRT Designer Professional. Each script takes several arguments. Paths and arguments that contain spaces must be enclosed in quotation marks ("").

Downloading files from an Encyclopedia volume

To download files and folders from an Encyclopedia volume to a project folder in the local file system, run `encycDownload.bat`. You must run `encycDownload.bat` in the workspace folder in which you want to create the project. You must re-run `encycDownload.bat` whenever the contents of the Encyclopedia volume change.

To display the syntax for `encycDownload.bat`, open a Command Prompt window and `cd` to `<BDPro_HOME>\eclipse`. Then type:

```
encycDownload.bat -help
```

encycDownload.bat takes the following arguments:

- host
The host from which to download. The default is localhost.
- port
The port number to connect on. The default is 8000.
- volume
The Encyclopedia volume.
- location
The Encyclopedia volume folder from which to download. The default is the root folder.
- login
The Encyclopedia volume user name.
- pass
The Encyclopedia volume user's password. The default is no password.
- project
The name of the project to create. The default is DiagramProject.

For example, you open a Command Prompt window and cd to the workspace folder in which you want to create the project. You then type:

```
"C:\Program Files (x86)\Actuate\BRDPro2\eclipse  
  \encycDownload.bat" -project MyProject -location "/"  
  -login MyUser -volume MyVolume -host MyServer -port 8000
```

encycDownload.bat does the following:

- Creates the project MyProject in the workspace folder
- Logs in to the Encyclopedia volume MyVolume as MyUser
- Downloads the contents of MyVolume to MyProject

Determining the dependencies between project files

Once you have downloaded the contents of an Encyclopedia volume to a project folder, you must determine the dependencies between files in the project by running encycAnalyze.bat. You must run encycAnalyze.bat in the workspace folder in which you ran encycDownload.bat. You must re-run encycAnalyze.bat whenever the contents of the project change.

To display the syntax for encycAnalyze.bat, open a Command Prompt window and cd to <BDPro_HOME>\eclipse. Then type:

```
encycAnalyze.bat -help
```

encycAnalyze.bat takes one argument, the project name. The default is DiagramProject.

For example, you open a Command Prompt window and cd to the workspace folder in which you ran encycDownload.bat. You then type:

```
"C:\Program Files (x86)\Actuate\BRDPro2\eclipse\encycAnalyze.bat"
    -project MyProject
```

encycAnalyze.bat determines the dependencies between files in the project MyProject.

Generating an impact report

Once you have determined the dependencies between files in a project, you can run encycFind.bat to assess the impact of a change to a file, data set, column, or parameter and generate an impact report. You must run encycFind.bat in the workspace folder in which you ran encycAnalyze.bat. You can re-run encycFind.bat any number of times.

To display the syntax for encycFind.bat, open a Command Prompt window and cd to <BDPro_HOME>\eclipse. Then type:

```
encycFind.bat -help
```

encycFind.bat takes the following arguments:

- **project**
The name of the project to search. The default is DiagramProject.
- **file**
Path to the file whose change impact you want to assess. If you do not provide a query argument, you must provide a file argument.
- **entity (optional)**
If you provide a file argument, you can also specify a data set, column, or parameter. Names that contain spaces must be enclosed in quotation marks ("). The entity type must be compatible with the file type. For example, if the specified file is a report design, you can specify a data set or data set column. To specify the data set MyDataSet, type:

```
-entity dataset:MyDataSet
```

To specify the First Name column in the data set MyDataSet, type:

```
-entity column:MyDataSet:"First Name"
```

If the specified file is a map or information object, you can specify a column or parameter. To specify the First Name column, type:

```
-entity column:"First Name"
```

To specify the p_MaxDate parameter, type:

```
-entity parameter:p_MaxDate
```

- **query**
A database search string using the same syntax as in the Search—Database Search dialog, <HostName>:<TableName>:<ColumnName>. Change impact is assessed for the data connection definition, map, or column returned by the search. If you do not provide a file argument, you must provide a query argument.
- **report**
Name of the impact report. The default is impactResults.rptdocument. The report is created in the workspace folder.

For example, you open a Command Prompt window and cd to the workspace folder in which you ran encycAnalyze.bat. You then type:

```
"C:\Program Files (x86)\Actuate\BRDPro2\eclipse\encycFind.bat"  
-project MyProject -file "/MyProject/IO Designs/MyProject/Data  
Sources/MyDatabase/dbo.Payments.sma" -entity column:paymentDate  
-report MyImpactReport.rptdocument
```

encycFind.bat does the following:

- Assesses the impact of a change to the paymentDate column in the map dbo.Payments.sma
- Generates an impact report called MyImpactReport.rptdocument in the workspace folder

Alternatively, you can type the following command to achieve the same result:

```
"C:\Program Files (x86)\Actuate\BRDPro2\eclipse\encycFind.bat"  
-project MyProject -query athena:dbo.Payments:paymentDate  
-report MyImpactReport.rptdocument
```

To assess the impact of a change to the First Name column in the data set MyDataSet, type:

```
"C:\Program Files (x86)\Actuate\BRDPro2\eclipse\encycFind.bat"  
-project MyProject -file "/MyProject/Report Designs  
/MyReport.rptdesign" -entity column:MyDataSet:"First Name"  
-report MyImpactReport.rptdocument
```


Actuate SQL reference

This chapter contains the following topics:

- About Actuate SQL
- Differences between Actuate SQL and ANSI SQL-92
- Actuate SQL syntax
- Data types and data type casting
- Functions and operators
- Providing query optimization hints
- Using pragmas to tune a query

About Actuate SQL

An information object encapsulates an Actuate SQL query. You can create the Actuate SQL query that defines an information object in the IO Design perspective by typing the Actuate SQL query in the textual query editor or by specifying the desired query characteristics in the graphical query editor. If you use the graphical query editor, you can view the resulting Actuate SQL query in SQL Preview.

If you already have one or more existing information objects, you can access the information object data by specifying a query on the information object using Information Object Query Builder. You can create the query on the information object by typing a Actuate SQL query in the textual query editor or by specifying the desired query characteristics in a graphical query editor. If you use the graphical query editor, you can view the resulting Actuate SQL query in SQL Preview.

A query that defines an information object and a query on an information object both use Actuate SQL. Actuate SQL is based on the ANSI SQL-92 standard. This chapter describes the differences between Actuate SQL and ANSI SQL-92.

Differences between Actuate SQL and ANSI SQL-92

Actuate SQL is based on ANSI SQL-92. This section provides an overview of the differences between Actuate SQL and ANSI SQL-92.

Limitations compared to ANSI SQL-92

Actuate SQL has the following limitations compared to ANSI SQL-92:

- Only the SELECT statement is supported.
- Data types are limited to integers, strings, timestamps, floating point numbers, and decimals.
- Intersection and set difference operations are not available.
UNION and UNION DISTINCT are not supported. UNION ALL is supported. To obtain the same results as a UNION DISTINCT operation, perform a UNION ALL operation followed by a SELECT DISTINCT operation. For example, IO3 performs a UNION ALL operation on IO1 and IO2:

```

SELECT empNo, eName
FROM "IO1.iob" AS IO1
UNION ALL
SELECT empNo, eName
FROM "IO2.iob" AS IO2

```

To obtain distinct results from IO3, create IO4, which performs a SELECT DISTINCT operation on IO3:

```

SELECT DISTINCT empNo, eName
FROM "IO3.iob" AS IO3

```

- LIKE operator patterns and escape characters must be literal strings, parameters, or expressions. The LIKE operator does not support column references, subqueries, or aggregate functions, for example, MAX and AVG.
- Unnamed parameters are not supported.
- Some subqueries are not supported.
- Not all ANSI SQL-92 functions and operators are available.

Extensions to ANSI SQL-92

Actuate SQL has the following extensions to ANSI SQL-92:

- Parameterized queries with named parameters
A parameterized query starts with a WITH clause that specifies the names and types of parameters that the query uses. The following example shows using parameters to specify returning rows where salesTotal is within a range specified by two parameters:

```

WITH ( minTotal DECIMAL, maxTotal DECIMAL )
SELECT o.id, o.date
FROM "/sales/orders.sma" o
WHERE o.salesTotal BETWEEN :minTotal AND :maxTotal

```

A query with a parameterized SELECT statement is typed and is subject to the same casting rules as a function call, except that parameter declarations specify the maximum scale, precision, and length of parameter values. All parameter values are required. A parameter value must be a literal, for example 'abc', NULL, a parameter reference, or an Actuate SQL expression. A parameter value cannot be a column reference, for example ORDERS.ORDERID.

- Parameterized table, view, and query references
A parameterized table or view reference in a query enables specification of the query without knowing the table or view until run time. At run time, the values of the parameters specify the table. In the following example, the table is specified by the IOB name and the team and position parameters:

```

WITH( team VARCHAR, position VARCHAR, minGamesPlayed
      INTEGER )
SELECT playername
FROM "/sports/baseball/japan/players.iob" [:team,:position]
WHERE GamesPlayed > :minGamesPlayed

```

Parameter passing is typed and is subject to the same casting rules as a function call.

- **Scalar subqueries**

A scalar subquery is a query that returns a scalar value that is used in a second query. For example, the following query returns a scalar value:

```

SELECT SUM(B.Quantity * B.UnitPrice)
FROM "Order Detail.sma" AS B

```

This second query uses the previous query as a scalar subquery, evaluating the result of the scalar subquery and checking if the result is greater than 1000:

```

SELECT CustomerID
FROM "Customers.sma" C
LEFT OUTER JOIN
"Orders.sma" O
ON ( C.CustomerID=O.CustomerID )
WHERE
( SELECT SUM(B.Quantity * B.UnitPrice)
FROM "Order Detail.sma" AS B
) > 1000

```

- **Join control syntax specifying the join algorithm**

In Actuate SQL, you can specify the algorithm to use for joins. There are three join algorithms in Actuate SQL:

- **Dependent join**

A dependent join specifies obtaining all the results for the left side of the join and then using each resulting row to process the right side of the join. This algorithm is especially efficient when the left side of the join does not return many rows and the data source of the right side can handle evaluating the join criteria.

- **Nested loop join**

A nested loop join specifies obtaining and storing in memory all the results for the right side of the join. Then, for each row resulting from the left side, a nested loop join evaluates the right side results for matches to the join criteria. This algorithm is especially efficient when the right side of the join does not return many rows and the join expression cannot be delegated to the data source of the right side.

- Merge join

A merge join specifies obtaining the results for the right and left sides of the join and comparing these results row by row. Merge joins are applicable only for joins where the value on the left must be equal to the value on the right. This algorithm uses less memory than a nested loop join. This algorithm is especially efficient if the data sources sort the rows but presorting is not required.

The following example shows a merge join in a simple SELECT statement:

```
SELECT customers.custid, customers.customname,  
       customers.city, salesreps.lastname, salesreps.email  
FROM customers MERGE JOIN salesreps  
ON customers.repid = salesreps.repid
```

The following example shows a dependent join in a parameterized SELECT statement:

```
WITH ( minRating INTEGER )  
SELECT c.name, o.date, o.shippingStatus  
FROM  
    "/uk/customers.sma" c  
DEPENDENT JOIN  
    "/sales/orders.sma" o  
ON c.id = o.custId  
WHERE c.rating >= :minRating
```

You can also specify whether the join is an inner join or left outer join. The following example shows a SELECT statement with a left outer join:

```
SELECT customers.custid, customers.contact_last,  
       customers.contact_first, salesreps.lastname,  
       salesreps.firstname  
FROM salesreps LEFT OUTER JOIN customers  
ON salesreps.firstname = customers.contact_first
```

For information about inner and outer joins, see the SQL reference guide for your database.

- Pragmas to alter query semantics
- Additional functions
- The ability to have ORDER BY items other than SELECT items or aliases, for example:

```
SELECT customers.contact_first || ' ' ||  
       customers.contact_last  
       "MOST_VALUED_CUSTOMERS"  
FROM "/customers.sma" customers  
WHERE customers.purchasevolume > 3  
ORDER BY customers.purchasevolume DESC
```

If an ORDER BY item is not a SELECT item or an alias, it must be a grouping column if a GROUP BY clause is present. ORDER BY items must be SELECT items if SELECT DISTINCT is specified.

Use ORDER BY only when creating a query in Information Object Query Builder. Do not use ORDER BY when you create an information object in the IO Design perspective.

- The ability to have GROUP BY items that are expressions, for example:

```
SELECT DATEPART('yyyy', orders.shipbydate) "YEAR",
       DATEPART('m', orders.shipbydate) "MONTH",
       COUNT(*) "NUM_ORDERS"
FROM "/orders.sma" orders
GROUP BY DATEPART('yyyy', orders.shipbydate),
         DATEPART('m', orders.shipbydate)
```

To use an expression as a GROUP BY item, the expression must appear as a SELECT item. Aggregate functions are not allowed in GROUP BY expressions unless they are outer references from a subquery and the subquery is contained in the HAVING clause of the parent query. Complex GROUP BY expressions cannot be used in the HAVING clause of the query.

- The ability to use references to aliases

Database limitations

Because the Integration service delegates many of its operations to the databases, these operations are affected by database limitations, such as the maximum precision of decimal types or the treatment of zero-length strings.

Actuate SQL syntax

Actuate SQL syntax is similar to SQL-92 syntax. Actuate SQL has additional syntax for naming tables and columns. Table 6-1 provides a description of the typographical conventions used in describing Actuate SQL grammar.

Table 6-1 Typographical conventions used in describing Actuate SQL grammar

Convention	Used for...
NORMAL UPPERCASE	Actuate SQL keywords.
<i>ITALICIZED</i> <i>UPPERCASE</i>	Tokens.
(vertical bar)	Separating syntax items. You choose one of the items.
[] (brackets)	Optional syntax items. Do not type the brackets.

Table 6-1 Typographical conventions used in describing Actuate SQL grammar

Convention	Used for...
{ } (braces)	Required syntax items. Do not type the braces.
[...n]	Indicating that the preceding item can be repeated n number of times. The item occurrences are separated by commas.
[...n]	Indicating that the preceding item can be repeated n number of times. The item occurrences are separated by blanks.
<label>	The name for a block of syntax. This convention is used to label syntax that can appear in more than one place within a statement. Each location in which the block of syntax can appear is shown with the label enclosed in chevrons, for example <label>.

Table 6-2 lists the tokens used in the Actuate SQL grammar.

Table 6-2 Tokens used in describing the Actuate SQL grammar

Token	Definition
IDENTIFIER	A sequence of Unicode letters, digits, dollar signs, and underscores combining characters and extenders. The first character must be a letter. Use double quotes to quote identifiers. To represent a double quote within a quoted identifier, use two double quotes. Quoted identifiers can include any characters except carriage return or new line.
CHAR_LITERAL	Any Unicode text between single quotes other than carriage return or new line. To represent a single quote, use two single quotes. Multiple consecutive character literals are concatenated.
DECIMAL_LITERAL	An integer literal followed by a decimal point and an optional integer representing the fractional part. Syntax: (INTEGER_LITERAL .) (. INTEGER_LITERAL) (INTEGER_LITERAL. [INTEGER_LITERAL])
DOUBLE_LITERAL	A number of the form 1.2E+3. If the sign is omitted, the default is positive. Syntax: ((. INTEGER_LITERAL) (INTEGER_LITERAL. [INTEGER_LITERAL])) [(e E) [- +] INTEGER_LITERAL]

(continues)

Table 6-2 Tokens used in describing the Actuate SQL grammar (continued)

Token	Definition
INTEGER_LITERAL	One or more consecutive digits.
TIMESTAMP_STRING	A literal string that is interpreted as a timestamp value, such as '2002-03-31 13:56:02.7'. Years are 4 digits. Seconds are 2 digits with an optional fraction up to 3 digits. All other fields are 2 digits. The space between the date and time sections is required. Format: 'yyyy-mm-dd hh:mm:ss.msec'

Actuate SQL grammar

The Actuate SQL grammar contains one statement. The syntax of this statement is:

```
[<Pragma>] [...n] [<QueryParameterDeclaration>] <SelectStatement>
```

Table 6-3 provides the syntax for the grammar parts used in these statements.

Table 6-3 Syntax for the Actuate SQL grammar parts

Grammar part	Syntax
AdditiveExpression	<MultiplicativeExpression> {(+ -) <MultiplicativeExpression>} [...n]
AggrExpression	COUNT (([ALL DISTINCT] <ValueExpression> *)) (AVG MAX MIN SUM) ([ALL DISTINCT] <ValueExpression>)
AndExpression	{<UnaryLogicalExpression>} [AND...n]
CardinalityType	1 ? * +
CaseExpression	CASE [<ValueExpression>] {<WhenClause>} [...n] [ELSE <ValueExpression>] END
CastExpression	CAST((<ValueExpression> NULL) AS <ScalarDataType>)
ColumnAlias	IDENTIFIER
CondExpr	{<AndExpression>} [OR...n]
ConditionalPrimary	(<CondExpr>) <SimpleCondition>
DataType	<ScalarDataType>
ExplicitInnerOuterType	LEFT [OUTER] INNER
ExplicitJoinType	MERGE NL DEPENDENT

Table 6-3 Syntax for the Actuate SQL grammar parts (continued)

Grammar part	Syntax
ExpressionList	{<ValueExpression>} [...n]
FilterClause	FILTERS(<i>IDENTIFIER</i> DataType 'ValueExpression' [...n]) Use FILTERS only from a report designer.
FromClause	{FROM <FromTableReference>} [...n]
FromTableName	IDENTIFIER [(<TableParameters>)] [[AS] IDENTIFIER] If the identifier is not enclosed in quotes, it is interpreted as a table. If the identifier is enclosed in quotes, it is interpreted as an absolute or relative path in the Encyclopedia volume.
FromTableReference	<JoinExpression> (<JoinExpression>) <FromTableName>
FunctionCallOrColumnRef	IDENTIFIER (([<ExpressionList>]) [. IDENTIFIER])
GroupByClause	GROUP BY {<ValueExpression>} [...n] ValueExpression can be an expression as long as the expression also appears as a SELECT item.
HavingClause	HAVING <CondExpr>
JoinCondition	ON <CondExpr> [{ CARDINALITY('<CardinalityType> - <CardinalityType>') }]
JoinElement	(<JoinExpression>) <FromTableName>
JoinExpression	<JoinElement> {<JoinSpec><JoinElement> [<JoinCondition>]} [...n]
JoinSpec	[[[LEFT RIGHT] OPTIONAL] <ExplicitInnerOuterType>] [<ExplicitJoinType>] JOIN
Length	INTEGER_LITERAL
MultiplicativeExpression	<UnaryExpression> {(* /) UnaryExpression} [...n]
NamedParameter	: <i>IDENTIFIER</i>
OrderByClause	ORDER BY {<ValueExpression> (ASC DESC)? } [...n] ValueExpression is not limited to SELECT items or aliases. If ValueExpression is not a SELECT item or an alias, it must be a grouping column if a GroupByClause is present. Use ORDER BY only when creating a query in Information Object Query Builder. Do not use ORDER BY when you create an information object in the IO Design perspective.
ParameterDeclaration	<i>IDENTIFIER</i> [<AS>] <Data Type>
ParamPlaceholder	<NamedParameter>
Pragma	PRAGMA <i>IDENTIFIER</i> := CHAR_LITERAL
Precision	INTEGER_LITERAL

(continues)

Table 6-3 Syntax for the Actuate SQL grammar parts (continued)

Grammar part	Syntax
PrimaryExpression	<FunctionCallOrColumnRef> <ParamPlaceholder> <UnsignedLiteral> <AggrExpression> (<ValueExpression>) <CastExpression>
QueryParameterDeclaration	WITH ({<ParameterDeclaration>} [,...n]) All parameters are required.
RelationalOperator	= < <=> >=
ScalarDataType	VARCHAR [(<Length>)] DECIMAL [(<Precision>, <Scale>)] INTEGER DOUBLE [<Precision>] TIMESTAMP
Scale	INTEGER_LITERAL
SelectItem	<ValueExpression> [[AS] <ColumnAlias>]
SelectList	{<SelectItem>} [,...n]
SelectStatement	(<SelectWithoutOrder> [<OrderByClause>]) <SelectWithoutFrom>
SelectWithoutFrom	SELECT <ValueSelectList>
SelectWithoutOrder	(SELECT [ALL DISTINCT] <SelectList> <FromClause> [<WhereClause>] [<GroupByClause>] [<HavingClause>] [<SetClause>]) (<SelectWithoutOrder>)) [<SetClause>]
SetClause	UNION ALL (<SelectWithoutOrder> <SelectWithoutFrom>)
SignedLiteral	CHAR_LITERAL [+ -]INTEGER_LITERAL [+ -]DOUBLE_LITERAL [+ -]DECIMAL_LITERAL TIMESTAMP TIMESTAMP_STRING

Table 6-3 Syntax for the Actuate SQL grammar parts (continued)

Grammar part	Syntax
SimpleCondition	EXISTS <SubQuery> <SubQuery> <RelationalOperator> <ValueExpression> <ValueExpression> (<RelationalOperator> (([ANY ALL] <SubQuery>) <ValueExpression>) IS [NOT] NULL [NOT] (BETWEEN <ValueExpression> AND <ValueExpression> LIKE <ValueExpression> [ESCAPE <ValueExpression>] IN <SubQuery> IN (ExpressionList)))) The escape character must evaluate to a single character other than a single quote, a percent sign, or an underscore.
SubQuery	(<SelectWithoutOrder> [OPTION (SINGLE EXEC)])
TableParameter	(<SignedLiteral> NULL <ParameterReference> <ValueExpression>)
TableParameters	<TableParameter> [,...n]
UnaryExpression	[+ -] <PrimaryExpression>
UnaryLogicalExpression	[NOT] <ConditionalPrimary>
UnsignedLiteral	CHAR_LITERAL INTEGER_LITERAL DOUBLE_LITERAL DECIMAL_LITERAL TIMESTAMP TIMESTAMP_STRING
ValueExpression	<AdditiveExpression> <CaseExpression>
ValueSelectItem	<ValueExpression> [[AS] <ColumnAlias>]
ValueSelectList	{<ValueSelectItem>} [,...n]
WhenClause	WHEN <ValueExpression> THEN <ValueExpression>
WhereClause	WHERE <CondExpr>

Using white space characters

White space characters include the space, tab, and new line characters. Multiple white space characters are not significant outside of literal strings and quoted identifiers.

Using keywords

The Actuate SQL keywords are shown in the following list:

- | | | |
|---------------|-----------|-------------|
| ■ ALL | ■ EXEC | ■ ON |
| ■ AND | ■ EXISTS | ■ OPTION |
| ■ ANY | ■ FALSE | ■ OPTIONAL |
| ■ AS | ■ FILTERS | ■ OR |
| ■ ASC | ■ FROM | ■ ORDER |
| ■ AVG | ■ GROUP | ■ OUTER |
| ■ BETWEEN | ■ HAVING | ■ PRAGMA |
| ■ BY | ■ IN | ■ PRECISION |
| ■ CARDINALITY | ■ INNER | ■ RIGHT |
| ■ CASE | ■ INT | ■ SELECT |
| ■ CAST | ■ INTEGER | ■ SINGLE |
| ■ COUNT | ■ IS | ■ SUM |
| ■ DEC | ■ JOIN | ■ THEN |
| ■ DECIMAL | ■ LEFT | ■ TIMESTAMP |
| ■ DEPENDENT | ■ LIKE | ■ TRUE |
| ■ DESC | ■ MAX | ■ UNION |
| ■ DISTINCT | ■ MERGE | ■ VARCHAR |
| ■ DOUBLE | ■ MIN | ■ WHEN |
| ■ ELSE | ■ NL | ■ WHERE |
| ■ END | ■ NOT | ■ WITH |
| ■ ESCAPE | ■ NULL | |

Actuate SQL keywords are not case-sensitive. To prevent incompatibility with other versions of SQL, do not use SQL-92 keywords. If you use an identifier that is also a keyword, place double quotes around the identifier.

Using comments

Precede a single-line comment with two hyphens. Enclose a multiline comment with `/*` and `*/`.

Specifying maps and information objects in Actuate SQL queries

In the IO Design perspective, a map or information object name should be qualified by its relative path in the Encyclopedia volume. The path is relative to the IOB file. Use forward slashes to separate components of the path, for example:

```
../Data Sources/MyDatabase/dbo.customers.sma
```

In a query from a report designer, a map or information object name should be qualified by its absolute path in the Encyclopedia volume. Use forward slashes to separate components of the path, for example:

```
/MyProject/Data Sources/MyDatabase/dbo.customers.sma
```

Using identifiers in Actuate SQL

Identifiers include table and column names. Actuate SQL identifiers have the same limitations as standard SQL identifiers. For example, you must enclose an identifier in double quotes if it contains an illegal character such as a space or if it is identical to a SQL-92 keyword. Unlike the SQL-92 standard, however, there is no length limitation on Actuate SQL identifiers. Identifiers can contain Unicode characters.

Using column aliases in Actuate SQL

When you use column aliases, the following rules apply:

- The column and alias names of the items in the first SELECT statement of a UNION of statements are definitive.
- Within the items in a SELECT statement, you can use previously defined aliases to create expressions, for example:

```
SELECT col1 AS a, col2 AS b, a+b
```
- Only SELECT and ORDER BY can use aliases.
- You cannot use an alias in an aggregate expression, for example, MAX(a).
- You can use aliases defined in an outer SELECT statement in a nested SELECT statement.
- You can use aliases from the items in the first SELECT statement in a set of UNION statements in the ORDER BY clause of the query.

Specifying parameter values

A parameter value must be one of the following:

- A literal value, for example 'abc' or 123
- The NULL literal value
- A parameter reference
- An expression consisting of literal values, parameter references, and Actuate SQL functions and operators

A parameter value cannot include column references, subqueries, or aggregate functions.

Examples MyInformationObject uses the parameters p1, p2, and p3. The following query passes the parameter values :p1, -100, and 'abc' to MyInformationObject. :p1 represents the value of parameter p1 provided by the user. -100 and 'abc' are literal values:

```
WITH ( p1 INTEGER, p2 INTEGER, p3 VARCHAR )
SELECT ...
FROM "MyInformationObject.iob" [ :p1, -100, 'abc' ]
```

MyInformationObject uses the parameter p1. The following query passes the NULL literal value to MyInformationObject:

```
WITH ( p1 INTEGER )
SELECT ...
FROM "MyInformationObject.iob" [ NULL ]
```

MyInformationObject uses the parameter p1. The following query passes the NULL literal value, cast as integer data type, to MyInformationObject:

```
WITH ( p1 INTEGER )
SELECT ...
FROM "MyInformationObject.iob" [ CAST(NULL AS INTEGER) ]
```

MyInformationObject uses the parameter p1. The following query passes the expression :p1 + 10 to MyInformationObject:

```
WITH ( p1 INTEGER )
SELECT ...
FROM "MyInformationObject.iob" [ :p1 + 10 ]
```

MyInformationObject uses the parameters p1 and p2. The following query passes the parameter reference :p1 and the expression :p1 || :p2 to MyInformationObject:

```
WITH ( p1 VARCHAR, p2 VARCHAR )
SELECT ...
FROM "MyInformationObject.iob" [ :p1, :p1 || :p2 ]
```

MyInformationObject uses the parameters p1 and p2. The following query passes two expressions to MyInformationObject:

```
WITH ( p1 INTEGER, p2 INTEGER )
SELECT ...
FROM "MyInformationObject.iob" [:p1 + 10, CASE WHEN :p2 > 100 THEN
    100 ELSE 0 END]
```

Using subqueries in Actuate SQL

Subqueries have the following limitations:

- Subqueries are supported in every clause except the FROM clause. Specifically:
 - Subqueries cannot be used in Actuate SQL parameters or JOIN conditions.
 - Subqueries cannot constitute derived tables.
Derived tables are tables in a FROM clause that are the result of running a subquery.
- Subqueries must be operands to the operators IN or EXISTS, or operands to a comparison operator such as =, >, or >=ALL. Only one operand of the comparison operator can be a subquery, not both.
- Only single-column subqueries are supported. In other words, each subquery must have only one SELECT item.
- Subqueries cannot have more than one SELECT statement. In other words, set operators such as UNION ALL are not allowed in subqueries.

Subqueries can use OPTION (SINGLE EXEC). The SINGLE EXEC option improves the performance of a query when the query cannot be pushed to the database. When the SINGLE EXEC option is specified, the non-correlated portion of the subquery is executed once against the target database, while the correlated portion is executed within the Integration service.

By default, a subquery from a different database is implemented using a dependent join. Using the SINGLE EXEC option, a subquery can be executed using a single dependent query instead of executing one dependent query for each row of the outer query, for example:

```
SELECT DISTINCT CUSTOMERS.CUSTID AS "CUSTID",
    ORDERS.ORDERID AS "ORDERID"
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
INNER JOIN "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS
ON CUSTOMERS.CUSTID = ORDERS.CUSTID
WHERE (SELECT count (ITEMS.PRICEQUOTE)
FROM "../Data Sources/YourDatabase/ITEMS.SMA" ITEMS
```

```
WHERE ORDERS.ORDERID = ITEMS.ORDERID
OPTION (SINGLE EXEC) < 100
ORDER BY CUSTOMERS.CUSTID, ORDERS.ORDERID
```

Using derived tables in Actuate SQL

A derived table is a virtual table that is calculated on the fly from a SELECT statement. A derived table can be used in a FROM clause, WHERE clause, HAVING clause, or subquery, for example:

```
SELECT Column01
FROM (Derived_table)
```

A derived table can have parameters.

Data types and data type casting

Table 6-4 lists Actuate SQL data types and a description of each data type.

Table 6-4 Actuate SQL data types

Data type	Description
String ("VARCHAR")	A sequence of Unicode characters. You can specify a maximum character length for the string. For example, VARCHAR (30) represents strings with a maximum length of 30 Unicode characters.
Integer number ("INTEGER")	32-bit two's-complement arithmetic numbers.
Decimal number ("DECIMAL")	Fixed point numbers consisting of up to 100 digits. You can specify a maximum scale and a maximum precision using the syntax (precision, scale). For example, DECIMAL (15, 4) represents decimals that can have up to 15 digits in all and up to 4 digits after the decimal point.
Floating point number ("DOUBLE")	64-bit IEEE double precision floating point numbers.
Timestamp ("TIMESTAMP")	A combined date and time (hour/minute/second).

Facets

The precision, scale, and length associated with a database data type are called facets. Facets are supported for the corresponding Actuate SQL data type.

An Actuate SQL expression that evaluates to a scalar value has facets. These facets are determined by the Actuate SQL functions used in the expression and the facets on the columns in the expression. You can specify facets for an Actuate SQL expression by using a cast, for example:

```
CAST (EXPR AS DECIMAL (38, 8))
```

```
CAST (EXPR AS VARCHAR(25))
```

Parameters in Actuate SQL queries have facets. These facets determine the maximum precision, scale, and length of parameter values. When no facets are specified for a parameter or a cast expression, the defaults are used. The default precision and scale for the Actuate SQL DECIMAL data type are (20, 8). The default length for the Actuate SQL VARCHAR data type is 50.

If the decimal value passed into a parameter or cast expression is too large for the precision and scale, an error results. Actuate SQL truncates digits after the decimal point to force the decimal value to fit within the precision and scale.

If the string or timestamp value passed into a string parameter, or into a cast expression to VARCHAR, is too large for the string length specified, the string or timestamp is truncated. If the string value passed into a cast expression to DECIMAL is too large for the precision and scale specified, an error results.

By default, Actuate SQL has a decimal precision of 38. The decimal precision can be set to a smaller or larger value up to 100. Results of calculations that exceed this limit may have their precision and scale truncated. Calculations may also be limited by the database. The same applies to operations on strings in the database.

Casting rules

The following casting rules apply:

- Integers can be implicitly cast to decimals and doubles. For implicit casts to decimals, the resulting decimals have a precision of 10 and a scale of 0. Integers can be explicitly cast to these types, as well as to strings.
- Decimals can be implicitly cast to doubles. Decimals can be explicitly cast to doubles, as well as to integers and strings. Conversion to integer type may result in rounding or truncation of data.
- Doubles can be explicitly cast to strings, as well as to integers and decimals. Conversion to decimal and integer types may result in rounding or truncation of data.
- Timestamps can be explicitly cast to strings. Casting to other types is not permitted.

- Strings can be implicitly or explicitly cast to timestamps. For explicit casting, the strings must be of the form:
`yyyy-MM-dd hh:mm:ss.fff`
Strings can be explicitly cast to integers, decimals, and doubles.
- Because databases vary in their implementation, casts to strings do not have a defined format. For example, the same value can be represented as 6E5, 60000, or 60000.00.
- All types can be implicitly cast to the same type.

Table 6-5 summarizes the casting rules for Actuate SQL data types.

Table 6-5 Casting rules for Actuate SQL types

	To INTEGER	To DECIMAL	To DOUBLE	To VARCHAR	To TIMESTAMP
From INTEGER	Implicit casting occurs	Implicit casting occurs	Implicit casting occurs	Explicit casting required	Casting not permitted
From DECIMAL	Explicit casting required	Implicit casting occurs	Implicit casting occurs	Explicit casting required	Casting not permitted
From DOUBLE	Explicit casting required	Explicit casting required	Implicit casting occurs	Explicit casting required	Casting not permitted
From VARCHAR	Explicit casting required	Explicit casting required	Explicit casting required	Implicit casting occurs	Implicit casting occurs
From TIMESTAMP	Casting not permitted	Casting not permitted	Casting not permitted	Explicit casting required	Implicit casting occurs

String comparison and ordering

The BIRT iHub Integration service compares and orders strings according to the Unicode code point value of each character. For example, Bright-Abbott is sorted before Brightman because the hyphen (-) has a Unicode value of 45, while lowercase m has a Unicode value of 109. The expression:

```
'Kirsten' LIKE 'ki%'
```

evaluates to False because uppercase K is different from lowercase k.

Although string comparison is case-sensitive by default, you can configure the Integration service to do case-insensitive comparison and ordering.

Functions and operators

Actuate SQL supports several built-in operators and named functions. Functions and operators are described in the following topics, grouped by related functionality.

Comparison operators: =, <>, >=, >, <=, <

Comparison operators are used to compare the value of two expressions, returning True if the comparison succeeds, and False if it does not. The following rules apply to the use of comparison operators handled by the Integration service:

- For numeric data types, the usual rules of arithmetic comparisons apply.
- For string comparisons, the shorter of the two strings is padded with space characters to equal the length of the longer string before the comparison is performed, as in SQL-92.
- Timestamps are compared using chronological order.
- An equality comparison between two floating point numbers does not return an error.

For information about the Integration service, see *Configuring BIRT iHub*. Comparison operations delegated to a remote data source may vary from the rules for comparison operators handled by the Integration service.

Range test operator: BETWEEN

The BETWEEN operator tests a value to see if it occurs in a given range including the endpoints. For example, the expression:

```
col BETWEEN 10 AND 20
```

evaluates to True if and only if the value of col is at least 10 but no more than 20. Table 6-6 shows the result type for using BETWEEN for each operand data type.

Table 6-6 Result data types for using BETWEEN with various operand types

First operand type	Second operand type	Third operand type	Result type
Boolean	Boolean	Boolean	Boolean
Integer	Integer	Integer	Boolean
Decimal	Decimal	Decimal	Boolean
Double	Double	Double	Boolean

(continues)

Table 6-6 Result data types for using BETWEEN with various operand types (continued)

First operand type	Second operand type	Third operand type	Result type
Varchar	Varchar	Varchar	Boolean
Timestamp	Timestamp	Timestamp	Boolean

The BETWEEN operator follows the same rules as the comparison operators.

Comparison operator: IN

The IN operator tests a row or scalar value to see if it occurs in a set of values. For example, the expression:

```
column IN (1,3,5,7,9)
```

evaluates to True if and only if the value of column is 1, 3, 5, 7, or 9.

Arithmetic operators: +, -, *, /

These operators implement addition, subtraction, multiplication, and division on the supported numeric data types. For decimal data types, the result's precision and scale are shown in Table 6-7. d1 represents an operand expression with precision p1 and scale s1, and d2 represents an operand expression with precision p2 and scale s2. The result's precision and scale may be truncated due to database limitations.

Table 6-7 Precision and scale of arithmetic operation results

Operation	Result's precision	Result's scale
d1 + d2	$\max(s1, s2) + \max(p1-s1, p2-s2) + 1$	$\max(s1, s2)$
d1 - d2	$\max(s1, s2) + \max(p1-s1, p2-s2) + 1$	$\max(s1, s2)$
d1 * d2	$p1 + p2 + 1$	$s1 + s2$
d1 / d2	$p1 - s1 + s2 + \max(6, s1 + p2 + 1)$	$\max(6, s1 + p2 + 1)$

Integer arithmetic operations are performed using 32-bit two's-complement semantics. Floating point operations are performed according to the IEEE double precision standard.

These general rules apply to operations handled by the Integration service. Operations delegated to remote data sources may vary in their semantics. For information about the Integration service, see *Configuring BIRT iHub*.

Table 6-8 shows the result type of using arithmetic operators with each operand type.

Table 6-8 Result data types for using arithmetic operators with various operand types

Left operand type	Right operand type	Result type
Integer	Integer	Integer
Decimal	Decimal	Decimal
Double	Double	Double

Numeric functions

Actuate SQL supports the following numeric functions:

- FLOOR, CEILING, MOD
- ROUND
- POWER

FLOOR, CEILING, MOD

FLOOR returns the largest integer not greater than the argument's value. The result is cast to the specified type:

Decimal **FLOOR**(value Decimal)
Double **FLOOR**(value Double)

Example The following code:

```
SELECT FLOOR(123.45), FLOOR(-123.45), FLOOR(0.0)
```

returns:

```
123, -124, 0
```

CEILING returns the smallest integer not less than the argument's value. The result is cast to the specified type:

Decimal **CEILING**(value Decimal)
Double **CEILING**(value Double)

Example The following code:

```
SELECT CEILING(123.45), CEILING(-123.45), CEILING(0.0)
```

returns:

```
124, -123, 0
```

MOD returns the remainder after division of two integers:

Integer **MOD**(v1 Integer, v2 Integer)

Example The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME  
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS  
WHERE MOD(CUSTOMERS.CUSTID, 2) = 1
```

returns:

```
101,Signal Engineering  
109,InfoEngineering  
111,Advanced Design Inc.  
...
```

For decimal data types, the result's precision and scale for the FLOOR and CEILING functions are $(p + 1, s)$, where (p, s) are the precision and scale of the operand.

ROUND

ROUND returns the number closest in value to the first argument, rounding away from zero. The second argument specifies the precision, with positive values indicating a position to the right of the decimal point, and negative values indicating a position to the left of the decimal point. All positions to the right of the specified position are zero in the result:

```
Integer ROUND( value Integer, precision Integer )  
Decimal ROUND( value Decimal, precision Integer )  
Double ROUND( value Double, precision Integer )
```

Example The following code:

```
SELECT ROUND(123.4567, 2), ROUND(123.4567, -1)
```

returns:

```
123.46, 120
```

For decimal data types, the result's precision and scale are $(p + 1, s)$, where (p, s) are the precision and scale of the operand.

POWER

POWER raises the left argument (base) to the power of the right argument (exponent):

```
Integer POWER( base Integer, exponent Integer )  
Decimal POWER( base Decimal, exponent Integer )  
Double POWER( base Double, exponent Integer )
```

Example The following code:

```
SELECT CUSTOMERS.CUSTID, POWER(CUSTOMERS.CUSTID, 2)  
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
```

returns:

```
101,10201
102,10404
104,10816
...
```

For decimal data types, the result's precision and scale are (P, s), where P is the maximum precision in the database or the Integration service, and s is the scale of the operand.

Null test operators: IS [NOT] NULL

These operators allow expressions to be tested for NULL values. For example, the expression:

```
column IS NULL
```

evaluates to True if and only if column has the value NULL.

Logical operators: AND, OR, NOT

These operators implement Boolean conjunction, disjunction, and negation, respectively. AND and OR take two Boolean operands each, while NOT takes a single operand. All return Boolean values.

For AND and OR, both operands may be evaluated even if one operand is undefined, particularly in queries against multiple databases. For example, the clause:

```
WHERE QUANTITY <> 0 AND TOTALCOST / QUANTITY > 50
```

may result in an error for rows where QUANTITY = 0.

String functions and operators

Actuate SQL supports the following string functions and operators:

- Case conversion functions: UPPER, LOWER
- Concatenation operator: ||
- Length function: CHAR_LENGTH
- LIKE operator
- Substring functions: LEFT, RIGHT, SUBSTRING
- Trimming functions: LTRIM, RTRIM, TRIM
- Search function: POSITION

Case conversion functions: UPPER, LOWER

These functions return a string formed by converting the characters in the argument to uppercase or lowercase respectively, provided the character is alphabetic:

```
Varchar UPPER( value Varchar )  
Varchar LOWER( value Varchar )
```

Examples The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME,  
       UPPER(CUSTOMERS.CUSTOMNAME)  
FROM "..Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
```

returns:

```
101,Signal Engineering,SIGNAL ENGINEERING  
109,InfoEngineering,INFOENGINEERING  
111,Advanced Design Inc.,ADVANCED DESIGN INC.  
...
```

The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME,  
       LOWER(CUSTOMERS.CUSTOMNAME)  
FROM "..Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
```

returns:

```
101,Signal Engineering,signal engineering  
109,InfoEngineering,infoengineering  
111,Advanced Design Inc.,advanced design inc.  
...
```

Concatenation operator: ||

This operator concatenates two string values, returning a new string that contains the characters from the left operand followed by the characters from the right operand.

Length function: CHAR_LENGTH

This function computes the length of a string, returning an integer count of its characters. Trailing spaces are significant:

```
Integer CHAR_LENGTH( value Varchar )
```

Example The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CONTACT_FIRST  
FROM "..Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS  
WHERE CHAR_LENGTH(CUSTOMERS.CONTACT_FIRST) > 5
```


returns:

```
102,Leslie
109,Michael
116,William
...
```

LIKE operator

The LIKE operator is used in an expression such as:

```
column LIKE 'Mar%'
```

In this example, values of column, such as Mary or Martin, satisfy the test because both start with Mar.

A LIKE operator pattern must be a literal string, for example, 'abc%', a parameter, or an expression. The LIKE operator does not support column references, subqueries, or aggregate expressions. Other examples include:

```
column LIKE :paramState
column LIKE CURRENT_USER( )
```

The following rules apply:

- Literal pattern characters must match exactly. LIKE is case-sensitive.
- An underscore character (_) matches any single character.
- A percent character (%) matches zero or more characters.

Escape a literal underscore, percent, or backslash character with a backslash character (\). Alternatively, use the following syntax:

```
test_string LIKE pattern_string ESCAPE escape_character
```

The escape character must obey the same rules as the LIKE operator pattern.

Substring functions: LEFT, RIGHT, SUBSTRING

These functions transform a string by retrieving a subset of its characters.

LEFT and RIGHT return the leftmost or rightmost n characters, respectively. Each takes the string as the first argument and the number of characters to retrieve as the second argument:

```
Varchar LEFT( value Varchar, offset Integer )
Varchar RIGHT( value Varchar, offset Integer )
```

Specifying an offset that is less than zero results in an error. If the offset is greater than the length of the string, these functions return the entire string.

SUBSTRING takes three arguments: the input string, the start position (one-based offset from the left side), and the number of characters to retrieve. It returns the substring located at this position:

Varchar **SUBSTRING**(input Varchar, start Integer, length Integer)

The following actions result in an error:

- Specifying a start position that is less than or equal to zero
- Specifying a length that is less than zero

Examples The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
WHERE LEFT(CUSTOMERS.CUSTOMNAME, 4) = 'Info'
```

returns:

```
109,InfoEngineering
117,InfoDesign
129,InfoSpecialists
...
```

The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
WHERE RIGHT(CUSTOMERS.CUSTOMNAME, 5) = 'Corp.'
```

returns:

```
104,SigniSpecialists Corp.
115,Design Solutions Corp.
118,Computer Systems Corp.
...
```

The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME,
       SUBSTRING(CUSTOMERS.CUSTOMNAME, 2, 5)
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS
```

returns:

```
101,Signal Engineering,ignal
102,Technical Specialists Co.,echni
104,SigniSpecialists Corp.,igniS
...
```

Trimming functions: LTRIM, RTRIM, TRIM

These functions strip space characters from a string. LTRIM strips only from the left side, RTRIM only from the right side, and TRIM from both sides. In all cases

the result value is a string identical to the argument except for the possible removal of space characters from either side. Other white space characters, including tabs and newlines, are not removed by these functions:

```
Varchar LTRIM( value Varchar )  
Varchar RTRIM( value Varchar )  
Varchar TRIM( value Varchar )
```

Examples The following code:

```
SELECT LTRIM(' Title '), 'Author'
```

returns:

```
Title , Author
```

The following code:

```
SELECT RTRIM(' Title '), 'Author'
```

returns:

```
Title, Author
```

The following code:

```
SELECT TRIM(' Title '), 'Author'
```

returns:

```
Title, Author
```

Search function: POSITION

The POSITION function takes two arguments: a substring and a search string. The POSITION function returns the position of the substring in the search string as an integer or as 0 if the substring is not found. If the substring is the empty string, the POSITION function returns 1. The POSITION function is case-sensitive:

```
Integer POSITION( substring Varchar, searchstring Varchar )
```

Example The following code:

```
SELECT CUSTOMERS.CUSTID, CUSTOMERS.CUSTOMNAME  
FROM "../Data Sources/MyDatabase/CUSTOMERS.SMA" CUSTOMERS  
WHERE POSITION('Inc.', CUSTOMERS.CUSTOMNAME) > 0
```

returns:

```
106, Technical MicroSystems Inc.  
111, Advanced Design Inc.  
113, Technical Design Inc.  
...
```

Timestamp functions

These functions perform operations on timestamp values:

- `CURRENT_TIMESTAMP`
- `CURRENT_DATE`
- `DATEADD`
- `DATEDIFF`
- `DATEPART`
- `DATESERIAL`

When using these functions, use the control strings listed in Table 6-9 to represent units of time. The control string used in a function must be a literal string, not an expression or a parameter.

Table 6-9 Control strings for various units of time

Unit of time	Control string
year	yyyy
quarter	q
month	m
day	d
day of year	y
day of week	w
hour	h
minute	n
second	s

CURRENT_TIMESTAMP

`CURRENT_TIMESTAMP` returns a timestamp value for the current date and time:

Timestamp `CURRENT_TIMESTAMP()`

Example The following code:

```
SELECT CURRENT_TIMESTAMP()
```

returns:

```
2004-10-27 14:49:23.0
```

CURRENT_DATE

CURRENT_DATE returns a timestamp value for the current date with the time set to 00:00:00.0:

```
Timestamp CURRENT_DATE()
```

Example The following code:

```
SELECT CURRENT_DATE()
```

returns:

```
2004-10-27 00:00:00.0
```

DATEADD

DATEADD takes three arguments: a control string, an integer delta value, and a timestamp value. It returns a timestamp that applies the delta value to the specified part of the original timestamp. The operation carries if the sum of the original field value and the delta is illegal:

```
Timestamp DATEADD( control Varchar, delta Integer,  
                  value Timestamp )
```

Example The following code:

```
SELECT ORDERS.ORDERID, ORDERS.SHIPBYDATE,  
       DATEADD('d', 14, ORDERS.SHIPBYDATE) AS ExpectedDelivery  
FROM "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS
```

returns:

```
1645,1995-05-22 00:00:00.0,1995-06-05 00:00:00.0  
1340,1995-06-03 00:00:00.0,1995-06-17 00:00:00.0  
1810,1995-04-12 00:00:00.0,1995-04-26 00:00:00.0  
...
```

DATEDIFF

DATEDIFF takes three arguments: a control string, a start timestamp, and an end timestamp. It returns the integer delta between the part of the two timestamps specified by the control string. Components smaller than the control string are ignored. Components larger than the control string contribute to the result:

```
Integer DATEDIFF( control Varchar, start Timestamp,  
                 end Timestamp )
```

Examples The following code:

```
SELECT ORDERS.ORDERID, ORDERS.SHIPBYDATE, ORDERS.FORECASTSHIPDATE,  
       DATEDIFF('d', ORDERS.SHIPBYDATE, ORDERS.FORECASTSHIPDATE) AS  
       ShipDateDifference  
FROM "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS
```

returns:

```
1645,1995-05-22 00:00:00.0,1995-06-02 00:00:00.0,11
1340,1995-06-03 00:00:00.0,1995-06-10 00:00:00.0,7
1810,1995-04-12 00:00:00.0,1995-04-27 00:00:00.0,15
...
```

The following expression:

```
DATEDIFF('d', CAST('2005-12-31 23:59:59.0' AS TIMESTAMP),
        CAST('2006-01-01 00:00:00.0' AS TIMESTAMP))
```

returns 1. The control string d indicates that the difference is in days. The difference between December 31, 2005 and January 1, 2006 is one day. The hours, minutes, and seconds components are ignored.

The following expression:

```
DATEDIFF('m', CAST('2005-12-31 23:59:59.0' AS TIMESTAMP),
        CAST('2006-01-01 00:00:00.0' AS TIMESTAMP))
```

returns 1. The control string m indicates that the difference is in months. The difference between December 31, 2005 and January 1, 2006 is one month. The day, hours, minutes, and seconds components are ignored.

DATEPART

DATEPART takes two arguments: a control string and a timestamp. It returns the part of the timestamp specified by the control string:

```
Integer DATEPART( control Varchar, value Timestamp )
```

Example The following code:

```
SELECT ORDERS.ORDERID, ORDERS.SHIPBYDATE
FROM "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS
WHERE DATEPART('m', ORDERS.SHIPBYDATE) = 5
```

returns:

```
1645,1995-05-22 00:00:00.0
1725,1995-05-10 00:00:00.0
1125,1995-05-03 00:00:00.0
...
```

DATESERIAL

DATESERIAL has two forms. The first form takes three arguments: a year value, a month value, and a day value. It returns a timestamp for the date corresponding to the specified year, month, and day with the time set to 00:00:00.0:

```
Timestamp DATESERIAL( year Integer, month Integer, day Integer )
```

The second form of DATESERIAL takes six arguments: values for the year, month, day, hour, minute, and second. It returns the timestamp for the specified values:

```
Timestamp DATESERIAL( year Integer, month Integer, day Integer,  
                        hour Integer, minute Integer, second Integer )
```

Example The following code:

```
SELECT ORDERS.ORDERID, ORDERS.ASKBYDATE  
FROM "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS  
WHERE ORDERS.ASKBYDATE >= DATESERIAL(1995, 6, 15, 12, 59, 59)
```

returns:

```
1555,1995-06-28 00:00:00.0  
1725,1995-06-23 00:00:00.0  
1720,1995-06-17 00:00:00.0  
...
```

Aggregate functions: COUNT, MIN, MAX, SUM, AVG

These functions aggregate an entire column of values into a single scalar result. For decimal data types:

- For the MIN, MAX, and AVG functions, the result's precision and scale are the same as the precision and scale of the operand.
- For the SUM function, the result's precision and scale are (P, s), where P is the maximum precision in the database or the Integration service, and s is the scale of the operand.

The COUNT function reduces any argument type to a single integer representing the number of non-NULL items. As in SQL-92, COUNT(*) counts the number of rows in a table:

```
Integer COUNT( column )
```

Example The following code:

```
SELECT COUNT(ORDERS.ORDERID) AS NumberOfOrders  
FROM "../Data Sources/MyDatabase/ORDERS.SMA" ORDERS
```

returns:

```
111
```

MIN and MAX accept any type and return the minimum or maximum value, using the same rules that apply to comparison of individual items:

```
ColumnType MIN( column )  
ColumnType MAX( column )
```

Examples The following code:

```
SELECT MIN (ITEMS.QUANTITY)
FROM "../Data Sources/MyDatabase/ITEMS.SMA" ITEMS
```

returns:

2

The following code:

```
SELECT MAX (ITEMS.QUANTITY)
FROM "../Data Sources/MyDatabase/ITEMS.SMA" ITEMS
```

returns:

6203

SUM and AVG can be applied to any of the three numeric types and produce the sum or average of all the numbers:

ColumnType **SUM**(*column*)

ColumnType **AVG**(*column*)

Examples The following code:

```
SELECT SUM (ITEMS.QUANTITY)
FROM "../Data Sources/MyDatabase/ITEMS.SMA" ITEMS
```

returns:

606177

The following code:

```
SELECT AVG (ITEMS.QUANTITY)
FROM "../Data Sources/MyDatabase/ITEMS.SMA" ITEMS
```

returns:

319

System function: CURRENT_USER

CURRENT_USER returns a string containing the user name for the current Encyclopedia volume user:

Varchar **CURRENT_USER**()

Example The following code:

```
SELECT CURRENT_USER()
```

returns:

user1

Providing query optimization hints

A report developer or business user uses an information object to create an Actuate SQL query. When you create the information object, you can provide hints that help to optimize the query. Specifically, you can:

- Indicate that a table in a join is optional.
- Specify the cardinality of a join.

For query optimization hints to take effect, you must create join conditions with the ON clause, not the WHERE clause.

Indicating that a table in a join is optional

When you create an information object, you indicate that a table in a join is optional using the OPTIONAL keyword. If you indicate that a table is optional and none of its columns appear in the query created by a report developer or business user (except in a join condition), the table is dropped from the optimized query.

The OPTIONAL keyword has no effect in queries created in the Information Object Query Builder.

For example, consider the following information object CustomersOrders:

```
SELECT Customers.custid, Customers.customname,  
       Customers.contact_last, Orders.orderid, Orders.custid,  
       Orders.amount, Orders.shipbydate  
FROM Customers.sma LEFT OPTIONAL INNER JOIN Orders.sma  
ON (Customers.custid = Orders.custid)
```

Now consider the following Actuate SQL query created by a report developer or business user using CustomersOrders:

```
SELECT Orders.custid, Orders.orderid, Orders.amount  
FROM CustomersOrders.iob  
WHERE Orders.amount BETWEEN 10000 and 20000
```

Because no column from the Customers table appears in the query, and because the join in CustomersOrders includes the LEFT OPTIONAL keywords, the Customers table is dropped from the optimized query:

```
SELECT Orders.custid, Orders.orderid, Orders.amount  
FROM Orders.sma  
WHERE Orders.amount BETWEEN 10000 and 20000
```

Now consider another Actuate SQL query created by a report developer or business user using CustomersOrders:

```
SELECT Customers.custid, Customers.contact_last
FROM CustomersOrders.iob
WHERE Customers.city = 'NYC'
```

No column from the Orders table appears in the query. But because the Orders table is not optional, it is not dropped from the query:

```
SELECT Customers.custid, Customers.contact_last
FROM Customers.sma INNER JOIN Orders.sma
ON (Customers.custid = Orders.custid)
WHERE Customers.city = 'NYC'
```

If you use the OPTIONAL keyword without the LEFT or RIGHT qualifier, it applies to both tables in the join.

The OPTIONAL keyword is ignored when it applies to:

- A table whose columns appear in the query created by a report developer or business user, for example in the SELECT list or in the ORDER BY, GROUP BY, HAVING, or WHERE clauses.
- The middle table in an information object, for example:

```
SELECT Customers.custid, Items.orderid, Items.itemcode,
       Items.description
FROM Customers RIGHT OPTIONAL INNER JOIN Orders
ON (Customers.custid = Orders.custid)
LEFT OPTIONAL INNER JOIN Items ON (Orders.orderid =
       Items.orderid)
```

In this information object, Orders is the middle table.

An information object that uses the OPTIONAL keyword cannot be joined to another information object. Therefore, an Actuate SQL query created by a report developer or business user cannot include more than one information object if that information object uses the OPTIONAL keyword.

Using the OPTIONAL keyword with a computed field

Do not define a computed field in an information object that contains the OPTIONAL keyword. Instead, define the computed field in a lower level information object.

For example, consider the information object MyInformationObject:

```
SELECT dbo_CUSTOMERS.CUSTID AS CUSTID, dbo_CUSTOMERS.CONTACT_FIRST
       AS CONTACT_FIRST, dbo_CUSTOMERS.CONTACT_LAST AS CONTACT_LAST,
       dbo_CUSTOMERS.CITY AS CITY, dbo_ORDERS.SHIPBYDATE AS
       SHIPBYDATE, dbo_ORDERS.FORECASTSHIPDATE AS FORECASTSHIPDATE,
```

```

        dbo_CUSTOMERS.ADDRESS AS ADDRESS,
        ( dbo_ITEMS.PRICEQUOTE * dbo_ITEMS.QUANTITY ) AS Total
FROM "dbo.CUSTOMERS.sma" AS dbo_CUSTOMERS
OPTIONAL INNER JOIN "dbo.ORDERS.sma" AS dbo_ORDERS
ON ( dbo_CUSTOMERS.CUSTID=dbo_ORDERS.CUSTID )
OPTIONAL INNER JOIN "dbo.ITEMS.sma" AS dbo_ITEMS
ON ( dbo_ORDERS.ORDERID=dbo_ITEMS.ORDERID )

```

MyInformationObject defines the computed field Total and also contains the OPTIONAL keyword.

Now consider the following Actuate SQL query created by a report developer or business user using MyInformationObject:

```

SELECT MyInformationObject.CUSTID AS CUSTID,
       MyInformationObject.CONTACT_FIRST AS CONTACT_FIRST,
       MyInformationObject.CITY AS CITY,
       MyInformationObject.CONTACT_LAST AS CONTACT_LAST
FROM "MyInformationObject.iob" AS MyInformationObject

```

The ORDERS and ITEMS tables are not dropped from the query even though the OPTIONAL keyword is applied to both tables in MyInformationObject and the SELECT clause does not contain columns from either table. The tables are not dropped because in MyInformationObject the columns ITEMS.PRICEQUOTE and ITEMS.QUANTITY are used in a computation outside the join condition.

To avoid this situation, define the computed field in a lower level information object such as ITEMS.iob. MyInformationObject then contains the following query:

```

SELECT dbo_CUSTOMERS.CUSTID AS CUSTID, dbo_CUSTOMERS.CONTACT_FIRST
       AS CONTACT_FIRST, dbo_CUSTOMERS.CONTACT_LAST AS CONTACT_LAST,
       dbo_CUSTOMERS.CITY AS CITY, dbo_ORDERS.SHIPBYDATE AS
       SHIPBYDATE, dbo_ORDERS.FORECASTSHIPDATE AS FORECASTSHIPDATE,
       dbo_CUSTOMERS.ADDRESS AS ADDRESS, ITEMS.Total AS Total
FROM "dbo.CUSTOMERS.sma" AS dbo_CUSTOMERS
OPTIONAL INNER JOIN "dbo.ORDERS.sma" AS dbo_ORDERS
ON ( dbo_CUSTOMERS.CUSTID=dbo_ORDERS.CUSTID )
OPTIONAL INNER JOIN "ITEMS.iob" AS ITEMS
ON ( dbo_ORDERS.ORDERID=ITEMS.ORDERID )

```

Using the OPTIONAL keyword with parentheses ()

You can control the processing of the OPTIONAL keyword with parentheses. For example, in the following query the tables CUSTOMERS and ORDERS can be dropped:

```

SELECT ITEMS.ORDERID, ITEMS.PRICEQUOTE, ITEMS.QUANTITY
FROM "CUSTOMERS.sma" AS CUSTOMERS INNER JOIN "ORDERS.sma" AS
ORDERS ON (CUSTOMERS.CUSTID = ORDERS.CUSTID) LEFT OPTIONAL
INNER JOIN "ITEMS.sma" AS ITEMS ON
(ORDERS.ORDERID = ITEMS.ORDERID)

```

In the following query, however, only the ORDERS table can be dropped because the join that includes the LEFT OPTIONAL keywords is enclosed in parentheses:

```
SELECT ITEMS.ORDERID, ITEMS.PRICEQUOTE, ITEMS.QUANTITY
FROM "CUSTOMERS.sma" AS CUSTOMERS INNER JOIN ("ORDERS.sma" AS
ORDERS LEFT OPTIONAL INNER JOIN "ITEMS.sma" AS ITEMS ON
(ORDERS.ORDERID = ITEMS.ORDERID) ) ON
(CUSTOMERS.CUSTID = ORDERS.CUSTID)
```

In the following examples, A, B, C, and D are tables.

Consider the following query that includes the RIGHT OPTIONAL keywords:

```
A RIGHT OPTIONAL JOIN B RIGHT OPTIONAL JOIN C RIGHT OPTIONAL
JOIN D
```

The Actuate SQL compiler interprets this query as:

```
((A RIGHT OPTIONAL JOIN B) RIGHT OPTIONAL JOIN C)
RIGHT OPTIONAL JOIN D
```

Tables B, C, and D can be dropped from the query.

Consider the following query that includes the LEFT OPTIONAL keywords without parentheses:

```
A LEFT OPTIONAL JOIN B LEFT OPTIONAL JOIN C LEFT OPTIONAL JOIN D
```

The Actuate SQL compiler interprets this query as:

```
((A LEFT OPTIONAL JOIN B) LEFT OPTIONAL JOIN C) LEFT OPTIONAL
JOIN D
```

Tables A, B, and C can be dropped from the query. It is not possible, however, to drop table C without dropping tables A and B, or to drop table B without dropping table A, without using parentheses.

Consider the following query that includes the LEFT OPTIONAL keywords with parentheses:

```
A LEFT OPTIONAL JOIN (B LEFT OPTIONAL JOIN (C LEFT OPTIONAL
JOIN D) )
```

Table C can be dropped from the query without dropping tables A and B. Table B can be dropped from the query without dropping table A.

Consider the following query that includes the OPTIONAL keyword without the LEFT or RIGHT modifier:

```
A OPTIONAL JOIN B OPTIONAL JOIN C OPTIONAL JOIN D
```

The Actuate SQL compiler interprets this query as:

```
((A OPTIONAL JOIN B) OPTIONAL JOIN C) OPTIONAL JOIN D
```

Any table or set of tables can be dropped from the query.

Using the OPTIONAL keyword with aggregate functions

If a query created by a report developer or business user contains the function COUNT(*), the OPTIONAL keyword, if it appears in the information object, is ignored. If a query contains another aggregate function, for example SUM or COUNT(column), the value returned by the aggregate function depends on whether the information object includes the OPTIONAL keyword. For example, consider the following Actuate SQL query created by a report developer or business user using the CustomersOrders information object:

```
SELECT COUNT(Customers.custid) AS CustomerCount
FROM CustomersOrders.iob
```

In the first case, consider the following information object CustomersOrders, which applies the OPTIONAL keyword to the Orders table:

```
SELECT Customers.custid, Customers.customname,
       Customers.contact_last, Orders.orderid, Orders.custid,
       Orders.amount, Orders.shipbydate
FROM Customers.sma RIGHT OPTIONAL INNER JOIN Orders.sma
ON (Customers.custid = Orders.custid)
```

Because no column from the Orders table appears in the query and because the join in CustomersOrders includes the RIGHT OPTIONAL keywords, the Orders table is dropped from the optimized query:

```
SELECT COUNT(Customers.custid) AS CustomerCount
FROM Customers.sma
```

In the second case, consider the following information object CustomersOrders, which does not apply the OPTIONAL keyword to the Orders table:

```
SELECT Customers.custid, Customers.customname,
       Customers.contact_last, Orders.orderid, Orders.custid,
       Orders.amount, Orders.shipbydate
FROM Customers.sma INNER JOIN Orders.sma
ON (Customers.custid = Orders.custid)
```

In this case, the Orders table is not dropped from the query:

```
SELECT COUNT(Customers.custid) AS CustomerCount
FROM Customers.sma INNER JOIN Orders.sma
ON (Customers.custid = Orders.custid)
```

The value of CustomerCount depends on whether the OPTIONAL keyword is applied to the Orders table in the CustomersOrders information object.

Specifying the cardinality of a join

You can specify the right-to-left and left-to-right cardinality of a join. Table 6-10 lists the cardinality types and a description of each type.

Table 6-10 Cardinality types

Cardinality type	Description
1	One record in the first table matches one record in the second table.
?	One record in the first table matches zero or one record in the second table.
*	One record in the first table matches zero or more records in the second table.
+	One record in the first table matches one or more records in the second table.

The right-to-left cardinality type is followed by a hyphen (-), and then by the left-to-right cardinality type. The cardinality type depends on the join column.

For example:

```
Customers JOIN Orders ON (Customers.custid = Orders.custid)
  {CARDINALITY ('1-+')}
```

indicates that:

- One record in Orders matches one record in Customers.
- One record in Customers matches one or more records in Orders:

```
Customers JOIN Orders ON (Customers.custid = Orders.custid)
  {CARDINALITY ('1-*')}
```

indicates that:

- One record in Orders matches one record in Customers.
- One record in Customers matches zero or more records in Orders:

```
Customers JOIN Orders ON (Customers.custid = Orders.custid)
  {CARDINALITY ('*-?')}
```

indicates that:

- One record in Orders matches zero or more records in Customers.
- One record in Customers matches zero or one record in Orders.

Using pragmas to tune a query

If an information object query joins maps or information objects that are based on different data sources, you may be able to tune the query using the following pragmas:

- EnableCBO
- applyIndexing
- MinRowsForIndexing

These pragmas are described in the following topics.

Disabling cost-based optimization

If you provide values for the map and join column properties, the Actuate SQL compiler uses these values to do cost-based query optimization. You can disable cost-based optimization using the pragma EnableCBO.

For example, consider the following query based on SQL Server and Oracle database tables:

```
SELECT
    NATION.N_NAME,
    SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT))
        AS Revenue
FROM
    "/SQL_Server/CUSTOMER.SMA" CUSTOMER,
    "/Oracle/ORDERS.SMA" ORDERS,
    "/SQL_Server/LINEITEM.SMA" LINEITEM,
    "/SQL_Server/SUPPLY.SMA" SUPPLY,
    "/Oracle/NATION.SMA" NATION,
    "/Oracle/REGION.SMA" REGION
WHERE
    CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
    AND LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
    AND LINEITEM.L_SUPPKEY = SUPPLY.S_SUPPKEY
    AND CUSTOMER.C_NATIONKEY = SUPPLY.S_NATIONKEY
    AND SUPPLY.S_NATIONKEY = NATION.N_NATIONKEY
    AND NATION.N_REGIONKEY = REGION.R_REGIONKEY
    AND REGION.R_NAME = 'ASIA'
    AND ORDERS.O_ORDERDATE >= TIMESTAMP '1993-01-01 00:00:00'
    AND ORDERS.O_ORDERDATE < TIMESTAMP '1994-01-01 00:00:00'
GROUP BY
    NATION.N_NAME
```

If you provide values for the map and join column properties, part of the query plan looks similar to Figure 6-1.

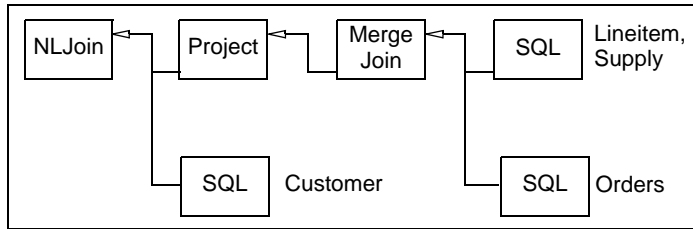


Figure 6-1 Example of part of the query plan for which values for the map and join column properties have been provided

To disable cost-based optimization for the query, set the pragma EnableCBO to False:

```
PRAGMA "EnableCBO" := 'false'
SELECT
  NATION.N_NAME,
  SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT))
    AS Revenue
FROM
  "/SQL_Server/CUSTOMER.SMA" CUSTOMER,
  "/Oracle/ORDERS.SMA" ORDERS,
  "/SQL_Server/LINEITEM.SMA" LINEITEM,
  "/SQL_Server/SUPPLY.SMA" SUPPLY,
  "/Oracle/NATION.SMA" NATION,
  "/Oracle/REGION.SMA" REGION
WHERE
  ...
```

Now this part of the query plan looks similar to Figure 6-2.

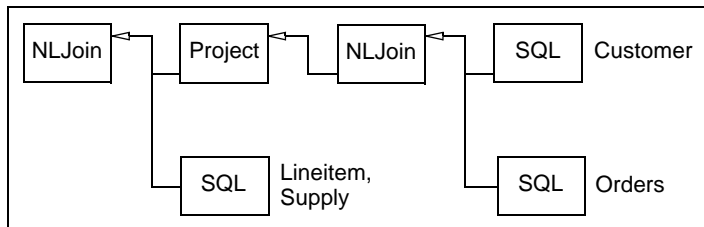


Figure 6-2 Example of part of a query plan with cost-based optimization disabled

Disabling cost-based optimization changes the join sequence and the join algorithm. The Customer and LineItem, Supply database subqueries switch positions, and the merge join is replaced with a nested loop join.

If you create a query using an information object for which cost-based optimization is disabled, cost-based optimization is disabled for the query as well.

You can disable cost-based optimization for all information object queries by setting the BIRT iHub configuration variable Enable cost based optimization to False. For more information about BIRT iHub configuration variables, see *Configuring BIRT iHub*.

Disabling indexing

By default, the Actuate SQL compiler creates indexes for rows that are materialized in memory during query execution, for example the rows returned when the right side of a nested loop join is executed. You can disable indexing using the pragma `applyIndexing`.

For example, to disable indexing for a query, set the pragma `applyIndexing` to False:

```
PRAGMA "applyIndexing" := 'false'
SELECT
    NATION.N_NAME,
    SUM (LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT))
        AS Revenue
FROM
    "/SQL_Server/CUSTOMER.SMA" CUSTOMER,
    "/Oracle/ORDERS.SMA" ORDERS,
    "/SQL_Server/LINEITEM.SMA" LINEITEM,
    "/SQL_Server/SUPPLY.SMA" SUPPLY,
    "/Oracle/NATION.SMA" NATION,
    "/Oracle/REGION.SMA" REGION
WHERE
    ...
```

If you create a query using an information object for which indexing is disabled, indexing is disabled for the query as well.

Specifying a threshold value for indexing

If cost-based optimization is enabled and you provide values for the map and join column properties, the Actuate SQL compiler creates an index when 100 rows are materialized in memory during query execution. You can change the number of materialized rows that triggers indexing using the pragma `MinRowsForIndexing`.

If cost-based optimization is disabled, or you do not provide values for the map and join column properties, an index is created for materialized rows if a suitable column is available.

For example, to change the number of materialized rows that triggers indexing to 1000, set the pragma `MinRowsForIndexing` to 1000:

```
PRAGMA "MinRowsForIndexing" := '1000'
SELECT
    NATION.N_NAME,
    SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT))
    AS Revenue
FROM
    "/SQL_Server/CUSTOMER.SMA" CUSTOMER,
    "/Oracle/ORDERS.SMA" ORDERS,
    "/SQL_Server/LINEITEM.SMA" LINEITEM,
    "/SQL_Server/SUPPLY.SMA" SUPPLY,
    "/Oracle/NATION.SMA" NATION,
    "/Oracle/REGION.SMA" REGION
WHERE
    ...
```

Specifying the number of materialized rows that triggers indexing for an information object has no effect on queries that use the information object.

You can specify the number of materialized rows that triggers indexing for all information object queries by setting the BIRT iHub configuration variable `Minimum rows to trigger creation of an index during materialize operation`. For more information about BIRT iHub configuration variables, see *Configuring BIRT iHub*.

Part Two

Configuring database types

7

Understanding database types

This chapter contains the following topics:

- About database types
- About preconfigured database types
- About configurable database types
- Working with XML files

About database types

A database type refers to a connection type and a mapping. Several preconfigured database types are available for use with the IO Design perspective. You can also configure your own database types. You choose a database type when you create a data connection definition, as shown in Figure 7-1.

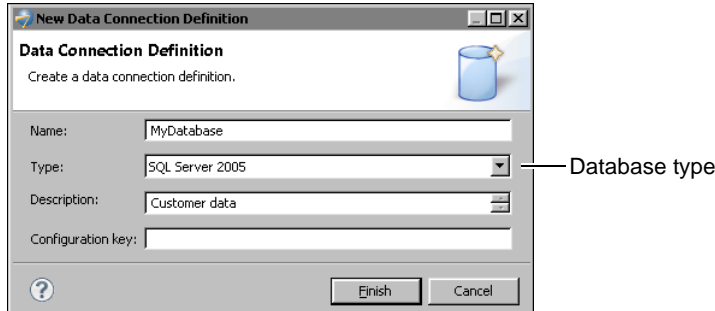


Figure 7-1 Choosing the database type for a data connection definition

About connection types

A connection type defines JDBC connection string syntax and connection parameters, for example user name and password. You provide values for connection parameters on the Data source connection properties page, as shown in Figure 7-2.

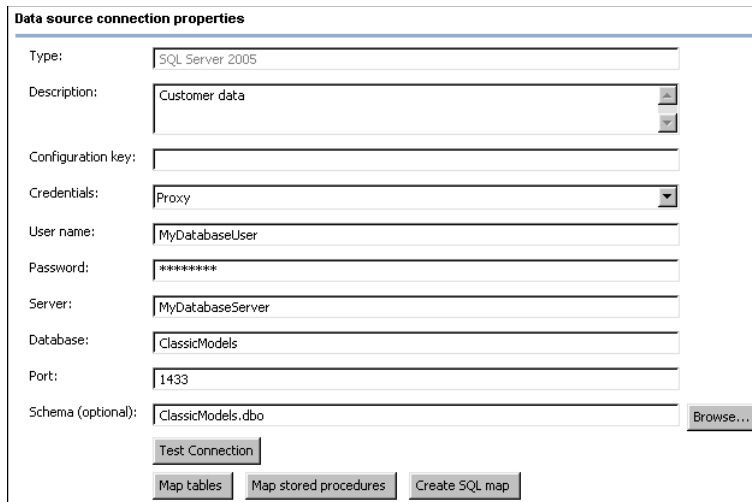


Figure 7-2 Providing values for the data source connection properties

About mappings

When the Integration service processes an Actuate SQL query, one of the following occurs:

- The entire query is executed by the database.
- Parts of the query are executed by the database, while other parts of the query are executed by the Integration service.
- The entire query is executed by the Integration service.

Figure 7-3 shows how the Integration service processes an Actuate SQL query when parts of the query are executed by the database, while other parts of the query are executed by the Integration service.

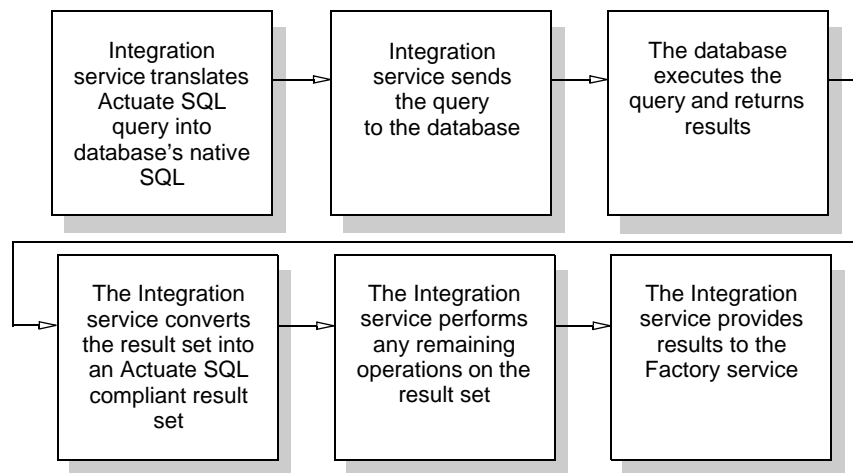


Figure 7-3 Actuate SQL query processing

You optimize performance by having the database execute as much of the query as possible. Mappings translate Actuate SQL into the database's native SQL and ensure that an Actuate SQL query returns the same results for every database type. For example, a DB2 database and an Oracle database have identical schemas. You create an Actuate SQL query using maps that represent tables from one of these databases. The query returns the same results whether the maps represent DB2 tables or Oracle tables.

Specifically, a mapping defines how to map Actuate SQL data types to a database's native SQL data types. A mapping also defines how to map the following Actuate SQL constructs to the corresponding native SQL constructs:

- Functions and operators
- Parameters

- Literals
- GROUP BY and ORDER BY clauses

About preconfigured database types

The IO Design perspective supports the following preconfigured database types:

- DB2
- Informix
- MySQL Enterprise
- Oracle
- PostgreSQL
- SQL Server
- Sybase

The connection type configuration and mappings files for preconfigured database types are located in the following iHub and BIRT Designer Professional directories:

```
$AC_SERVER_HOME/etc
```

```
<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>  
  \Config\aisconfigfiles\etc
```

The connection type configuration for preconfigured database types resides in the file `intsrvsources.xml`. Do not modify this file.

The mapping for a preconfigured database type resides in the `mappings.xml` file in the directory for that type. For example, Figure 7-4 shows the location of the `mappings.xml` file for the SQL Server database type.

The following topics explain how data types for preconfigured database types are mapped to Actuate SQL data types and vice versa. Some database data types cannot be mapped to an Actuate SQL data type, for example binary types with specially defined operations. These topics also discuss special considerations for each database type.

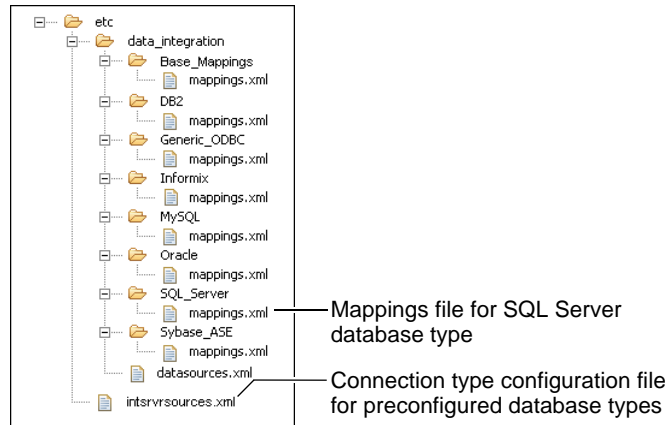


Figure 7-4 Location of the connection type configuration file and mappings files for preconfigured database types

DB2 data type mapping and issues

The following DB2 functions convert DECIMAL values to the DOUBLE data type. The corresponding Actuate SQL functions are implemented using these native functions, and may therefore return slightly inaccurate values, especially for calculations involving very large or very small numbers:

- POWER ()
- ROUND ()
- FLOOR ()
- CEILING ()

When a numeric type is cast to VARCHAR(n) and n is not large enough to accommodate the string, Actuate SQL returns an error. DB2, however, truncates the value without returning an error.

Table 7-1 shows how DB2 data types map to Actuate SQL data types.

Table 7-1 Mapping of DB2 data types to Actuate SQL data types

DB2 data type	Actuate SQL data type	Compiled to DB2 data type	DB2 data type limitations
BIGINT	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for DB2 DECIMAL is 31.
CHAR	VARCHAR	VARCHAR	DB2 VARCHAR has a maximum length of 32,672 characters.

(continues)

Table 7-1 Mapping of DB2 data types to Actuate SQL data types (continued)

DB2 data type	Actuate SQL data type	Compiled to DB2 data type	DB2 data type limitations
DATE	TIMESTAMP	DATETIME	
DECIMAL	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for DB2 DECIMAL is 31.
DOUBLE	DOUBLE	DOUBLE	
INTEGER	INTEGER	INTEGER	
REAL	DOUBLE	DOUBLE	
SMALLINT	INTEGER	INTEGER	
TIMESTAMP	TIMESTAMP	DATETIME	
VARCHAR	VARCHAR	VARCHAR	DB2 VARCHAR has a maximum length of 32,672 characters.

Informix data type mapping and issues

The NCHAR and NVARCHAR data types are not supported.

If a query contains a CASE statement that returns a string, Informix pads the string with spaces so that its length matches the length of the longest string in the CASE statement. For example, the following CASE statement returns 'O ' (O followed by two spaces), not 'O'. The string length matches the length of the longest string in the CASE statement, 'N/A':

```
SELECT
    CASE ORDERS.status
        WHEN 'Open' Then 'O'
        WHEN 'Closed' Then 'C'
        WHEN 'In Evaluation' Then 'E'
        ELSE 'N/A'
    END
    AS "Short Status", ORDERS.orderid, ORDERS.custid,
    ITEMS.quantity
FROM "../Data Sources/MyDatabase/ORDERS.sma" ORDERS
INNER JOIN "../Data Sources/MyDatabase/ITEMS.sma" ITEMS
    ON ORDERS.orderid = ITEMS.orderid
WHERE ORDERS.orderid < 1120 OR ORDERS.orderid > 2000
```

Table 7-2 shows how Informix data types map to Actuate SQL data types.

Table 7-2 Mapping of Informix data types to Actuate SQL data types

Informix data type	Actuate SQL data type	Compiled to Informix data type	Informix data type limitations
CHAR	VARCHAR	VARCHAR	Informix VARCHAR has a maximum length of 254 characters.
CHARACTER VARYING	VARCHAR	VARCHAR	Informix VARCHAR has a maximum length of 254 characters.
DATE	TIMESTAMP	DATETIME	
DATETIME	TIMESTAMP	DATETIME	
DECIMAL	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Informix DECIMAL is 32.
FLOAT	DOUBLE	FLOAT	Informix FLOAT is 4-byte, not 8-byte, floating point. The maximum number of significant digits (precision) is 16.
INT8	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Informix DECIMAL is 32.
INTEGER	INTEGER	INTEGER	
MONEY	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Informix DECIMAL is 32.
SMALLFLOAT	DOUBLE	FLOAT	Informix FLOAT is 4-byte, not 8-byte, floating point. The maximum number of significant digits (precision) is 16.
SMALLINT	INTEGER	INTEGER	
VARCHAR	VARCHAR	VARCHAR	Informix VARCHAR has a maximum length of 254 characters.

Oracle data type mapping and issues

Oracle treats zero-length VARCHAR2 and NVARCHAR2 values as NULL values. Oracle also treats NULL VARCHAR2 and NVARCHAR2 values as zero-length values. For this reason, Oracle queries may return different results than queries against other databases, for example:

- The CONCAT function, when concatenating a NULL value and a non-NULL value, returns a non-NULL value because the NULL value is treated as an empty string.
- Comparisons with empty strings using the operators =, <, >, and <> never evaluate to TRUE. For example, the expression:

```
'Hello' <> ''
```

does not evaluate to TRUE because Oracle treats " as a NULL value, and any comparison with NULL evaluates to UNKNOWN, not TRUE.

Actuate SQL uses SQL-92 semantics to perform VARCHAR string comparisons. In most cases, when a query is pushed to the database, the mappings.xml file preserves SQL-92 semantics. For Oracle databases, however, using SQL-92 semantics results in poor performance. For this reason, a query that is pushed to an Oracle database uses Oracle semantics to perform string comparisons. If you are working solely with Oracle databases, string comparisons yield consistent results. If you join a table in an Oracle database to a table in a database of another type, string comparisons may yield inconsistent results because the query pushed to the other database uses SQL-92 semantics. Moreover, if the query is not pushed to the database, the Integration service uses SQL-92 semantics. If you want Oracle databases to use SQL-92 semantics, you must use a different mappings.xml file. Contact Actuate Support to obtain this file.

Table 7-3 shows how Oracle data types map to Actuate SQL data types. The letters p and s represent precision and scale.

Table 7-3 Mapping of Oracle data types to Actuate SQL data types

Oracle data type	Actuate SQL data type	Compiled to Oracle data type	Oracle data type limitations
CHAR	VARCHAR	NVARCHAR2	Oracle NVARCHAR2 has a maximum length of 2000 bytes or characters.
DATE	TIMESTAMP	DATE (Oracle 8i) TIMESTAMP (Oracle 9i and later)	Oracle versions earlier than 9i do not support the TIMESTAMP data type. For those versions the milliseconds field of timestamp values is ignored in comparisons and sorting.
FLOAT	DOUBLE	FLOAT	Oracle FLOAT has a maximum precision of 38 decimal digits.
NCHAR	VARCHAR	NVARCHAR2	Oracle NVARCHAR2 has a maximum length of 2000 bytes or characters.
NUMBER(p,s)	INTEGER if s=0 and p<=9 DECIMAL if s<>0 or p>9	INTEGER DECIMAL	Oracle supports only the NUMBER and FLOAT data types for internal storage. INTEGER and DECIMAL data types are provided to support queries written with standard SQL types. The maximum number of significant digits (precision) for Oracle DECIMAL is 38.
NVARCHAR2	VARCHAR	NVARCHAR2	Oracle NVARCHAR2 has a maximum length of 2000 bytes or characters.
TIMESTAMP	TIMESTAMP	TIMESTAMP	

Table 7-3 Mapping of Oracle data types to Actuate SQL data types

Oracle data type	Actuate SQL data type	Compiled to Oracle data type	Oracle data type limitations
VARCHAR2	VARCHAR	NVARCHAR2	Oracle NVARCHAR2 has a maximum length of 2000 bytes or characters.

SQL Server data type mapping and issues

The behavior of the Actuate SQL DatePart(), DateAdd(), and DateDiff() functions sometimes differs from the behavior of the corresponding Transact-SQL functions.

If you want to use the Actuate collation UNICODE_BIN against a SQL Server database that uses a collation other than unicode_bin, you must install the SQL_Latin1_General_Cp850_BIN2 collation. This installation requires HotFix 816039 from Microsoft. For information about HotFix 816039, go to:

<http://support.microsoft.com/?id=816039>

Using parameters in a GROUP BY clause may result in a SQL Server database error. For example, a query of the form:

```
SELECT CUSTID+? FROM CUSTOMERS
GROUP BY CUSTID+?
```

results in the following error:

[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'EIITESTDB.dbo.CUSTOMERS.CUSTID' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause

SQL Server considers the two parameters to be different since they are not named parameters. The solution is to include a reference to the CUSTID column in the GROUP BY clause.

Table 7-4 shows how Transact-SQL data types map to Actuate SQL data types.

Table 7-4 Mapping of Transact-SQL data types to Actuate SQL data types

Transact-SQL data type	Actuate SQL data type	Compiled to Transact-SQL data type	Transact-SQL data type limitations
CHAR	VARCHAR	NVARCHAR	Transact-SQL NVARCHAR has a maximum length of 4000 characters on SQL Server.

(continues)

Table 7-4 Mapping of Transact-SQL data types to Actuate SQL data types (continued)

Transact-SQL data type	Actuate SQL data type	Compiled to Transact-SQL data type	Transact-SQL data type limitations
DATETIME	TIMESTAMP	DATETIME	Transact-SQL DATETIME stores values from January 1, 1753. The accuracy of dates is to one three-hundredths of a second (3.33 milliseconds).
DECIMAL	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Transact-SQL DECIMAL is 38.
FLOAT	DOUBLE	FLOAT	
INT	INTEGER	INTEGER	
MONEY	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Transact-SQL DECIMAL is 38.
NCHAR	VARCHAR	NVARCHAR	Transact-SQL NVARCHAR has a maximum length of 4000 characters on SQL Server.
NVARCHAR	VARCHAR	NVARCHAR	Transact-SQL NVARCHAR has a maximum length of 4000 characters on SQL Server.
REAL	DOUBLE	FLOAT	
SMALLDATE TIME	TIMESTAMP	DATETIME	Transact-SQL DATETIME stores values from January 1, 1753. The accuracy of dates is to one three-hundredths of a second (3.33 milliseconds).
SMALLINT	INTEGER	INTEGER	
SMALLMONEY	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Transact-SQL DECIMAL is 38.
TINYINT	INTEGER	INTEGER	
VARCHAR	VARCHAR	NVARCHAR	Transact-SQL NVARCHAR has a maximum length of 4000 characters on SQL Server.

Sybase data type mapping and issues

Sybase table and column names must not exceed 28 characters.

When using Sybase, an expression in a WHERE clause can contain the power function and a decimal value with up to five digits in the fractional part of the value. The following WHERE clause shows this type of expression:

```
WHERE power (ACNULLDATATYPES.ACDECIMAL, 2) > 1.12345
```

Do not use expressions in the WHERE clause that have a power function and a decimal value with six or more digits in the fractional part of the value. For example, the following WHERE clause causes an error:

```
WHERE power (ACNULLDATATYPES.ACDECIMAL, 2) > 1.123456
```

Table 7-5 shows how Sybase data types map to Actuate SQL data types.

Table 7-5 Mapping of Sybase data types to Actuate SQL data types

Sybase data type	Actuate SQL data type	Compiled to Sybase data type	Sybase data type limitations
CHAR	VARCHAR	NVARCHAR	Sybase NVARCHAR has a maximum length of 255 characters.
DATETIME	TIMESTAMP	DATETIME	Sybase DATETIME stores values from January 1, 1753. The accuracy of dates is to one three-hundredths of a second (3.33 milliseconds).
DECIMAL	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Sybase DECIMAL is 38.
DOUBLE PRECISION	DOUBLE	FLOAT	The maximum number of significant digits (precision) for Transact-SQL DECIMAL is 38.
FLOAT	DOUBLE	FLOAT	
INT	INTEGER	INTEGER	
MONEY	DECIMAL	DECIMAL	
NCHAR	VARCHAR	NVARCHAR	Sybase NVARCHAR has a maximum length of 255 characters.
NVARCHAR	VARCHAR	NVARCHAR	Sybase NVARCHAR has a maximum length of 255 characters.
REAL	DOUBLE	FLOAT	

(continues)

Table 7-5 Mapping of Sybase data types to Actuate SQL data types (continued)

Sybase data type	Actuate SQL data type	Compiled to Sybase data type	Sybase data type limitations
SMALLDATETIME	TIMESTAMP	DATETIME	Sybase DATETIME stores values from January 1, 1753. The accuracy of dates is to one three-hundredths of a second (3.33 milliseconds).
SMALLINT	INTEGER	INTEGER	
SMALLMONEY	DECIMAL	DECIMAL	The maximum number of significant digits (precision) for Sybase DECIMAL is 38.
TINYINT	INTEGER	INTEGER	
VARCHAR	VARCHAR	NVARCHAR	Sybase NVARCHAR has a maximum length of 255 characters.

About configurable database types

If the IO Design perspective does not provide a preconfigured database type for a database you want to use, you can configure your own database type. The connection type configuration and mappings files for configurable database types are located in the following iHub and BIRT Designer Professional directories:

```
$AC_SERVER_HOME/etc/data_integration
```

```
<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>  
  \Config\aisconfigfiles\etc\data_integration
```

The connection type configuration and mappings files for configurable database types on iHub and on the desktop must be identical.

You can use the default mappings file if both of the following statements are true:

- Your database implementation adheres closely to the SQL-92 standard.
- Your JDBC driver supports features that mask differences between databases, such as ODBC escape sequences and Generic SQL type codes.

The default mappings file is located in the Generic_ODBC directory.

If your database differs significantly from the SQL-92 standard, you must create a mappings file. For example, the configurable database type MySQL Enterprise 4.1 is installed with BIRT iHub. Figure 7-5 shows the location of the connection type configuration file for configurable database types and the mappings file for MySQL Enterprise.

When you configure a database type, first test your queries with the default mappings file. If any of your queries fail, you must create a mappings file for the database type. The mappings file in the Base_Mappings directory contains the default mapping as a reference.

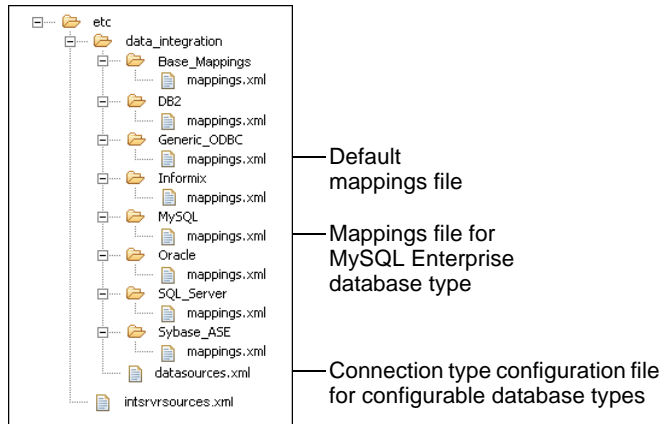


Figure 7-5 Location of the connection type configuration file and the mappings file for MySQL Enterprise

How to configure a database type

- 1 Configure a connection type for the database type using a `ConnectionType` element in `datasources.xml`.
- 2 Configure the database type using a `DatabaseType` element in `datasources.xml`

Do not specify the `DataSourceMapping` attribute, for example:

```
<DatabaseTypes>
  <DatabaseType Name="MyDatabaseType"
    ConnectionType="MyConnectionType" />
</DatabaseTypes>
```

The Integration service and BIRT Designer Professional use the `mappings.xml` file in the `Generic_ODBC` directory.

- 3 Restart the Integration service and BIRT Designer Professional to reload database type configurations.
- 4 Test your queries.

If any of your queries fail, go to step 5.

- 5 Create a directory and a mappings file for the database type in `$AC_SERVER_HOME/etc/data_integration` and `<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\data_integration`.

Start the mappings file by making a copy of the `mappings.xml` file in the `Generic_ODBC` directory.

- 6 Modify the `DatabaseType` element.

The `DataSourceMapping` attribute gives the name of the directory in which the mappings file resides, for example:

```
<DatabaseTypes>
  <DatabaseType Name="MyDatabaseType"
    ConnectionType="MyConnectionType"
    DataSourceMapping="MyDataSourceMapping" />
</DatabaseTypes>
```

- 7 Restart the Integration service and BIRT Designer Professional to reload database type configurations.
- 8 Test your queries and modify the mappings file as necessary.

Working with XML files

When you configure a database type, you modify the `datasources.xml` file and possibly create a `mappings.xml` file. When you work with XML, the following characters require special treatment:

- `<` (less than)
- `>` (greater than)
- `"` (double quotation mark)

Use the codes in Table 7-6 to represent these characters.

Table 7-6 XML codes for special characters

Special character	XML code
<code><</code>	<code>&lt;</code>
<code>></code>	<code>&gt;</code>
<code>"</code>	<code>&quot;</code>

In the following example, " represents the character ":

```
<Initializer>  
    SET SESSION sql_mode=&quot;ANSI_QUOTES&quot;;  
</Initializer>
```

Alternatively, you can use CDATA to escape an element value. XML parsers do not interpret the string data inside CDATA. The following example uses < and > but is acceptable because the element value is enclosed in ![CDATA[]]:

```
<FunctionMapping FunctionName="NE">  
    <![CDATA[    $P0 <> $P1    ]]>  
</FunctionMapping>
```


8

Configuring connection types

This chapter contains the following topics:

- About configuring connection types
- JDBC driver requirements and installation
- Working with datasources.xml

About configuring connection types

A connection type defines JDBC connection string syntax and connection parameters, for example user name and password. You provide values for connection parameters on the Data source connection properties page, as shown in Figure 8-1.

Data source connection properties

Type:

Description:

Configuration key:

Credentials:

User name:

Password:

Server:

Database:

Port:

Schema (optional):

Figure 8-1 Providing values for data source connection properties

To configure a connection type:

- Confirm that the database has a JDBC driver that meets the requirements described in this chapter.
- Install the JDBC driver.
- Modify the datasources.xml file:
 - Define the ConnectionType element.
 - Define the ConnectionType attribute for the appropriate DatabaseType elements.

JDBC driver requirements and installation

When you configure a connection type, you must specify the JDBC driver that the connection uses. The Integration service uses a JDBC driver to retrieve a list of database tables, views, or stored procedures when you create maps in the

IO Design perspective. For example, Figure 8-2 shows a list of tables in the Available pane.

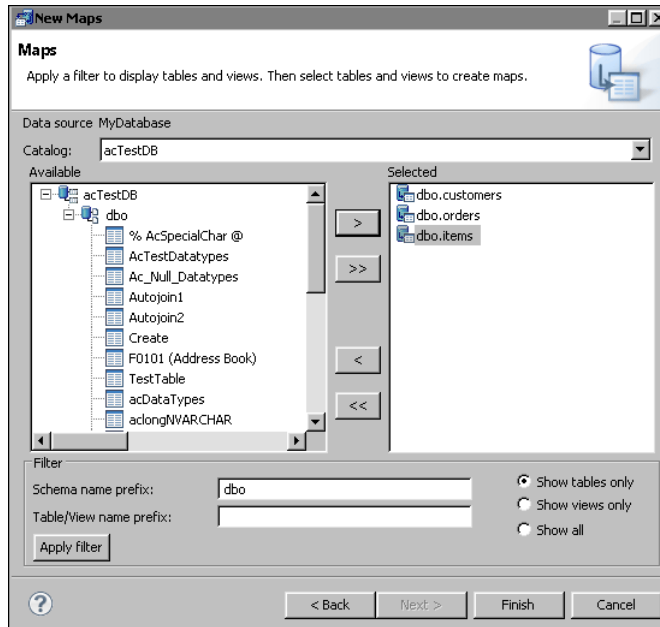


Figure 8-2 Selecting tables

The Integration service also uses a JDBC driver to execute Actuate SQL queries.

JDBC driver requirements

JDBC drivers must be JDBC 3.0 compatible. Specifically, the function `Driver.jdbcCompliant()` must return `TRUE`, and `DatabaseMetaData.getJDBCMajorVersion()` must return at least 3. Because the iHub compiler is based on Java 1.5, JDBC drivers should be compatible with JRE version 1.5 or earlier.

The JDBC driver must be able to retrieve a list of tables using the `DatabaseMetaData.getTables()` method. Some tables returned by this method may not be callable by the database user specified in the data connection definition. Actuate SQL queries that use these tables may fail at query execution time. For the IO Design perspective to display only the tables that can be selected, `DatabaseMetaData.allTablesAreSelectable()` must return `TRUE`.

The JDBC driver must be able to retrieve a list of stored procedures using the `DatabaseMetaData.getProcedures()` method. Some stored procedures returned by this method may not be callable by the database user specified in the data connection definition. Actuate SQL queries that use these stored procedures may fail at query execution time. For the IO Design perspective to display only the

callable stored procedures, `DataBaseMetaData.allProceduresAreCallable()` must return `TRUE`.

Installing a JDBC driver

You must install the JDBC driver on both the iHub and BIRT Designer Professional computers. You can store the driver anywhere in the file system as long as the user can access it. You specify the absolute path to the driver when you create the connection type. You can include the driver by name to ensure that no other driver is loaded.

Working with datasources.xml

To configure a connection type, you must modify `datasources.xml`. `datasources.xml` is located in the following directories:

```
$AC_SERVER_HOME/etc/data_integration
```

```
<BDPro_HOME>\eclipse\plugins\com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc\data_integration
```

`datasources.xml` contains two main elements:

- **ConnectionTypes**
ConnectionTypes contains connection type configurations.
- **DatabaseTypes**
DatabaseTypes associates a connection type and a mapping with a database type.

Configuring connection types: ConnectionTypes element

The `ConnectionTypes` element has one or more child elements called `ConnectionType`. Each `ConnectionType` element specifies how the Integration service connects to a database. You define a name for each connection type using the `Name` attribute.

The `ConnectionType` element has two child elements:

- **JDBCDriver**
- **ConnectionParams**

Here is an example of a ConnectionTypes element:

```
<ConnectionTypes>
  <ConnectionType Name="MySQL">
    <JDBCDriver DriverName="com.mysql.jdbc.MySQLDriver">
      <ConnectionString>
        jdbc:mysql:mysql://%server%:%port%;SID=%sid%
      </ConnectionString>
      <ConnectionProperties>
        <Property Name="Username">%username%</Property>
        <Property Name="Password">%password%</Property>
      </ConnectionProperties>
      <LibraryPath>
        <Location>/home/jsmith/bin/</Location>
        <Location>/home/jsmith/mysql/bin/</Location>
      </LibraryPath>
    </JDBCDriver>
    <ConnectionParams>
      <ConnectionParam Name="server"
        Display="Server"
        Type="String"
        DefaultValue="end2243"
        Optional="true"
        ValueIsCaseSensitive="false">
      </ConnectionParam>
      ...
    </ConnectionParams>
  </ConnectionType>
</ConnectionTypes>
```

ConnectionType child element: JDBCDriver

The JDBCDriver element contains information used to create JDBC connections. This element has one attribute, DriverName, as shown in Table 8-1. The JDBCDriver element has three child elements, ConnectionString, ConnectionProperties, and LibraryPath, as shown in Table 8-2.

Table 8-1 Attribute of the JDBCDriver element

Attribute	Description	Required
DriverName	Class name of the JDBC driver, for example com.mysql.jdbc.MySQLDriver.	Yes

Table 8-2 Child elements of the JDBCDriver element

Element	Description	Required
ConnectionString	JDBC connection string syntax. Do not include user name and password.	Yes
Connection Properties	User name and password properties.	Yes
LibraryPath	Paths to search for libraries used by the JDBC driver. Use a separate Location element for each path.	Yes

ConnectionString element

The JDBCDriver element has a ConnectionString child element, for example:

```
<ConnectionString>
  DRIVER={MySQL 4.0};DB=%database%;PORT=%port%;IP=%server%
</ConnectionString>
```

The ConnectionString element provides a template for the JDBC connection string. The parameters enclosed in percent signs (%), for example %server%, are placeholders for the values you type on the Data source connection properties page, shown in Figure 8-1, when you create a data connection definition. These values are retrieved from the data connection definition (DCD) file when the Integration service creates a connection.

You can exclude a portion of a connection string when no value is provided for a connection parameter by enclosing it in double brackets ([[...]]). In the following example, to exclude the IANAAppCodePage parameter from the connection string when the value is left blank, define the ConnectionString element as follows:

```
<ConnectionString>
  DRIVER={MySQL 4.3}; HOST=%server%; PORT=%port%;
  SID=%sid%;[[IANAAppCodePage : CODEPAGE=%IANAAppCodePage%]]
</ConnectionString>
```

[[IANAAppCodePage : CODEPAGE=%IANAAppCodePage%]] is not included in the connection string unless the IANAAppCodePage parameter is set.

To include the following literal characters in a ConnectionString element, precede the character with a backslash (\):

- \
- %

- [[
-]]
- ++

ConnectionType child element: CatalogFilter

The CatalogFilter element filters catalogs returned by the JDBC driver when the IO Design perspective displays a list of tables, views, or stored procedures in the New Maps dialog. By default, all catalogs are returned. Like the template for the JDBC connection string, the catalog filter can contain placeholders for the values of connection parameters, for example:

```
<CatalogFilter>%database%</CatalogFilter>
```

ConnectionType child element: ConnectionParams

The ConnectionParams element defines the parameters that are used in the ConnectionString element. The ConnectionParams element has a single child element, ConnectionParam. Here is an example of a ConnectionParam element that defines the server parameter:

```
<ConnectionParam Name="server"
    Display="Server"
    Type="String"
    DefaultValue="end2243"
    Optional="true"
    ValueIsCaseSensitive="false">
</ConnectionParam>
```

One ConnectionParam element is required for each parameter. Each ConnectionParam element has the attributes shown in Table 8-3.

Table 8-3 Attributes of the ConnectionParam element

Attribute	Description	Required?	Default value
Name	Name of the connection parameter. This attribute is case-insensitive.	Yes	
Display	Display name that appears on the Data source connection properties page in the IO Design perspective.	Yes	

(continues)

Table 8-3 Attributes of the ConnectionParam element (continued)

Attribute	Description	Required?	Default value
Type	Connection parameter type. Must be one of the following: <ul style="list-style-type: none">■ String■ Boolean■ Integer■ Masked (Use for a string whose value should be hidden, such as a password.)	Yes	
DefaultValue	Default value of the parameter.	No	
Optional	Specifies whether a parameter is optional.	No	True
ValueIsCaseSensitive	Specifies whether the parameter value is case-sensitive. Used when comparing two DCD files to see if they are equivalent.	No	True

Configuring database types: DatabaseTypes element

The DatabaseTypes element has one or more child elements called DatabaseType. Each DatabaseType element specifies the connection type and mapping for a database type. Several database types can use the same connection type, provided they use similar JDBC drivers, or the same mapping. For example, two different versions of a MySQL Enterprise database can use the same connection type.

Here is an example of a DatabaseTypes element:

```
<DatabaseTypes>
  <DatabaseType Name="MySQL4"
    ConnectionType="MySQL"
    DataSourceMapping="MySQL4" />
  <DatabaseType Name="Ingres"
    ConnectionType="Ingres"
    DataSourceMapping="Ingres" />
</DatabaseTypes>
```

Each DatabaseType element has the attributes shown in Table 8-4.

Table 8-4 Attributes of the DatabaseType element

Attribute	Description	Required?
Name	Name of the database type.	Yes
DisplayName	Display name for the database type in the New Data Connection Definition dialog in the IO Design perspective.	No
ConnectionType	Name of the connection type to use with this database type. The connection type is configured using the ConnectionType element.	Yes
DataSourceMapping	Directory where the mappings.xml file is located. This directory must be in the \$AC_SERVER_HOME/etc /data_integration and <BDPro_HOME>\eclipse \plugins \com.actuate.ais.embeddable _<version>\Config \aisconfigfiles\etc \data_integration directories.	No. If not specified, the Integration service uses the mappings.xml file in the Generic_ODBC directory. If you set this attribute to No_Mappings, all operations are executed by the Integration service executor.

Mapping data types

This chapter contains the following topics:

- About data type mapping
- `DataTypeMapper` element

About data type mapping

When you create a map of a database table, a database view, or a stored procedure result set, the IO Design perspective assigns an Actuate SQL data type to each column in the map. For example, Figure 9-1 shows the output columns for a map of a SQL Server database table and the Actuate SQL data types for each column.

Source column	Name	Data type
<input checked="" type="checkbox"/> orderNumber	orderNumber	Integer
<input checked="" type="checkbox"/> orderDate	orderDate	Timestamp
<input checked="" type="checkbox"/> requiredDate	requiredDate	Timestamp
<input checked="" type="checkbox"/> shippedDate	shippedDate	Timestamp
<input checked="" type="checkbox"/> status	status	Varchar
<input checked="" type="checkbox"/> comments	comments	Varchar
<input checked="" type="checkbox"/> customerNumber	customerNumber	Integer

Figure 9-1 Output columns and Actuate SQL data types for a map of a database table

The Actuate SQL data type for a column is determined as follows:

- The JDBC driver provides the generic SQL data type for the column, for example INTEGER, FLOAT, VARCHAR, or DATE.
- Based on the generic SQL data type, the Integration service chooses the best Actuate SQL data type for the column.

For example, in Figure 9-2 a column in a SQL Server database table has FLOAT data type. The JDBC driver provides the generic SQL data type FLOAT. The Integration service then chooses the Actuate SQL data type DOUBLE.

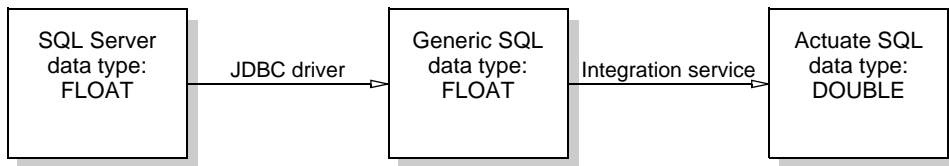


Figure 9-2 Mapping a database data type to an Actuate SQL data type

Table 9-1 lists the generic SQL types supported by the Integration service.

Table 9-1 Generic SQL types supported by the Integration service

Generic SQL type	Facets	Description	Actual SQL type
BIT		1-bit integer, with value 1 or 0.	INTEGER
TINYINT		8-bit integer.	INTEGER
SMALLINT		16-bit integer.	INTEGER
INTEGER		32-bit integer.	INTEGER
BIGINT		64-bit integer.	DECIMAL
FLOAT	Size	Floating point that can vary between single and double binary precision, depending on the value of size.	DOUBLE
REAL		Floating point (single binary precision).	DOUBLE
DOUBLE		Floating point (double binary precision).	DOUBLE
NUMERIC	Size, decimals	Decimal with scale decimals, and precision that cannot exceed size.	DECIMAL
DECIMAL	Size, decimals	Same as NUMERIC.	DECIMAL
CHAR	Size	Fixed-width character, maximum length is specified by size.	VARCHAR
VARCHAR	Size	Variable-width character, maximum length is specified by size.	VARCHAR
LONG VARCHAR	Size	Long variable-width character. Maximum length is specified by size. The Integration service does not support strings with a maximum length of more than 64,000 characters.	VARCHAR
DATE		Date with no time component.	TIMESTAMP
TIMESTAMP		Date with time component, with or without fractions-of-a-second field, containing up to 3 digits.	TIMESTAMP

DataTypeMapper element

If the JDBC driver does not provide a generic SQL data type for a column or the JDBC driver provides an incorrect data type, use the `DataTypeMapper` element in `mappings.xml` to map the column's database data type to a generic SQL data type. The `DataTypeMapper` element has one child element, `DataTypes`. The `DataTypes` element has one child element, `DataType`.

For each database data type you want to map, you define a `DataType` element. You should declare all `VARCHAR` and `DECIMAL` (or `NUMERIC`) data types using the `DataType` element. If these data types are not declared, the Integration service uses the ODBC escape sequences to convert these types. The escape sequences cannot specify the string length, decimal precision, or decimal scale of the result.

Each `DataType` element has the attributes shown in Table 9-2.

Table 9-2 Attributes of the `DataType` element

Attribute	Description	Required?
Name	Name of the database data type, for example <code>FLOAT</code> , <code>NUMBER</code> , <code>VARCHAR2</code> .	Yes
GenericSQLType	Generic SQL type the data type corresponds to.	Yes
MaxSize	Maximum size (decimal precision or string length) for the database data type. Applies only to <code>CHAR</code> , <code>VARCHAR</code> , <code>LONGVARCHAR</code> , <code>NUMERIC</code> , and <code>DECIMAL</code> types.	No, but highly recommended.

MaxSize attribute

For every database type, there is a maximum string length and a maximum decimal precision (the maximum number of decimal digits in a `DECIMAL` value). For example, for SQL Server 2000, `NVARCHAR` strings cannot exceed 4000 characters. When performing calculations involving strings and decimals, the Integration service uses these maximum sizes to determine how big the result can be. Strings and decimal digits may be truncated by the Integration service based on these maximum values. Also, if the maximum assumed by the Integration service is too large, the Integration service may create queries that the database cannot accept.

You should specify the `MaxSize` attribute for `CHAR`, `VARCHAR`, `LONGVARCHAR`, `NUMERIC` and `DECIMAL` types. Table 9-3 gives the default values for the `MaxSize` attribute.

Table 9-3 Default values for the `MaxSize` attribute

Generic SQL type	Default value of <code>MaxSize</code> attribute
<code>CHAR</code> , <code>VARCHAR</code>	255
<code>LONGVARCHAR</code>	64000
<code>NUMERIC</code> , <code>DECIMAL</code>	20

The following example uses the `MaxSize` attribute to specify the maximum string length for the database data type `NVARCHAR` and the maximum decimal precision for the database data type `DECIMAL`:

```
<DataTypeMapper>
  <DataTypes>
    <DataType Name="NVARCHAR" GenericSQLType="VARCHAR"
      MaxSize="1000" />
    <DataType Name="DECIMAL" GenericSQLType="DECIMAL"
      MaxSize="32" />
  </DataTypes>
</DataTypeMapper>
```

DataType child element: Aliases

The database may have other names or aliases for a data type. For example, on SQL Server `INT` is also called `INTEGER`. You must define the aliases so that the Integration service can recognize the data type when the database column uses an alias. To define aliases, use the `Aliases` element. The `Aliases` element has one child element, `Alias`. Define each alias using the `Alias` element.

The following example assumes you have a database that has the following data types: `CHAR`, `VARCHAR`, `NVARCHAR`, `SMALLINT`, `INT`, `DECIMAL`, `FLOAT`, and `DATETIME`. The Integration service maps all of these to generic SQL data types except `NVARCHAR` and `FLOAT`. For these data types, the Integration service returns an error such as `Type is not supported`. The Integration service returns an error because the JDBC driver does not provide the generic SQL type for `FLOAT` or `NVARCHAR`. You must define a mapping for data types that the JDBC driver cannot map.

Using your database documentation and the descriptions in Table 9-1, you select generic SQL types for `NVARCHAR` and `FLOAT`. You also define an alias for `FLOAT`:

```
<DataTypeMapper>
  <DataTypes>
    <DataType Name="NVARCHAR" GenericSQLType="VARCHAR" />
    <DataType Name="FLOAT" GenericSQLType="DOUBLE">
      <Aliases>
        <Alias>DOUBLE</Alias>
      </Aliases>
    </DataType>
  </DataTypes>
</DataTypeMapper>
```


Mapping functions and operators

This chapter contains the following topics:

- About mapping functions and operators
- Syntax for mapping functions and operators
- Using operators in a mapping
- Using initialization statements

About mapping functions and operators

If an Actuate SQL query contains a function, the Integration service must convert the function into a database function that returns the same results. To convert an Actuate SQL function to a database function, the Integration service uses function templates in a mappings.xml file. A function template contains a database function that is substituted for the Actuate SQL function when the query is sent to the database. The mappings.xml file in the Base_Mappings directory contains the default function templates.

If your database is ODBC-compliant and the database function is exactly like an Actuate SQL function, then the Integration service can perform the mapping for you using ODBC escape sequences. Relational databases, however, frequently differ in their implementation of SQL functions. Thus, when you configure a new database type you must resolve discrepancies in function implementations.

If your database does not support a function used in an Actuate SQL query, then you cannot map the function. The function must be performed by the Integration service.

About ODBC escape sequences

ODBC escape sequences are a set of standard patterns that are recognized by both ODBC and JDBC drivers. The sequences are used for creating SQL statements that are platform-independent. When sent to a JDBC or ODBC driver, the driver converts the escape sequences to SQL expressions that are recognized by the database. An ODBC escape sequence is enclosed in braces and starts with FN, which stands for function. An ODBC escape sequence has the following pattern:

```
{FN functionName(parameters)}
```

For example, the following query sent through a JDBC driver for SQL Server converts the values from the column Name to uppercase:

```
SELECT {FN UCASE (Name)} FROM Customers
```

The JDBC driver translates the expression to:

```
SELECT UPPER (Name) FROM Customers
```

The Integration service uses ODBC escape sequences to map functions. However, not all drivers implement all the ODBC escape sequences. In some cases, a database has no equivalent of an escape sequence. When a driver does not have an implementation for an escape sequence, the driver returns an error. When your driver does not support an escape sequence, you must provide a function mapping in mappings.xml. Additionally, when the escape sequence implementation is incompatible with the Actuate SQL specification, you must also edit mappings.xml.

The following example demonstrates one possible transformation of an Actuate SQL query into a database query, in this case SQL Server. The Integration service reads the mappings.xml file for function mappings but may determine that it can do some of the query operations more efficiently than the database. Thus, the query that the Integration service sends to the database may not use all mappings.

The following Actuate SQL query selects three columns from a single table, TOPDEALS:

```
SELECT CUSTID, CUSTOMNAME, FLOOR(AMOUNT) AS AMOUNT
FROM "../TOPDEALS.sma" AS TOPDEALS
WHERE Upper(CUSTOMNAME) LIKE Upper('DES%')
```

For the AMOUNT column, you use the FLOOR function to round down the values returned. In the WHERE clause, you define a filter condition so that the query only returns customers whose name starts with DES. With the exception of the FROM clause, which refers to a map file, the query looks like a generic SQL query.

To translate the query into a database query, the Integration service loads function templates from the mappings.xml file for the database type. The Integration service finds mappings for the Actuate SQL functions used in the query, as shown in the following plan:

```
Actuate SQL: FLOOR(), ODBC Escape Sequence: {FN FLOOR ($P0)},
SQL Server SQL: FLOOR()
Actuate SQL: UPPER(), ODBC Escape Sequence: {FN UCASE ($P0)},
SQL Server SQL: Upper()
```

The Integration service determines an optimal query execution plan after parsing the query and assessing the mappings. FLOOR() is not sent to the database. Instead, the Integration service performs this operation on the returned data. The following query is sent to the database:

```
SELECT CUSTID, CUSTOMNAME, AMOUNT
FROM TOPDEALS
WHERE UPPER( CUSTOMNAME ) LIKE Upper('DES%')
```

You can see the query sent to the database in the IO Design perspective Query Profiler.

Once the database returns results, the Integration service uses the data type mappings you defined in mappings.xml to convert the data in the result set to Actuate SQL data types. Finally, the Integration service performs any remaining operations, in this case the FLOOR function, before sending the data to the Factory service to generate the report.

Disabling the default mapping for a function

Your database may not support certain SQL-92 functions that are used by default to implement Actuate SQL functions. You must disable the default mappings for functions that are not supported. If you disable the mapping for a function, the Integration service performs the function instead of the database.

For example, some databases do not have an implementation of the POSITION function. The ODBC driver returns an error when the Integration service issues a query containing the POSITION function to such a database. To prevent the Integration service from encountering such errors, you should disable the mapping for the POSITION function in the mappings.xml file.

Differences between Actuate SQL functions and database functions

Your database's implementation of certain SQL functions may differ slightly from the Actuate SQL implementation of the same functions. For example, the Actuate SQL function DATEPART (weekday) is used to find the weekday of a date. The Actuate SQL function DATEPART(weekday) returns 0 for Sunday. On your database, a similar function WEEKDAY may return 1 for Sunday. The JDBC driver may hide this from the Integration service, but if it does not, a query would return incorrect values for the weekday of each date. Thus, you need to edit mappings.xml to define an appropriate substitute for the Actuate SQL function DATEPART.

About Generic_ODBC mappings.xml

Most of the elements in the Generic_ODBC mappings.xml are for mapping functions or operators. Functions and operators are implemented differently on different databases. Thus, most of your work in configuring a new database type involves mapping functions and operators. To map a function, you provide a template that includes a database function that the Integration service uses when rewriting a query.

The Generic_ODBC mappings.xml file contains the following elements for mapping functions and operators:

```
<DataSourceMappings>
  <DataSourceMapper>
    ...
    <BooleanOpMapper />
    <ComparisonOpMapper />
    <ArithOpMapper />
    <NumericFuncMapper />
    <BasicStringFuncMapper />
    <SubStringFuncMapper />
```



```

    <LikeOpMapper />
    <DatePartMapper />
    <DateDiffMapper />
    <DateAddMapper />
    <NullFuncMapper />
    <CondFuncMapper />
    <MultiRowBoolFuncMapper />
    <CastFuncMapper />
    ...
  </DataSourceMapper>
</DataSourceMappings>

```

The Generic_ODBC mappings.xml file does not contain examples or the default templates for mapping Actuate SQL functions and operators. The default templates are listed in the mappings.xml file in the \$AC_SERVER_HOME/etc /data_integration/Base_Mappings and <BDPro_HOME>\eclipse\plugins \com.actuate.ais.embeddable_<version>\Config\aisconfigfiles\etc \data_integration\Base_Mappings directories. In subsequent topics, you can find specific examples of how to map functions and operators.

Syntax for mapping functions and operators

Though it is possible to have the Integration service perform all Actuate SQL operations, it is not recommended. Whenever possible, you should execute functions and other operations on your database to optimize performance. Since Actuate SQL supports many common functions, for each Actuate SQL function there is usually an expression on the database that performs the same operation.

The goal of mapping a function is to specify a database function that provides the correct result for an Actuate SQL function. Sometimes it is possible to find a function on a database that behaves exactly like an Actuate SQL function. For example, Actuate SQL has a SUBSTRING function that takes three operands: \$P0 for the string to be evaluated, \$P1 for the start position, and \$P2 for the number of characters to retrieve. Your database may have a function called SUBSTR that has the same syntax and performs the same operation. Therefore, Actuate SQL's SUBSTRING (\$P0, \$P1, \$P2) maps to SUBSTR (\$P0, \$P1, \$P2) on the database.

In most cases, however, the mapping is not so straightforward. The syntax of the database function is usually different from the Actuate SQL syntax. Perhaps the SUBSTR function on the database does not specify the length of the substring as the third operand. Instead the SUBSTR function specifies the end position of the substring. For example, if the substring begins at position 3 and has 10 characters, the value of the third operand is 12. To map the SUBSTRING function, you must create an expression using the SUBSTR function that produces the same result, for example SUBSTR (\$P0, \$P1, \$P1 + \$P2 - 1).

The mappings.xml files for preconfigured database types contain many examples of function mapping. Do not modify the mappings.xml file for a preconfigured database type.

Mapping functions and operators: FunctionMapping element

The Generic_ODBC mappings.xml file contains empty elements for functions and operators. You customize function mappings by finding the appropriate element, such as BooleanOpMapper or NumericFuncMapper, and adding a FunctionMapping child element. The FunctionMapping element contains the attributes listed in Table 10-1.

Table 10-1 Attributes of the FunctionMapping element

Attribute	Description	Required?
FunctionName	Name of the Actuate SQL function.	Yes.
OperandTypes	Space-separated list of the Actuate SQL data types of the operands. The list can consist of the tokens BOOLEAN, INTEGER, DECIMAL, DOUBLE, VARCHAR, TIMESTAMP, and TABLE.	No. If the function can have many different operand types (overloading), then the mapping applies to all versions of the function. For DATEDIFF, DATEADD, and DATEPART, use the DatePart attribute instead.
DatePart	Used only for DATEDIFF, DATEADD, and DATEPART. Specifies the date part being mapped. Must be one of: <ul style="list-style-type: none"> ■ yyyy (year) ■ q (quarter) ■ m (month) ■ d (day) ■ h (hour) ■ n (minute) ■ s (second) ■ w (weekday) ■ y (day of year) 	Yes, for DATEDIFF, DATEADD, and DATEPART. Not required for other functions.
Disabled	Whether the mapping is disabled. If set to True, then the expression is not sent to the database. It is handled by the Integration service.	No. Default is False.

About function templates

You use the syntax shown in Table 10-2 to define function templates.

Table 10-2 Syntax used to define function templates

Syntax	Represents
\$Pn	The (n+1)th operand. \$P0 is the first operand, \$P1 is the second operand, \$P2 is the third operand, and so on. An operand can be a literal like 'Hello' or 6. An operand can also be a column on the database or a parameter.
\$R	The database data type that the expression returns, for example CAST \$P0 AS \$R.
\$\$	The dollar sign (\$).

The Integration service calculates the return data type for a function. You cannot calculate the return data type manually in a reliable manner. For example, when mapping the CAST function, do not use CAST (\$P0 AS NVARCHAR). Instead, use CAST (\$P0 AS \$R). The Integration service determines the correct data type. When you use the \$R syntax, you must explicitly declare the name of the database data type in the DataTypeMapper element, otherwise the ODBC name, for example SQL_VARCHAR, is used.

To help you understand how to use function templates, three examples of customized function mappings are given below.

Example: Mapping the POWER function

When you test your queries, you discover that the default mapping for the Actuate SQL POWER function returns an error. The Integration service uses the ODBC escape sequence {FN POWER (\$P0, \$P1)} as the default mapping. However, your JDBC driver does not have an implementation for the escape sequence and thus cannot rewrite the expression. Therefore, you must map the POWER function.

After checking your database documentation, you determine that the database has a function called POWER and that it takes two arguments. You compare this to the Actuate SQL POWER prototypes:

```
Integer POWER( base Integer, exponent Integer )
Decimal POWER( base Decimal, exponent Integer )
Double POWER( base Double, exponent Integer )
```

You use the NumericFuncMapper element. Because there is no OperandTypes attribute, the mapping applies to all versions of the POWER function:

```

<NumericFuncMapper>
  <FunctionMappings>
    <FunctionMapping
      FunctionName="POWER"> <!-- The Actuate SQL function -->
      POWER ($P0, $P1) <!-- The database function -->
    </FunctionMapping>
  </FunctionMappings>
</NumericFuncMapper>

```

For another example, refer to the mappings.xml file for Oracle, which requires a very different mapping for the POWER function. To map POWER, you use the OperandTypes attribute and define three different mappings:

```

<NumericFuncMapper>
  <FunctionMappings>
    ...
    <FunctionMapping FunctionName="POWER"
      OperandTypes="INTEGER INTEGER">
      CAST( TRUNC(POWER( $P0, $P1 )) AS $R )
    </FunctionMapping>
    <FunctionMapping FunctionName="POWER"
      OperandTypes="DECIMAL INTEGER">
      CAST(POWER( $P0, $P1 ) AS $R)
    </FunctionMapping>
    <FunctionMapping FunctionName="POWER"
      OperandTypes="DOUBLE INTEGER">
      POWER( $P0, $P1 )
    </FunctionMapping>
    ...
  </FunctionMappings>
</NumericFuncMapper>

```

Example: Mapping the DATEDIFF function with date part yyyy

The Actuate SQL DATEDIFF function uses a prototype that enables you to provide a date part such as yyyy. Thus, you can use the same function to do date subtraction for years, months, days, etc.:

```

Integer datediff( datepart Varchar, start Timestamp, end
  Timestamp )

```

When you test your queries you discover there is a problem with the default mapping for the DATEDIFF function with date part yyyy. The driver is mapping the DATEDIFF function to the YEARS_BETWEEN function on the database. The query gives incorrect results because the year is consistently off by one.

You can resolve the problem using the following mapping:

```
<DateDiffMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="DATEDIFF" DatePart="yyyy">
      YEARS_BETWEEN ($P0, $P1) - 1
    </FunctionMapping>
  </FunctionMappings>
</DateDiffMapper>
```

As another example, DB2 uses the following mapping for the DATEDIFF function with date part yyyy:

```
<FunctionMapping FunctionName="DATEDIFF" DatePart="yyyy">
  (YEAR( $P1 ) - YEAR ( $P0 ))
</FunctionMapping>
```

Example: Disabling the POSITION function

You determine that your database does not support the POSITION function. Therefore, you must disable the mapping of the POSITION function:

```
<SubStringFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="POSITION"
      Disabled="true" />
  </FunctionMappings>
</SubStringFuncMapper>
```

Mapping Boolean operators: BooleanOpMapper element

The BooleanOpMapper element is used for customizing the mappings of the Boolean operators AND, OR, and NOT. Table 10-3 shows the default templates for mapping Boolean operators.

Table 10-3 Default templates for mapping Boolean operators

Boolean operator	Operand data types	Default template
AND	<BOOLEAN>, <BOOLEAN>	\$P0 AND \$P1
OR	<BOOLEAN>, <BOOLEAN>	\$P0 OR \$P1
NOT	<BOOLEAN>	NOT \$P0

Example: Mapping the NOT operator

You determine that the database sometimes returns errors when the argument of the NOT operator is not enclosed in parentheses. To resolve this problem, you use the following mapping:

```
<BooleanOpMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="NOT">
      NOT ($P0)
    </FunctionMapping>
  </FunctionMappings>
</BooleanOpMapper>
```

Mapping comparison operators: ComparisonOpMapper element

The ComparisonOpMapper element is used for customizing the mappings of the comparison operators listed in Table 10-4. The table also shows the default template for each operator.

Table 10-4 Default templates for mapping comparison operators

Comparison operator	Operand data types	Default template	Remarks
=	<INTEGER>, <INTEGER>	\$P0 = \$P1	Use EQ as the FunctionName for =.
>	<DECIMAL>, <DECIMAL>	\$P0 > \$P1	Use GT as the FunctionName for >.
<	<DOUBLE>, <DOUBLE>	\$P0 < \$P1	Use LT as the FunctionName for <.
>=	<VARCHAR>, <VARCHAR>	\$P0 >= \$P1	Use GE as the FunctionName for >=.
<=	<TIMESTAMP>, <TIMESTAMP>	\$P0 <= \$P1	Use LE as the FunctionName for <=.
<>		\$P0 <> \$P1	Use NE as the FunctionName for <>.

Example: Mapping the <> operator

You change the mapping of the <> operator for VARCHAR because on your database the comparison operator for strings is an exclamation point followed by an equals sign (!=). You use NE as the FunctionName:

```
<ComparisonOpMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="NE"
      OperandTypes="VARCHAR VARCHAR">
      $P0 != $P1
    </FunctionMapping>
  </FunctionMappings>
</ComparisonOpMapper>
```

Mapping arithmetic operators: ArithOpMapper element

The ArithOpMapper element is used for customizing the mappings of the arithmetic operators listed in Table 10-5. The table also shows the default template for each operator.

Table 10-5 Default templates for mapping arithmetic operators

Arithmetic operator	Operand data types	Default template	Remarks
+	<INTEGER>, <INTEGER>	(\$P0 + \$P1)	Use ADD as the FunctionName for +.
-	<DOUBLE>, <DOUBLE>	(\$P0 - \$P1)	Use SUB as the FunctionName for -.
*	<DECIMAL>, <DECIMAL>	(\$P0 * \$P1)	Use MULT as the FunctionName for *.
/	<INTEGER>, <INTEGER>	(\$P0 / \$P1)	Use DIV as the FunctionName for /.
	<DOUBLE>, <DOUBLE>	(\$P0 / \$P1)	
	<DECIMAL>, <DECIMAL>	Generated by the Integration service	
-	<INTEGER> <DOUBLE> <DECIMAL>	-(\$P0)	Use NEG as the FunctionName for -.

Example: Mapping the negation operator

You change the mapping of the negation operator because your database uses a different syntax. You use NEG as the FunctionName:

```

<ArithOpMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="NEG">
      NEGATE ($P0)
    </FunctionMapping>
  </FunctionMappings>
</ArithOpMapper>

```

Mapping numeric functions: NumericFuncMapper element

The NumericFuncMapper element is used for customizing the mappings of the numeric functions listed in Table 10-6. The table also shows the default template for each function.

Table 10-6 Default templates for mapping numeric functions

Numeric function	Operand data types	Default template
ROUND	<DECIMAL>, <INTEGER> <DOUBLE>, <INTEGER>	{FN ROUND (\$P0, \$P1)}
FLOOR	<DECIMAL> <DOUBLE>	{FN FLOOR (\$P0)}
CEILING	<DECIMAL> <DOUBLE>	{FN CEILING (\$P0)}
POWER	<INTEGER>, <INTEGER> <DECIMAL>, <INTEGER> <DOUBLE>, <INTEGER>	{FN POWER (\$P0, \$P1)} Generated by the Integration service {FN POWER (\$P0, \$P1)}
MOD	<INTEGER>, <INTEGER>	{FN MOD (\$P0, \$P1)}

Example: Mapping the POWER function

The POWER function with DECIMAL and INTEGER operands does not give accurate results on your database. To obtain more accurate results, you convert the second operand to a decimal:

```

<NumericFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="POWER"
      OperandTypes="DECIMAL INTEGER">
      POWER ($P0, CAST ($P1 AS DECIMAL (10, 0))
    </FunctionMapping>
  </FunctionMappings>
</NumericFuncMapper>

```


Mapping string functions: BasicStringFuncMapper element

The BasicStringFuncMapper element is used for customizing the mappings of the functions listed in Table 10-7. The table also shows the default template for each function.

Table 10-7 Default templates for mapping string functions

String function	Operand data types	Default template
CHAR_LENGTH	<VARCHAR>, <INTEGER>	{FN LENGTH (\$P0)}
UPPER	<VARCHAR>	{FN UCASE (\$P0)}
LOWER	<VARCHAR>	{FN LCASE (\$P0)}
LTRIM	<VARCHAR>	{FN LTRIM (\$P0)}
RTRIM	<VARCHAR>	{FN RTRIM (\$P0)}
CONCAT	<VARCHAR>, <VARCHAR>	{FN CONCAT (\$P0, \$P1)}

Example: Mapping the CHAR_LENGTH function

The Actuate SQL CHAR_LENGTH function returns the length of a string including trailing spaces. The corresponding database function, however, ignores trailing spaces. To resolve this discrepancy, the following mapping appends an underscore (_) to the string, applies the database function LEN, and subtracts one:

```
<BasicStringFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="CHAR_LENGTH">
      LEN ($P0 + ' _') - 1
    </FunctionMapping>
  </FunctionMappings>
</BasicStringFuncMapper>
```

Mapping substring functions: SubStringFuncMapper element

The SubStringFuncMapper element is used for customizing the mappings of the functions listed in Table 10-8. The table also shows the default template for each function.

Table 10-8 Default templates for mapping substring functions

Substring function	Operand data types	Default template
SUBSTRING	<VARCHAR>, <INTEGER>, <INTEGER>	{FN SUBSTRING (\$P0, \$P1, \$P2)}
LEFT	<VARCHAR>, <INTEGER>	{FN LEFT (\$P0, \$P1)}
RIGHT	<VARCHAR>, <INTEGER>	{FN RIGHT (\$P0, \$P1)}
POSITION	<VARCHAR>, <VARCHAR>	{FN LOCATE (\$P0, \$P1)}

Example: Mapping the POSITION function

You must map the POSITION function because your driver does not implement the escape sequence {FN LOCATE (\$P0, \$P1)}:

```
<SubStringFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="POSITION">
      POSITION ($P0, $P1)
    </FunctionMapping>
  </FunctionMappings>
</SubStringFuncMapper>
```

Mapping the LIKE operator: LikeOpMapper element

The LikeOpMapper element is used for customizing the mapping of the LIKE operator. The default template is \$P0 LIKE \$P1 ESCAPE '@'. \$P1 is the pattern against which to compare the string \$P0. \$P0 and \$P1 are both of type VARCHAR.

To map the LIKE operator, do the following:

- Determine whether the database has an equivalent for the LIKE operator. The database equivalent must be able to support escaping special characters in the pattern, for example through the ESCAPE clause. If no database equivalent exists, disable the mapping by setting the Disabled attribute to true.
- Identify the character that the database uses for matching a single character. For example, standard SQL uses the underscore (_). Set the SingleMatchChar attribute to this character. If no such character exists, disable the mapping by setting the Disabled attribute to true.
- Identify the character that the database uses for matching any number of characters. For example, standard SQL uses the percent sign (%). Set the GreedyMatchChar attribute to this character. If no such character exists, disable the mapping by setting the Disabled attribute to true.
- Determine how special characters are escaped on the database. Typically, you specify an escape character, for example backslash (\), using an ESCAPE

clause in the LIKE template, for example \$P0 LIKE \$P1 ESCAPE '\'. You then specify the EscapeTemplate attribute to show how to escape special characters. For example, \\$ indicates that the backslash precedes the special character.

- Identify any additional special characters that the database recognizes within the pattern. For example, some databases allow pattern matching using the square bracket syntax [a-z0-9]. In this case, the square brackets must be escaped whenever the Integration service pushes queries to the database to ensure that the database interprets these characters as literals.

The attributes used for customizing the LIKE operator mapping are listed in Table 10-9.

Table 10-9 Attributes for customizing the LIKE operator mapping

Attribute name	Description	Required?
Disabled	Set to true to disable the mapping for the LIKE operator.	No. Default is false.
SingleMatchChar	Character used on the database to match a single character.	No. Default is underscore (_). For example, on the database '_rown' matches 'Brown' and 'Crown'.
GreedyMatchChar	Character used on the database to match any number of characters.	No. Default is percent sign (%). For example, on the database 'Hat%' matches 'Hatcher' and 'Hathaway'.
EscapeTemplate	Template that shows how to escape a special character on the database. In the template, \$ stands for the special character, while \$\$ stands for the dollar sign.	No. Default is @\$. By default, the characters used on the database for single match and greedy match are escaped by prepending an @.
AdditionalSpecialChars	Any special characters other than the single match character and the greedy match character. Additional special characters are listed without spaces.	No. Default is at sign (@), the default escape character.

Example: Mapping the LIKE operator

Your database has an equivalent for the LIKE operator called MATCH. The MATCH operator uses the question mark (?) to match a single character and the asterisk (*) to match any number of characters. The MATCH operator uses square brackets to escape special characters, for example [?]:

```

<LikeOpMapper SingleMatchChar="?"
  GreedyMatchChar="*"
  EscapeTemplate="[$]"
  AdditionalSpecialChars="[]">
  <FunctionMappings>
    <FunctionMapping FunctionName="LIKE">
      MATCH ($P0, $P1)
    </FunctionMapping>
  </FunctionMappings>
</LikeOpMapper>

```

Example: Changing the escape character

Your database supports the LIKE operator but errors occur when you use the at sign (@) as the escape character, so you use the backslash (\) instead:

```

<LikeOpMapper
  EscapeTemplate="\$"
  AdditionalSpecialChars="\ ">
  <FunctionMappings>
    <FunctionMapping FunctionName="LIKE">
      $P0 LIKE $P1 ESCAPE '\'
    </FunctionMapping>
  </FunctionMappings>
</LikeOpMapper>

```

Example: Disabling the LIKE operator

Your database has no equivalent for the LIKE operator so you disable the mapping. Disabling the mapping means that the Integration service processes LIKE expressions, not the database:

```

<LikeOpMapper Disabled="true" />

```

Example: Specifying additional special characters

Your database supports the LIKE operator, but extends it to recognize patterns such as [a-z0-9]. If the characters open square bracket ([), close square bracket (]), and hyphen (-) appear in a string, they must be escaped so that the database interprets them as literals instead of assigning special meaning to them:

```

<LikeOpMapper AdditionalSpecialChars="@[]-" />

```

Mapping DATEPART functions: DatePartMapper element

DATEPART takes two arguments: a date part and a timestamp. It returns the part of the timestamp specified by the date part:

```

Integer datepart( datepart Varchar, value Timestamp )

```

The DatePartMapper element is used to customize the mappings for the date parts listed in Table 10-10. The table also shows the default template for each date part.

Table 10-10 Default templates for mapping date parts with the DATEPART function

Date part	Default template
yyyy (year)	{FN YEAR (\$P0)}
q (quarter)	{FN QUARTER (\$P0)}
m (month)	{FN MONTH (\$P0)}
d (day)	{FN DAYOFMONTH (\$P0)}
h (hour)	{FN HOUR (\$P0)}
n (minute)	{FN MINUTE (\$P0)}
s (second)	{FN SECOND (\$P0)}
w (day of week)	{FN DAYOFWEEK (\$P0)}
y (day of year)	{FN DAYOFEAR (\$P0)}

Example: Mapping the DATEPART functions

Your database has a different syntax for the DATEPART functions. You define each part using a mapping:

```
<DatePartMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="DATEPART"
      DatePart="yyyy">
      TO_NUMBER (TO_CHAR ($P0, 'YYYY'))
    </FunctionMapping>
    ...
    <FunctionMapping FunctionName="DATEPART"
      DatePart="y">
      TO_NUMBER (TO_CHAR ($P0, 'DDD'))
    </FunctionMapping>
  </FunctionMappings>
</DatePartMapper>
```

Mapping date subtraction functions: DateDiffMapper element

DATEDIFF takes three arguments: a date part, a start timestamp, and an end timestamp. It returns the integer delta between the part of the two timestamps specified by the date part:

```
Integer datediff( datepart Varchar, start Timestamp, end
Timestamp )
```

The DateDiffMapper element is used to customize the mappings for the date parts listed in Table 10-11. The table also shows the default template for each date part.

Table 10-11 Default templates for mapping date parts with the DATEDIFF function

Date part	Default template
yyyy (year)	{FN TIMESTAMPDIFF (SQL_TSI_YEAR, \$P0, \$P1)}
q (quarter)	{FN TIMESTAMPDIFF (SQL_TSI_QUARTER, \$P0, \$P1)}
m (month)	{FN TIMESTAMPDIFF (SQL_TSI_MONTH, \$P0, \$P1)}
d (day)	{FN TIMESTAMPDIFF (SQL_TSI_DAY, \$P0, \$P1)}
h (hour)	{FN TIMESTAMPDIFF (SQL_TSI_HOUR, \$P0, \$P1)}
n (minute)	{FN TIMESTAMPDIFF (SQL_TSI_MINUTE, \$P0, \$P1)}
s (second)	{FN TIMESTAMPDIFF (SQL_TSI_SECOND, \$P0, \$P1)}
w (day of week)	{FN TIMESTAMPDIFF (SQL_TSI_DAY, \$P0, \$P1)} / 7
y (day of year)	{FN TIMESTAMPDIFF (SQL_TSI_DAY, \$P0, \$P1)}

Examples: Mapping the DATEDIFF function with date part yyyy

The following examples show different ways of mapping the DATEDIFF function with date part yyyy.

Example 1

```
<FunctionMapping FunctionName="DATEDIFF"
  DatePart="yyyy">
  (YEAR( $P1 ) - YEAR ( $P0 ))
</FunctionMapping>
```

Example 2

```
<FunctionMapping FunctionName="DATEDIFF"
  DatePart="yyyy">

  CAST(
    TO_NUMBER( TO_CHAR( $P1, 'YYYY' ) )
    -
    TO_NUMBER( TO_CHAR( $P0, 'YYYY' ) )
    AS NUMBER(9)
  )
</FunctionMapping>
```

Example 3

```
<FunctionMapping FunctionName="DATEDIFF"
  DatePart="yyyy">
  DATEDIFF( year, $P0, $P1 )
</FunctionMapping>
```

Mapping date addition functions: DateAddMapper element

DATEADD takes three arguments: a date part, an integer delta value, and a timestamp value. It returns a timestamp that applies the delta value to the specified part of the original timestamp:

```
Timestamp dateadd( datepart Varchar, delta Integer, value
  Timestamp )
```

The DateAddMapper element is used to customize the mappings for the date parts listed in Table 10-12. The table also shows the default template for each date part.

Table 10-12 Default templates for mapping date parts with the DATEADD function

Date part	Default template
yyyy (year)	{FN TIMESTAMPADD (SQL_TSI_MONTH, \$P0*12, \$P1)}
q (quarter)	{FN TIMESTAMPADD (SQL_TSI_MONTH, \$P0*3, \$P1)}
m (month)	{FN TIMESTAMPADD (SQL_TSI_MONTH, \$P0, \$P1)}
d (day)	{FN TIMESTAMPADD (SQL_TSI_DAY, \$P0, \$P1)}
h (hour)	{FN TIMESTAMPADD (SQL_TSI_HOUR, \$P0, \$P1)}
n (minute)	{FN TIMESTAMPADD (SQL_TSI_MINUTE, \$P0, \$P1)}
s (second)	{FN TIMESTAMPADD (SQL_TSI_SECOND, \$P0, \$P1)}
w (day of week)	{FN TIMESTAMPADD (SQL_TSI_DAY, \$P0, \$P1)}
y (day of year)	{FN TIMESTAMPADD (SQL_TSI_DAY, \$P0, \$P1)}

Example: Mapping the DATEADD functions

Your database has a different syntax for the DATEADD functions. You define each part using a mapping:

```
<DateAddMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="DATEADD"
      DatePart="yyyy">
      ($P1 + $P0 YEARS)
    </FunctionMapping>
```

```

...
<FunctionMapping FunctionName="DATEADD"
    DatePart="y">
    ($P1 + $P0 DAYS)
</FunctionMapping>
</FunctionMappings>
</DateAddMapper>

```

Mapping date serialization functions: DateSerialMapper element

DATESERIAL has two forms. The first form takes three arguments: a year value, a month value, and a day value. It returns a timestamp for the date corresponding to the specified year, month, and day with the time set to 00:00:00.0:

```

Timestamp dateserial( year Integer, month Integer,
    day Integer )

```

The second form of dateserial takes six arguments: values for the year, month, day, hour, minute, and second. It returns the timestamp for the specified values:

```

Timestamp dateserial( year Integer, month Integer, day Integer,
    hour Integer, minute Integer, second Integer )

```

The DateSerialMapper element is used for customizing the mappings of the DATESERIAL functions. The default templates are generated by the Integration service. In most cases, it is not necessary to override them.

Example: Disabling the DATESERIAL functions

The DATESERIAL templates generated by the Integration service do not work, so you disable the mappings for both versions of DATESERIAL:

```

<DateSerialMapper>
    <FunctionMappings>
        <FunctionMapping FunctionName="DATESERIAL"
            Disabled="true" />
    </FunctionMappings>
</DateSerialMapper>

```

Mapping NULL functions: NullFuncMapper element

The NullFuncMapper element is used for customizing the mappings of the NULL functions listed in Table 10-13. The table also shows the default template for each function.

Table 10-13 Default templates for mapping NULL functions

NULL function	Operand data types	FunctionName	Default template
IS NULL	<INTEGER> <DECIMAL> <DOUBLE> <VARCHAR> <TIMESTAMP>	Use IS_NULL as FunctionName for IS NULL.	\$P0 IS NULL
CAST (NULL AS INTEGER)		Use CAST_NULL_AS_INTEGER as FunctionName.	Generated by the Integration service.
CAST (NULL AS DECIMAL)		Use CAST_NULL_AS_DECIMAL as FunctionName.	
CAST (NULL AS DOUBLE)		Use CAST_NULL_AS_DOUBLE as FunctionName.	
CAST (NULL AS VARCHAR)		Use CAST_NULL_AS_VARCHAR as FunctionName.	
CAST (NULL AS TIMESTAMP)		Use CAST_NULL_AS_TIMESTAMP as FunctionName.	

Example: Disabling the CAST (NULL AS . . .) functions

Your database does not support the NULL literal, so the CAST (NULL AS . . .) functions must be disabled:

```
<NullFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="CAST_NULL_AS_INTEGER"
      Disabled="true" />
    <FunctionMapping FunctionName="CAST_NULL_AS_DECIMAL"
      Disabled="true" />
    <FunctionMapping FunctionName="CAST_NULL_AS_DOUBLE"
      Disabled="true" />
    <FunctionMapping FunctionName="CAST_NULL_AS_VARCHAR"
      Disabled="true" />
    <FunctionMapping FunctionName="CAST_NULL_AS_TIMESTAMP"
      Disabled="true" />
  </FunctionMappings>
</NullFuncMapper>
```

Mapping conditional functions: CondFuncMapper element

The CondFuncMapper element has attributes that you use to customize CASE statements. These attributes are listed in Table 10-14.

Table 10-14 Attributes of the CondFuncMapper element

Attribute name	Description	Required?
CaseWhenString	String to use instead of CASE WHEN	No. Default is CASE WHEN.
WhenString	String to use instead of WHEN	No. Default is WHEN.
ThenString	String to use instead of THEN	No. Default is THEN.
ElseString	String to use instead of ELSE	No. Default is ELSE.
EndString	String to use instead of END	No. Default is END.

Example: Mapping the CASE statement

Your database uses a SWITCH statement instead of a CASE statement. SWITCH is not standard SQL. The Actuate SQL CASE prototype is as follows:

```
CASE [<ValueExpression>]
  {<WhenClause>} [...n]
  [ELSE <ValueExpression>]
END
```

The mapping for SWITCH is as follows:

```
<CondFuncMapper CaseWhenString="SWITCH ("
  WhenString=", "
  ThenString=", "
  ElseString=", TRUE,"
  EndString=")" />
```

This mapping produces a SWITCH statement such as:

```
SWITCH (
  Country IN ('Canada', 'Mexico', 'USA'), 'North America',
  Country IN ('Argentina', 'Brazil', 'Venezuela'), 'South
    America',
  Country IS NULL, '(Not Known)',
  TRUE, 'Rest of the world')
```

Mapping aggregate functions: AggrFuncMapper element

The AggrFuncMapper element is used for customizing the mappings of the aggregate functions listed in Table 10-15. The table also shows the default template for each function.

Table 10-15 Default templates for mapping aggregate functions

Aggregate function	Operand data types	Default template	Remarks
SUM	<INTEGER>	SUM (\$P0)	
AVG	<DECIMAL>	AVG (\$P0)	
	<DOUBLE>	AVG (DISTINCT \$P0)	
MAX	<INTEGER>	MAX (\$P0)	
MIN	<DECIMAL>	MIN (\$P0)	
COUNT	<DOUBLE>	COUNT (\$P0)	
	<VARCHAR>		
	<TIMESTAMP>		
COUNT (*)		COUNT (*)	Use COUNT_ROWS as the FunctionName for COUNT (*).

Example: Mapping the AVG function

The default template for the AVG function does not return the correct result for DECIMAL data types, so you use the following mapping:

```
<AggrFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="AVG"
      OperandTypes="DECIMAL">
      CAST(AVG( $P0 ) AS $R)
    </FunctionMapping>
  </FunctionMappings>
</AggrFuncMapper>
```

Mapping multi-row Boolean operators: MultiRowBoolFuncMapper element

The MultiRowBoolFuncMapper element is used for customizing the mappings of the multi-row Boolean operators listed in Table 10-16. The table also shows the default template for each operator.

The = ANY and <> ANY operators are implemented using the IN and NOT IN operators. The = ALL operators are implemented using the = ANY operators.

Table 10-16 Default templates for mapping multi-row Boolean operators

Multi-row Boolean operator	Operand data types	Default template	Remarks
EXISTS	<TABLE>	EXISTS \$P0	
IN	<INTEGER>, <TABLE>	\$P0 IN \$P1	
NOT IN	<DECIMAL>, <TABLE> <DOUBLE>, <TABLE>	\$P0 NOT IN \$P1	Use NOT_IN as FunctionName for NOT IN.
< ANY	<VARCHAR>, <TABLE> <TIMESTAMP>,	\$P0 < ANY \$P1	Use LT_ANY as FunctionName for < ANY.
> ANY	<TABLE>	\$P0 > ANY \$P1	Use GT_ANY as FunctionName for > ANY.
<= ANY		\$P0 <= ANY \$P1	Use LE_ANY as FunctionName for <= ANY.
>= ANY		\$P0 >= ANY \$P1	Use GE_ANY as FunctionName for >= ANY.

Mapping cast functions: CastFuncMapper element

The CastFuncMapper element is used for customizing the mappings of the cast functions listed in Table 10-17. The Integration service generates the default templates for the cast functions.

Table 10-17 Operand data types for the cast functions

Cast function	Operand data types	Remarks
CAST (AS INTEGER)	<DECIMAL> <DOUBLE> <VARCHAR>	Use CAST_AS_INTEGER as FunctionName.
CAST (AS DECIMAL)	<INTEGER>, <INTEGER>, <INTEGER> <DECIMAL>, <INTEGER>, <INTEGER> <DOUBLE>, <INTEGER>, <INTEGER> <VARCHAR>, <INTEGER>, <INTEGER>	Use CAST_AS_DECIMAL as FunctionName. The second and third operands are the decimal precision and scale, for example 20 and 8 in CAST (AS DECIMAL (20, 8)). Specify the default precision and scale using the iHub configuration variables DefaultDecimalPrecision and DefaultDecimalScale.

Table 10-17 Operand data types for the cast functions

Cast function	Operand data types	Remarks
CAST (AS DOUBLE)	<INTEGER> <DECIMAL> <VARCHAR>	Use CAST_AS_DOUBLE as FunctionName.
CAST (AS VARCHAR)	<INTEGER>, <INTEGER> <DECIMAL>, <INTEGER> <DOUBLE>, <INTEGER> <VARCHAR>, <INTEGER> <TIMESTAMP>, <INTEGER>	Use CAST_AS_VARCHAR as FunctionName. The second operand is the string length, for example 50 in CAST(AS VARCHAR (50)). Specify the default string length using the iHub configuration variable DefaultStringLength.
CAST (AS TIMESTAMP)	<VARCHAR>	Use CAST_AS_TIMESTAMP as FunctionName.

Example: Mapping the CAST functions

The default templates generated by the Integration service are not compatible with your database, so you change the mappings to use the CONVERT function. \$R represents the return data type:

```

<CastFuncMapper>
  <FunctionMappings>
    <FunctionMapping FunctionName="CAST_AS_INTEGER">
      CONVERT ($R, $P0)
    </FunctionMapping>
    <FunctionMapping FunctionName="CAST_AS_DECIMAL">
      CONVERT ($R, $P0)
    </FunctionMapping>
    <FunctionMapping FunctionName="CAST_AS_DOUBLE">
      CONVERT ($R, $P0)
    </FunctionMapping>
    <FunctionMapping FunctionName="CAST_AS_VARCHAR">
      CONVERT ($R, $P0)
    </FunctionMapping>
    <FunctionMapping FunctionName="CAST_AS_TIMESTAMP">
      CONVERT ($R, $P0)
    </FunctionMapping>
  </FunctionMappings>
</CastFuncMapper>

```

Using operators in a mapping

When including operators in a mapping, use the following guidelines.

Symbolic operators require parentheses

To avoid problems with operator precedence on a database, all symbolic operators must be enclosed in parentheses. For example, use parentheses around an arithmetic expression, such as `($P0 + $P1)`. Do not use `$P0 + $P1`. The only exception to this rule is operators that return Boolean values such as `=`, `<`, and `>`.

The operators `NOT`, `AND`, and `OR` should also be enclosed in parentheses. For example, use `($P0 IS NULL AND $P1 IS NOT NULL)`, not `$P0 IS NULL AND $P1 IS NOT NULL`.

You also need parentheses in function mappings that use such operators, regardless of whether you are mapping an Actuate SQL function or operator.

You do not need parentheses if the template already contains the equivalent of parentheses, for example the parentheses of a function or the commas that separate operands.

Negative sign must be followed by a space

When a mapping contains a negative sign `(-)` followed by an operand, place a space after the negative sign. Two negative signs indicate a comment in SQL, so `-$P0` in a template would cause a problem when `$P0` is a negative number such as `-1`. This would translate to `--1`, and would be interpreted as a comment.

Less than (<) and greater than (>) symbols must be escaped

When you create a function template that uses a less than `(<)` or greater than `(>)` symbol, you must use CDATA to escape the symbol. For example, the default mapping for the not-equal-to operator is the following:

```
<FunctionMapping FunctionName="NE">
  <![CDATA[    $P0 <> $P1    ]]>
</FunctionMapping>
```

Alternatively, you can use `<` and `>` to represent the less than and greater than symbols.

If you use a less than or greater than symbol without escaping it, the Integration service returns an error because it cannot parse the mappings.xml file correctly.

Example: Mapping the not-equal-to operator

The mapping for the not-equal-to operator returns a Boolean and does not contain NOT, AND, or OR, so no parentheses are required:

```
<FunctionMapping FunctionName="NE">
    $P0 != $P1
</FunctionMapping>
```

The mapping for the not-equal-to operator uses the function NEQ, which contains parentheses. No additional parentheses are required:

```
<FunctionMapping FunctionName="NE">
    NEQ (CASE WHEN $P0 IS NULL THEN ' ' ELSE $P0 END,
        CASE WHEN $P1 IS NULL THEN ' ' ELSE $P1 END)
</FunctionMapping>
```

The mapping for the not-equal-to operator uses the AND operator. You must enclose it in parentheses:

```
<FunctionMapping FunctionName="NE">
    ($P0 IS NOT NULL AND $P1 IS NOT NULL AND $P0 <> $P1)
</FunctionMapping>
```

Example: Mapping the CONCAT function

The mapping for the CONCAT function uses symbols and returns a string, not a Boolean. You must enclose it in parentheses:

```
<FunctionMapping FunctionName="CONCAT">
    ($P0 + $P1)
</FunctionMapping>
```

Example: Mapping the DATEDIFF function

The mapping for the DATEDIFF function uses symbols and does not return Boolean values. Parentheses are required:

```
<FunctionMapping FunctionName="DATEDIFF" DatePart="yyyy">
    (YEAR ($P2) - YEAR ($P1))
</FunctionMapping>
```

Example: Mapping the CHAR_LENGTH function

In the following mapping, you do not need to place additional parentheses around the argument of the LEN function:

```
<FunctionMapping FunctionName="CHAR_LENGTH">
    (LEN ($P0 + '_' ) - 1)
    <!-- Note: No need for (LEN (( $P0 + '_' )) - 1) -->
</FunctionMapping>
```

Example: Mapping the negative sign (-)

You must place a space after the negative sign to map the subtraction and negation operators:

```
<FunctionMapping FunctionName="SUB">
    ($P0 - $P1)
    <!-- Note: Not ($P0-$P1) -->
</FunctionMapping>

<FunctionMapping FunctionName="NEG">
    - $P0
    <!-- Note: Not -$P0 -->
</FunctionMapping>
```

Using initialization statements

Initialization statements are SQL statements that are sent to the database before executing an Integration service query. Initialization statements are defined by the database and are usually of the form SET <variable> = <value> or SET <variable> <value>. Initialization statements force the database to behave in a way that is compatible with Actuate SQL. Initialization statements take effect only for the session in which the query is executed, so they do not affect queries that are not sent by the Integration service. The Integration service does not send initialization statements when executing SQL stored procedures on the database.

Example: Specifying the behavior of concatenation with NULL

By default, when a string is concatenated with a NULL string, your database returns the original string. Actuate SQL, however, returns a NULL. You check your database documentation and find that there is an initialization statement that changes this behavior. You add the following code to your data source mapping:

```
<Initializers>
    <Initializer>SET CONCAT_NULL_YIELDS_NULL ON</Initializer>
</Initializers>
```


11

Mapping literals and clauses

This chapter contains the following topics:

- Mapping literals: LiteralMapper element
- Mapping clauses

Mapping literals: LiteralMapper element

Literals in Actuate SQL are converted to expressions on the database. For example, the Actuate SQL literal 'Hello' is translated into the expression N'Hello' on the database if the database uses the N-syntax for string literals. You can customize the literal mapping for each Actuate SQL data type using the LiteralMapping element in mappings.xml.

Template format for VARCHAR literals

By default, Actuate SQL string literals are enclosed in single quotes and passed to the database without modification. The template variable \$V represents the string and the single quotes. The default template for string literals is \$V.

Template format for TIMESTAMP literals

TIMESTAMP \$V represents an Actuate SQL timestamp literal, for example TIMESTAMP '2001-02-03 12:11:10'. \$V is the value of the timestamp, including the single quotes. The default template for timestamp literals uses the ODBC escape syntax {ts \$V}.

Example: Mapping VARCHAR and TIMESTAMP literals

Your database uses the N-syntax for Unicode string literals. Also, your JDBC driver does not support the ODBC escape syntax for timestamp literals. As in the FunctionMapping element, \$R represents the return data type of the expression:

```
<LiteralMapper>
  <LiteralMappings>
    <LiteralMapping DataTypeName="VARCHAR">
      <Template>N$V</Template>
    </LiteralMapping>
    <LiteralMapping DataTypeName="TIMESTAMP">
      <Template>CAST ($V AS $R)</Template>
    </LiteralMapping>
  </LiteralMappings>
</LiteralMapper>
```

Mapping clauses

You can customize the mappings for the ORDER BY and GROUP BY clauses.

Mapping the ORDER BY clause: OrderByClauseMapper element

To map the ORDER BY clause, use the OrderByClauseMapper element in mappings.xml. The OrderByClauseMapper element has two attributes, UseSelectedItemIndexes and PushComplexExprs.

UseSelectedItemIndexes attribute

Most databases support expressions in the ORDER BY clause of a query, for example:

```
SELECT contact_last  
FROM CUSTOMERS  
ORDER BY city, contact_last || ', ' || contact_first
```

Some databases, however, do not. The SELECT clause must contain the expression and the ORDER BY clause must reference the expression by index, for example:

```
SELECT city, contact_last || ', ' || contact_first  
FROM CUSTOMERS  
ORDER BY 1, 2
```

If your database does not support expressions in the ORDER BY clause, set the UseSelectedItemIndexes attribute to true, for example:

```
<OrderByClauseMapper UseSelectedItemIndexes="true" />
```

PushComplexExprs attribute

Some databases do not support ORDER BY expressions other than column references. For such databases, an ORDER BY expression that is not a column reference should not be sent to the database. For example, ORDER BY contact_last should be sent to the database, but ORDER BY contact_last || ', ' || contact_first should not.

If your database does not support ORDER BY expressions other than column references, set the PushComplexExprs attribute to false, for example:

```
<OrderByClauseMapper PushComplexExprs="false" />
```

Mapping the GROUP BY clause: GroupByClauseMapper element

To map the GROUP BY clause, use the GroupByClauseMapper element in mappings.xml. The GroupByClauseMapper element has two attributes, UseSelectedItemIndexes and PushComplexExprs.

UseSelectedItemIndexes attribute

Most databases support expressions in the GROUP BY clause of a query, for example:

```
SELECT contact_last  
FROM CUSTOMERS  
GROUP BY city, contact_last || ', ' || contact_first
```

Some databases, however, do not. The SELECT clause must contain the expression and the GROUP BY clause must reference the expression by index, for example:

```
SELECT city, contact_last || ', ' || contact_first  
FROM CUSTOMERS  
GROUP BY 1, 2
```

If your database does not support expressions in the GROUP BY clause, set the UseSelectedItemIndexes attribute to true, for example:

```
<GroupByClauseMapper UseSelectedItemIndexes="true" />
```

PushComplexExprs attribute

Some databases do not support GROUP BY expressions other than column references. For such databases, a GROUP BY expression that is not a column reference should not be sent to the database. For example, GROUP BY contact_last should be sent to the database, but GROUP BY contact_last || ', ' || contact_first should not.

If your database does not support GROUP BY expressions other than column references, set the PushComplexExprs attribute to false, for example:

```
<GroupByClauseMapper PushComplexExprs="false" />
```

Working with collations and byte-based strings

This chapter contains the following topics:

- Working with collations
- Working with byte-based strings

Working with collations

A collation is an algorithm for ordering strings. When an Actuate SQL query is executed, the collation determines the result of sort and comparison operations, including:

- The order of string items produced by the ORDER BY clause
- The grouping of string items produced by the GROUP BY clause
- The strings returned by SELECT DISTINCT
- The result of string comparison operations, for example 'abc' > 'ABC'
- The result returned by the MAX and MIN functions when used with strings
- The result of the LIKE and POSITION operators
- The result returned by the COUNT (DISTINCT) function when used with strings

Databases support one or more collations. The database collation is usually determined by the database locale. The Integration service, however, supports only the Unicode and ASCII code-point collations, which order strings based on the Unicode or ASCII numbers corresponding to each character. Together, the database collation and the Integration service collation determine which operations are sent to the database and which operations must be performed by the Integration service.

About Integration service collations

The Integration service supports two collations: Unicode binary and ASCII case-insensitive. The default collation is Unicode binary. The Integration service collations are explained in Table 12-1.

Table 12-1 Integration service collations

Integration service collation	Description	Examples
Unicode_BIN	Unicode code point order (binary order). All characters are different from one another and are sorted by their Unicode values.	'E' < 'e', since E = U+0045 and e = U+0065. 'o' < 'Ö' (O with an umlaut), since o = U+006F and Ö = U+00D6. 'E' < 'I', since E = U+0045 and I = U+005D.

Table 12-1 Integration service collations

Integration service collation	Description	Examples
ASCII_CI	ASCII code point order, with uppercase characters given the same value as lowercase characters.	'E' = 'e' since case is not considered. Cannot sort Ö, since it is outside the ASCII range (U+00D6). ASCII range is between U+0000 and U+007F. 'E' > '[', since E = U+0065 and [= U+005D.

Choose Unicode_BIN if either of the following statements is true:

- Your databases contain characters outside the ASCII range.
- Strings must be compared case-sensitively.

Choose ASCII_CI if your databases contain only ASCII characters and case-insensitive sorting is required.

When the Integration service collation matches the database collation, all string comparison and sort operations are sent to the database. When the Integration service collation does not match the database collation, some or all string comparison and sort operations must be performed by the Integration service. For this reason, performance is optimized when the Integration service collation matches the collation on as many of your databases as possible. For example, you have five different databases, all of which contain only ASCII data, and either case-insensitive or case-sensitive sorting is acceptable. You can choose either ASCII_CI or UNICODE_BIN as the Integration service collation. However, if four of the databases sort case-insensitively, while the fifth database sorts case-sensitively, you should choose ASCII_CI so that performance suffers only when the Integration service must compare and sort strings from the fifth database.

About database collations

A database collation falls into one of the categories listed in Table 12-2. Refer to your database documentation to determine the appropriate category for your database collation.

Table 12-2 Database collations

Database collation	Description
unicode_bin	Same as Integration service collation UNICODE_BIN.
ascii_ci	Same as Integration service collation ASCII_CI.
null_sensitive	Does not correspond to either Integration service collation. No two characters have the same value.
null	Does not correspond to either Integration service collation. More than one character can have the same value, for example 'E' = 'e'.

About collation implementations

Together, the Integration service collation and the database collation determine which operations are sent to the database and which operations must be performed by the Integration service, as shown in Table 12-3. If the database would not perform an operation in the same way as the Integration service, the operation must be performed by the Integration service.

Table 12-3 Collation implementations

Integration service collation	Database collation	Collation implementation
UNICODE_BIN	unicode_bin	All sort and comparison operations are sent to the database.
UNICODE_BIN	null_sensitive	GROUP BY and SELECT DISTINCT operations, DISTINCT aggregations, LIKE, POSITION, and string equality (=) comparisons are sent to the database. ORDER BY on strings, string comparisons other than equality, MAX, MIN, etc. must be performed by the Integration service.
UNICODE_BIN	null	All operations must be performed by the Integration service.
ASCII_CI	ascii_ci	All sort and comparison operations are sent to the database.
ASCII_CI	null	All operations must be performed by the Integration service.

Specifying the Integration service and database collations

You specify the Integration service and database collations using the Configuration Console.

How to specify the Integration service and database collations

- 1 Choose Servers—Properties—Advanced—Integration Service—General Data Source Information.
 - In Default collation of ASQL strings, select the Integration service collation, as shown in Figure 12-1.
 - In Default collation of target database strings, select the database collation, then choose OK.

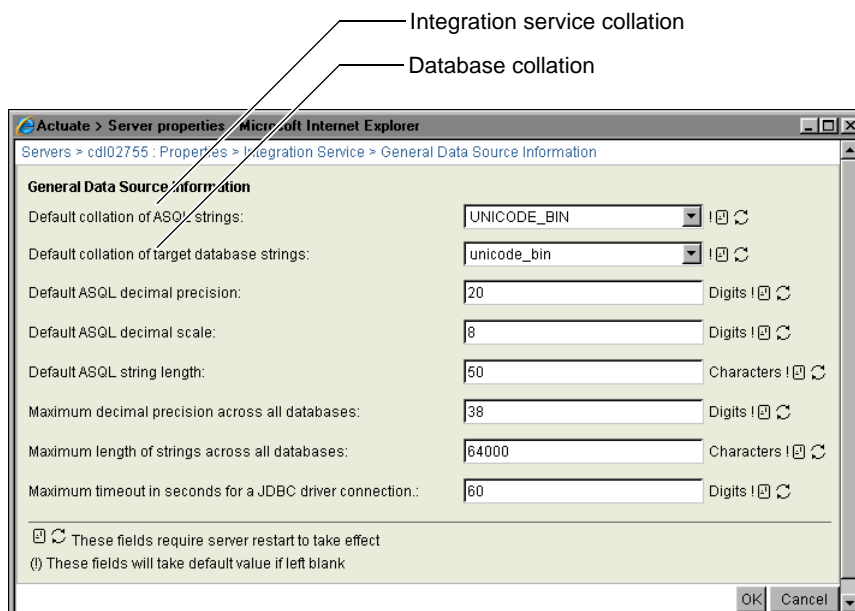


Figure 12-1 Specifying the Integration service and database collations

- 2 Restart the Integration service.

Working with byte-based strings

ASCII and Latin 1 characters, for example the letter A, consist of one byte. Chinese, Japanese, and Korean characters consist of two or more bytes. The Integration service processes strings by character, not by byte. Some databases,

however, process strings by byte, not by character. If the database processes strings by byte and contains multibyte characters, the following string operations should not be sent to the database:

- SUBSTRING, LEFT, and RIGHT functions
- POSITION function
- CAST functions from VARCHAR to VARCHAR where the length is specified, for example CAST (CUSTOMERS.CUSTOMNAME AS VARCHAR (50))
- Actuate SQL parameters specified using syntax such as CAST (? AS VARCHAR (30))

For example, SUBSTRING (CUSTOMERS.CUSTOMNAME, 1, 7) should return the first seven characters of the customer name, not the first seven bytes. If the database processes strings by byte, the SUBSTRING operation should not be sent to the database.

To indicate that string operations should not be sent to the database, set the UseCharStringImplByDefault attribute to true for the database type in mappings.xml, for example:

```
<DataSourceMapper Name="MyDatabaseTypeMapper"
    UseCharStringImplByDefault="true">
...
</DataSourceMapper>
```

Index

Symbols

- _ (underscore) character 96, 181, 247
- : (colon) character 102
- != operator 259
- ? (question mark) character 51, 247
- . (period) character 96, 109
- ' (single quotation mark) character 95, 109, 163
- " (double quotation mark) character
 - column aliases and 70
 - Oracle string comparisons and 208
 - parameter names and 109
 - SQL identifiers and 163, 169
 - XML files and 214
- () (parentheses) characters 100, 191, 258
- [] (brackets) characters 222, 247
- { } (curly brace) characters 234
- @ (at-sign) character 247
- * (asterisk) character 247
- * operator 176, 243
- / (forward slash) character 119, 169
- / operator 176, 243
- \ (backslash) character 96, 181, 248
- > character code 214
- < character code 214
- " character code 214
- % (percent) character 96, 181, 222, 247
- + operator 176, 243
- < (less than) character 214, 258
- < ANY operator 256
- < operator 91, 242
- <= ANY operator 256
- <= operator 92, 242
- <> operator 92, 94, 242
- = operator 84, 91, 242
- > (greater than) character 214, 258
- > ANY operator 256
- > operator 91, 242
- >= ANY operator 256
- >= operator 91, 242
- || operator 180
- operator 176, 243, 260

- (hyphen) character 174, 194
- \$ (dollar sign) character 247
- \$\$ symbol 239
- \$Pn symbol 239
- \$R symbol 239
- \$V variable 262

A

- absolute paths 119, 169, 220
- access permissions 141
- accessing
 - information objects 69
 - maps 69
 - multiple data sources 123
 - Prompt editor 111
 - SQL editor 117
- acserverprofile.xml 139
- Actuate SQL 158, 159, 237
 - See also* SQL statements
- Actuate SQL compiler 86, 121
- Actuate SQL data types 172, 173, 228
 - See also* SQL data types
- Actuate SQL expressions. *See* SQL expressions
- Actuate SQL functions 175, 236, 237
 - See also* SQL functions
- Actuate SQL grammar 163, 164
- Actuate SQL identifiers 163, 168, 169
- Actuate SQL keywords 168
- Actuate SQL parameters 108–110
- Actuate SQL syntax conventions 162
- ADD operator 243
- adding
 - aggregate functions 187
 - column aliases 70
 - columns to queries 70, 169
 - configuration keys 12, 13, 15, 16, 40
 - connection definitions 12, 15
 - connection properties 40
 - database types 212
 - filter conditions 90, 92, 93, 99
 - filters 81

- adding (*continued*)
 - functions to expressions 68
 - functions to queries 159
 - parameters to joins 124
 - parameters to queries 109, 159, 173
 - projects 10
 - server profiles 139
 - summary tables 76
 - tables to queries 86, 159, 172
- addition operator 176, 243
- AdditionalSpecialChars attribute 247
- AdditiveExpression declaration (SQL) 164
- aggregate columns 92, 103, 107
- aggregate expressions 164
- aggregate functions
 - described 187
 - GROUP BY expressions and 162
 - LIKE operator and 159
 - mapping 254
 - OPTIONAL keyword and 193
 - specifying default 74
- Aggregate Type property 74
- aggregation 77, 103, 268
- AggrExpression declaration (SQL) 164
- AggrFuncMapper element 254
- Alias element 231
- aliases
 - column names and 70, 76
 - data types and 231
 - references to 162
 - SQL queries and 118, 169
- Aliases element 231
- alignment 76
- Allow this Source to be used in Cartesian Joins property 90
- allProceduresAreCallable method 220
- allTablesAreSelectable method 219
- alternate names. *See* aliases
- Analysis Type property 70
- analysis type property 76, 77
- AND operator
 - Boolean values 179, 241
 - filter conditions 99
 - join conditions 84
 - not-equal-to operator and 259
 - operator precedence and 258
- AndExpression declaration (SQL) 164

- ANSI SQL 158, 159
- applyIndexing pragma 197
- arguments. *See* parameters
- arithmetic expressions 258
- arithmetic operators 176, 243, 258
- ArithOpMapper element 243
- ASCII case-insensitive collation 266, 267, 268
- ASCII characters 267, 269
- ASCII_CI collation 267
 - See also* ASCII case-insensitive collation
- ascii_ci collation 268
 - See also* ASCII case-insensitive collation
- asterisk (*) character 247
- at-sign (@) character 247
- attribute analysis type 77
- Augment operator 124
- auto suggest option 83, 113
- automatic grouping 105
- averages 188
- AVG function 187, 188, 255

B

- backslash (\) character 96, 181, 248
- balloon help 75
- BasicStringFuncMapper element 245
- BETWEEN operator 91, 175
- BIGINT data type 229
- binary collation 266
- binary types 204
- BIRT Designer Professional 4
- BIRT documents 15, 18
- BIRT iHub servers. *See* iHub servers
- BIRT reports 40, 76, 130
- BIRT Studio 4, 73
- BIT data type 229
- blank values 81, 95
- Boolean operators 241, 242, 255
- Boolean values 179, 258
 - See also* conditional expressions
- BooleanOpMapper element 241
- Box operator 124
- brackets ([]) characters 222, 247
- build error messages 138
- Build Project command 138
- Business Intelligence and Report Tools. *See* BIRT

buttons 75
byte-based strings 269

C

caching data 125
calculated columns. *See* computed fields
calculations
 date values and 97
 DB2 data type mappings and 205
 decimal values and 230
 limitations for 173
 queries and 176
 string values and 230
CallExecutionUnit operator 124
cardinality (joins) 87, 88, 89, 194
CARDINALITY keyword 86
Cardinality property 88, 89
CardinalityType declaration (SQL) 164
Cartesian joins 89
case conversions 180, 206
case sensitivity
 Actuate SQL keywords 168
 connection definition file names 12
 connection properties 43, 44
 information object file names 66
 map file names 45
 map filters 46, 56
 search filters 149
 string comparisons 174
CASE statements 164, 206, 254
CaseExpression declaration (SQL) 164
case-insensitive collation 266
case-insensitive comparisons 174
CaseWhenString attribute 254
cast expressions 173
CAST function 51, 173, 253, 256, 270
CAST statements 164
CAST_AS_DECIMAL function 256
CAST_AS_DOUBLE function 257
CAST_AS_INTEGER function 256
CAST_AS_TIMESTAMP function 257
CAST_AS_VARCHAR function 257
CastExpression declaration (SQL) 164
CastFuncMapper element 256
casting rules 173, 174
 See also typecasting

CatalogFilter element 223
catalogs (maps) 46, 149, 223
categories. *See* column categories
category names 71
Category Path property 74
CDATA keyword 215, 258
CEILING function 177, 205, 244
changing
 column aliases 70
 column names 10, 47, 48
 configuration files 204, 214, 234, 238
 data 148, 151
 data types 48
 filter conditions 101
 maps 50
 output column properties 80
 parameter names 10, 53, 59, 63
 parameter properties 115
 parameters 151
 project directories 10
 resources 138
 SQL statements 117, 118
 tables 48
 translation keys 134
 translation strings 134
 views 48
CHAR data type 52, 229
CHAR_LENGTH function 180, 245, 259
CHAR_LITERAL token (SQL) 163
character conversions 180, 206
character data types 88
character encoding 24, 27
character patterns 96, 181, 247, 248
character sets 18, 22, 24, 269
character strings. *See* strings
characters
 blank values 95
 column aliases and 70
 database collation and 266
 filter conditions and 94
 LIKE operator and 246, 248
 literal text and 163
 MATCH operator and 247
 ODBC escape sequences and 234
 parameter names and 109
 parameter values and 159

- characters (*continued*)
 - relative paths and 119
 - resource names and 4
 - SQL identifiers and 163
 - string operations and 270
 - XML files and 214
- Charset property 18, 22
- Clear Impact command 152
- CLIENT_LOCALE variable 27
- code pages 24, 27
- code points 174, 266, 267
- code-point collations 266
- collation 209, 266–269
- Collection property 18
- colon (:) character 102
- column aliases 70, 76, 118, 169
- column categories
 - creating maps and 45, 48
 - defining output columns and 71–72
 - displaying 72
- Column Categories page 71, 72
- column descriptions 75
- column headings 75
- column name duplication 11
- column names
 - See also* column aliases; column headings
 - changing 10, 47, 48
 - converting to expressions 84
 - creating information objects and 47, 70, 76
 - displaying 75
 - entering in SQL statements 169
 - updating 10, 48
- ColumnAlias declaration (SQL) 164
- columns
 - See also* computed columns; output columns
 - adding to queries 70, 169
 - changing order of 104, 105
 - comparing values between 97
 - defining joins and 84, 88, 89
 - deleting 11, 105, 107
 - filtering blank values in 95
 - filtering null values in 95
 - filtering on 47, 71, 81, 92, 95
 - grouping on 103, 104, 105
 - localizing information objects and 130, 132, 134, 135
 - locating 71
 - mapping to database 49, 228
 - prompting for values and 110, 111
 - propagating values for 10, 138
 - referencing 159
 - removing dropped 49
 - renaming 10, 47, 48
 - retrieving type information for 228, 229, 231
 - returning subsets of 124
 - searching for 148
 - setting analysis type for 76, 78
 - viewing blank values in 95
 - viewing null values in 95
- Columns page (graphical information object editor) 69
- Columns page (SQL editor) 119
- comma-separated values files 19
- comments (SQL) 169, 258
- comparison operators 94, 175, 176, 242, 258
- ComparisonOpMapper element 242
- comparisons
 - collation and 266
 - date-and-time values 95
 - filter conditions 92, 96
 - range of values 94
 - string values 96, 174, 175, 207
 - values in expressions 175
 - values in multiple columns 94, 97
- Compile IO and dependents command 129
- Compile IO button 75, 116
- compiler 86, 121, 219
- compiler errors 47, 70
- compiling 129, 138
- computed columns 10, 103
 - See also* computed fields
- computed fields 75, 105, 190
- CONCAT function 207, 245, 259
- concatenation 163, 180, 207, 260
- concatenation operator 180
- Conceal Value property 74, 113
- CondExpr declaration (SQL) 164
- CondFuncMapper element 254
- conditional expressions 164
 - See also* Boolean values
- ConditionalPrimary declaration (SQL) 164
- configurable database types 212–214

- configuration files
 - changing 204, 214, 234, 238
 - connection properties in 12, 15, 40
 - connection types in 204, 212, 218, 220
 - database function mappings in 234, 236, 238
 - database types in 204, 212, 213, 214
 - externalized connections in 40, 41, 43, 44
 - JDBC drivers and 221
 - locating connection 41
 - passthrough security and 40
 - SQL statement mappings in 263
 - string operations and 270
- Configuration key property 12, 15
- configuration keys 12, 13, 15, 16, 40
- conjunction 179
- connection definition files 12
- connection definitions
 - See also* connections
 - creating 11–28
 - database schemas and 28
 - moving projects and 39
 - non Unicode character sets and 24
 - searching for 148
 - viewing file dependencies for 150
- connection parameter types 224
- connection parameters 43, 202, 218, 223
- connection properties
 - See also* connection definitions
 - data source configurations 11, 13, 40
 - DB2 databases 41
 - externalizing 39–44
 - iHub profiles 139
 - ODA data sources 15, 16, 44
 - ODBC databases 12
 - preconfigured connection types 17
- connection strings 202, 218, 222
- connection types
 - configuring 204, 212, 213, 218, 220
 - described 202
 - displaying 223
 - externalizing properties for 41, 43
 - listed 17
 - naming 220
 - setting parameters for 202, 223
- ConnectionParam element 223
- ConnectionParams element 42, 43, 220, 223
- ConnectionProperties element 222
- connections
 - creating custom data source 15
 - creating database 12, 220
 - encrypting and decrypting 28
 - locating configuration files for 41
 - retrieving properties for 12, 15
 - setting character encodings for 24
 - setting parameters for. *See* connection parameters
 - setting properties for. *See* connection properties
 - setting run time 40
 - specifying port numbers for 13, 139
 - testing 14, 17
- ConnectionString element 222
- ConnectionType attribute 225
- ConnectionType element 42, 43, 213, 220
- ConnectionTypes element 220, 221
- ConnectOptions element 40, 42, 43, 44
- control strings (units of time) 184
- control type constants 114
- control types
 - changing 115
 - entering display names for 82, 111
 - prompting for input and 113, 115
 - specifying 75, 82, 114
- CONVERT function 257
- Copy Information Objects command 141
- cost-based optimization (joins) 195–197
- COUNT function 187, 193, 255
- COUNT_ROWS function 255
- counting non-null values 187
- country codes 131
- Create SQL map option 14
- creating
 - column aliases 70
 - column categories 71
 - connection definitions 11–28
 - custom filters 75
 - data filters 81, 90, 98
 - function templates 239
 - information objects 4, 66–67
 - joins 83–85, 89
 - list of values 81, 82, 110, 111

- creating (*continued*)
 - maps 45–64
 - projects 10
 - queries. *See* queries; textual queries
 - server profiles 139
 - subqueries 171–172
 - summary tables 76
- Credentials property 13
- CSV files 19
- curly brace ({}) characters 234
- CURRENT_DATE function 185
- CURRENT_TIMESTAMP function 184
- CURRENT_USER function 188
- Custom Data Source Properties dialog box 16
- custom data sources. *See* ODA data sources
- Custom driver class property 23
- customizing
 - cast functions 256
 - data filters 75
 - null functions 252
 - numeric functions 244
 - query mappings 262–264
 - SQL operators 255
 - string functions 245
 - substring functions 245

D

- dashboard designs 148, 153
- dashboard reports 28
- dashboards 74, 76
- data
 - See also* values
 - aggregating. *See* aggregation
 - aligning 76
 - caching 125
 - changing 148, 151
 - creating joins and 84, 86, 88, 90, 195
 - filtering. *See* data filters; filtering data
 - grouping 102–107, 162
 - retrieving from
 - databases 90
 - information objects 4, 110, 127
 - multiple data sources 4, 123
 - simulating 126
- Data Connection Definition command 12, 15
- data connection definition files 12

- Data Connection Definition page 12, 15
- data connection definitions
 - See also* connections
 - creating 11–28
 - database schemas and 28
 - moving projects and 39
 - non Unicode character sets and 24
 - searching for 148
 - viewing file dependencies for 150
- data filters
 - See also* filtering data
 - building maps and 46, 53, 56, 58, 62
 - comparing date-and-time values and 95
 - comparing numeric values and 96
 - comparing string patterns and 96
 - comparing values in multiple columns and 97
 - creating 81, 90, 98
 - customizing 75
 - defining output columns and 47, 71, 75
 - disabling 75
 - excluding null or blank values and 95
 - excluding sets of values and 94, 99
 - prompting for 101
 - removing columns and 11
 - search operations and 149
 - setting conditions for. *See* filter conditions
 - setting control type for 75
 - setting default values for 75
 - setting evaluation order for 100
 - specifying parameters as 93
- Data Preview view 6, 121
- data repository. *See* Encyclopedia volumes
- data rows. *See* rows
- data sets
 - See also* result sets
 - filtering 90, 94, 99
 - mapping to ODA data sources and 61, 64
 - viewing file dependencies for 151
- Data source connection properties page
 - JDBC connections and 202, 218
 - ODA connections and 16, 61
 - ODBC connections and 13, 202, 218
- Data Source page (New Maps) 45, 52, 55
- data sources
 - See also* specific type

- connecting to 12, 15, 39, 40
- defining joins and 84, 86, 88, 89, 195
- externalizing connection properties for 39–44
- locating configuration files for 41
- querying remote 175, 176
- retrieving data from multiple 4, 123
- retrieving distinct values from 70
- setting connection properties for 11, 13, 17
- testing connections for 17
- viewing queries for 122, 127
- Data Type property 75, 113
- data types
 - accessing non-native 230
 - assigning to parameters 109, 113, 224
 - casting 159, 173, 256
 - changing 48
 - displaying 49, 77, 119, 120
 - filtering null values and 95
 - join column properties and 88
 - mapping 204, 228, 229, 231
 - mapping queries and 51, 53
 - mapping stored procedures and 56, 58, 59
 - mapping to
 - DB2 databases and 205
 - Informix databases and 206
 - ODA data sources and 60
 - Oracle databases and 207
 - SQL databases and 209
 - Sybase databases and 211
 - redefining aliases for 231
 - returning from function calls 239
 - SQL queries and 158, 172
 - supported 228
- database collation 209, 266–269
- database connection types. *See* connection types
- database drivers. *See* drivers
- Database property 18, 19, 21, 22, 23
- database schemas 28, 46, 56, 149
- database servers. *See* servers
- database types
 - See also* specific type
 - applying default mappings and 212, 213
 - configuring 213, 214, 224, 270
 - defining connection types for 218, 220
 - described 202, 204, 212
 - mapping functions for 236, 238
 - running queries and 234
- databases
 - See also* data sources; specific database type
 - connecting to 12, 220, 224
 - creating queries for 162, 171, 172, 203
 - disabling function mappings for 236
 - externalizing connections for 41, 43
 - mapping literal strings for 262
 - mapping to data in 45–48, 203
 - mapping to data types in 228, 230
 - mapping to functions in 234, 237, 238
 - mapping to result sets from 51, 55
 - ordering strings in 266, 267, 268, 269
 - retrieving data from 90
 - running queries from 203, 237
 - searching 148
 - setting character encodings for 24
 - testing connections for 14
- DatabaseType element 213, 224
- DatabaseTypes element 220, 224
- DataSourceMapping attribute 214, 225
- datasources.xml 43, 214, 220
- DataType declaration (SQL) 164
- DataType element 230
- DataTypeMapper element 229, 239
- DataTypes element 229
- DATE data type 229
- date functions 248, 249, 251, 252
- date stamps 240, 248, 249, 251, 252
- date values 95, 98, 184, 240
- DATEADD function 185, 238, 251
- DateAdd function 209
- DateAddMapper element 251
- DATEDIFF function 97, 185, 238, 249
- DateDiff function 209
- DATEDIFF function mappings 240, 250, 259
- DateDiffMapper element 241, 250
- DatePart attribute 238
- DATEPART function 186, 238, 248, 249
- DatePart function 209
- DatePartMapper element 249
- DATESERIAL function 186, 252
- DateSerialMapper element 252
- DB_LOCALE variable 27

- DB2 data types 205
- DB2 databases
 - connecting to 18, 204
 - creating queries for 203
 - externalizing connections for 41, 42
 - mapping to 205
 - renaming stored procedures for 55
 - setting character encodings for 24
- .dcd files 12
 - See also* connection definitions
- DECIMAL data type 172, 173, 229, 230
 - See also* decimal values
- decimal precision 173, 230
- decimal separators 96, 109
- decimal values
 - aggregating data and 187
 - calculating data and 176
 - converting 205
 - creating SQL queries and 172, 173
 - mapping to data types and 211, 230
 - rounding 178
 - setting precision for 230
- DECIMAL_LITERAL token (SQL) 163
- decrypt method 30, 36
- decryption 28
- decryption algorithms 28
- Default Analytics button 77
- Default collation of ASQL strings option 269
- Default collation of target database strings
 - option 269
- default function templates 234, 257
- default mappings file 212
- default names 53, 59
- Default Value property 75, 113
- default values
 - analytics properties 76, 77
 - connection parameters 224
 - data filters 75
 - output parameters 59
 - overriding 11
 - run-time parameters 113
 - SQL parameters 109
- DefaultDecimalPrecision variable 256
- DefaultDecimalScale variable 256
- DefaultStringLength variable 257
- DefaultValue attribute 224
- Define Default Column Analytics wizard 77, 78
- deleting
 - column categories 72
 - columns 11, 105, 107
 - filter conditions 101
 - join conditions 86
 - output columns 71
 - parameters 11, 110
 - query execution plans 130
- DELIMIDENT variable 27
- delta values 185, 249, 251
- dependent joins 87, 124, 160
- DependentJoin operator 124
- derived tables 172
- Describe Query button 119, 120
- Description Key property 75, 114
- Description property 75, 81, 114, 130
- design files 139, 145
- designs
 - creating filters for 75, 81, 82
 - creating queries and 169
 - downloading 144, 145
 - externalizing connections and 40
 - file dependencies and 148, 150, 153
 - localizing information objects and 130
 - publishing 139
 - specifying control type for 75
- developers 4
- dimension analysis type 76
- directory paths
 - column categories 74
 - connection type configurations 204, 212, 220
 - database type mappings 213, 214, 225
 - externalized connection properties 41, 43, 44
 - impact scripts 153
 - information objects 82, 169
 - JDBC drivers 220, 222
 - SQL function mappings 169, 234, 236
 - SQL queries 119
 - WSDL files 23
 - XML data sources 23
- Disabled attribute 238, 246, 247
- Disabled value 75
- disjunction 179

- Display attribute 223
- Display Control Type property 75, 114
- display control types. *See* control types
- Display Format property 75, 114
- Display Length property 75, 114
- Display Name Key property 75, 114
- Display Name property 75, 81, 114, 130
- display names
 - connection types 223
 - control types 82, 111
 - localizing information objects and 130
 - output columns 75
 - parameter prompts 114
- displaying
 - column categories 72
 - column names 75
 - data types 49, 77, 119, 120
 - error messages 6, 138
 - hidden messages 7
 - impact reports 153
 - information objects 68
 - map files 68
 - output columns 47, 53, 62, 119
 - parameters 120
 - project model diagrams 152
 - SQL queries 6, 122, 127
- DisplayName attribute 225
- DISTINCT keyword 70, 77, 162
- Distinct Values Count property 88
- Distinct values only option 70
- DIV operator 243
- division 176, 177
- DNS servers 18
- Do Not Prompt property 75, 114
- Do not show this message again setting 6
- documentation ix
- documents 15, 18
 - See also* reports
- dollar sign (\$) character 247
- DOUBLE data type 172, 173, 205, 229
 - See also* double values
- double quotation mark (") character
 - column aliases and 70
 - Oracle string comparisons and 208
 - parameter names and 109
 - SQL identifiers and 163, 169
 - XML files and 214
- double values 172, 173, 175, 176, 205
 - See also* numeric values
- DOUBLE_LITERAL token (SQL) 163
- Download from Server command 145
- downloading files 144, 145, 153
- Driver class path property 23
- DriverName attribute 221
- drivers
 - configuring JDBC 221
 - connecting to databases and 12, 18, 212, 218, 221
 - connecting to ODA data sources and 15
 - creating map files and 56, 228, 229, 231
 - customizing 23
 - installing JDBC 218, 219, 220
 - installing non-Unicode characters and 24
 - naming 221
 - running platform-independent queries and 234
 - specifying paths to 220, 222
- drop-down lists 75, 83, 113
- Dup operator 124
- duplicate rows 70
- dynamic filters 75, 83
- Dynamic list of values option 83, 111

E

- e.Reports Data Connector for iHub data
 - sources 15, 17
- Eclipse IDE 4
- Eclipse platforms 31
- Eclipse-based encryption extensions 28
- Edit in SQL text editor option 67, 117
- Edit SQL button 117
- editors 153
- ELSE keyword 254
- ElseString attribute 254
- empty strings 207
- empty values 95
- Enable cost based optimization variable 197
- EnableCBO pragma 195, 196
- encoding 18, 24, 27
- Encoding for XML source and schema
 - property 24
- encrypt method 29, 30, 36
- encryption 28

- encryption algorithms 28
- encryption extension point plug-in 28–39
- encryption extension point plug-in loading errors 39
- encryption plug-in class 35
- EncryptionProviderID.exsd 28
- encycAnalyze.bat 153, 154
- encycDownload.bat 153
- encycFind.bat 153, 155
- Encyclopedia volumes
 - accessing information objects in 169
 - determining file dependencies 154
 - downloading files from 144, 145, 153
 - generating impact reports for 153, 155
 - publishing to 131, 138, 140, 142
 - specifying server profiles for 139
- END keyword 254
- EndString attribute 254
- environments 39
- .epf files
 - ODA data sources and 60, 62
 - stored procedures and 54, 59
- EQ operator 242
- Equal to operator 91
- equality comparisons 175
- equality operator 84, 242
- equijoins 87, 88, 125
- error messages 6, 138
- errors
 - column aliases and 70
 - column names and 47
 - data type mappings and 209, 211, 231
 - escape sequences and 248
 - SQL operators and 258
 - SQL queries 173
 - unknown escape sequences and 234
 - unsupported functions and 236
- escape characters 159, 167, 181, 248
- ESCAPE clause 246
- ESCAPE keyword 181
- escape sequences 212, 214, 230, 234
- EscapeTemplate attribute 247
- EXISTS operator 256
- ExplicitInnerOuterType declaration (SQL) 164
- ExplicitJoinType declaration (SQL) 164
- exponentiation 178

- Expression Builder 67–68, 70
- Expression property 75
- ExpressionList declaration (SQL) 165
- expressions
 - adding column names to 84
 - adding functions to 68
 - adding SQL operators to 258
 - changing column names and 10, 49
 - comparing values in 175
 - creating computed fields and 75
 - creating joins and 84, 87
 - creating queries and 67, 173, 175
 - defining output columns and 70
 - filtering data and. *See* filter expressions
 - grouping data and 162
 - mapping literal strings and 262
 - matching character patterns and 181
 - setting default values and 54
- extension points, selecting 33
- external procedure object files
 - ODA data sources and 60, 62
 - stored procedures and 54, 59
- external procedures 151
- externalizing connection properties 39–44

F

- facets (defined) 172
- FakeData operator 126
- FakeFileData operator 126
- false values. *See* Boolean values
- fields. *See* columns
- file dependencies 148, 150, 154
- file names 12, 45, 66
- file paths. *See* directory paths
- files 144, 148, 150
- filter conditions
 - adding multiple 98, 99, 100
 - aggregating data and 107, 108
 - changing 101
 - defining 90, 92, 93, 97
 - deleting 101
 - grouping 99, 100
 - including parameters in 102
 - selecting multiple values for 94
- Filter Conditions dialog box 91, 92, 101, 102
- filter expressions
 - changing column names in 10

- comparison operators in 94, 95, 96
- empty or blank values in 95
- filter conditions in. *See* filter conditions
- functions in 97
- grouping conditions in 100
- literal characters in 96
- logical operators in 99
- multiple values in 94, 97
- Filter property 71, 75, 81, 82
- filter specification. *See* filter expressions
- FilterClause declaration (SQL) 165
- filtering data
 - at run time 101
 - in tables and views 46, 149
 - information objects and 90, 99, 107
 - multiple conditions and 99
 - report designs and 81, 82
- filtering error messages 138
- filters
 - building maps and 46, 53, 56, 58, 62
 - comparing date-and-time values and 95
 - comparing numeric values and 96
 - comparing string patterns and 96
 - comparing values in multiple columns and 97
 - creating 81, 90, 98
 - customizing 75
 - defining output columns and 47, 71, 75
 - disabling 75
 - excluding null or blank values and 95
 - excluding sets of values and 94, 99
 - prompting for 101
 - removing columns and 11
 - search operations and 149
 - setting conditions for. *See* filter conditions
 - setting control type for 75
 - setting default values for 75
 - setting evaluation order for 100
 - specifying parameters as 82, 93, 101
- Filters page (graphical information object editor) 92, 93, 97, 99, 101
- FILTERS statements 165
 - See also* SQL statements
- fixed point numbers 172
 - See also* decimal values
- flat file data sources 15, 18, 61, 126
- flat joins 124, 125
- Flatfile style property 19
- FLOAT data type 229
- floating point numbers 172, 175, 176, 229
 - See also* numeric values
- FLOOR function 177, 205, 235, 244
- Folder property 18
- folders
 - connecting to flat file data sources and 18
 - localizing information objects and 130, 131
 - publishing information objects and 140, 142
 - saving projects and 10
 - storing connection definitions and 11
 - storing map files and 45, 54, 60
- foreign keys 77
- formats
 - join column properties 88
 - output columns 75
- forward slash (/) character 119, 169
- fractional numbers 211
- FROM clause 165
 - See also* SQL statements
- FromClause declaration (SQL) 165
- FromTableName declaration (SQL) 165
- FromTableReference declaration (SQL) 165
- function mapping examples 239
- function templates 234, 236, 237, 239, 257
- FunctionCallOrColumnRef declaration (SQL) 165
- FunctionMapping element 238, 262
- FunctionMappings element 241
- FunctionName attribute 238
- functions
 - aggregation and. *See* aggregate functions; aggregation
 - compatibility with 236
 - DB2 databases and 205
 - disabling default mappings for 236, 241
 - filtering data and 97
 - mapping 234, 236, 238
 - null values and 252
 - numeric data types and 177, 244
 - Oracle databases and 207
 - overloading 238
 - SQL databases and 209
 - SQL queries and 51, 68, 159, 173
 - string data types and 179, 245

functions (*continued*)

- substrings and 181, 183, 245
- Sybase databases and 211
- testing driver version and 219
- timestamp values and 184
- type casting and 256
- volume user and 188

G

- GE operator 242
- GE_ANY function 256
- Generate Impact Report command 153
- Generate Query button 83
- generating impact reports 153, 155
- Generic SQL type codes 212
- Generic_ODBC mappings file 236, 238
- GenericSQLType attribute 230
- getJDBCMinorVersion method 219
- getProcedures method 219
- getTables method 219
- graphical information object editor
 - accessing 67
 - aggregate columns and 108
 - categorizing output columns and 71, 73
 - changing filter conditions with 101
 - creating joins and 83, 84
 - creating queries and 68, 117
 - defining output columns and 69
 - defining parameters with 109, 115
 - editing queries and 117
 - filtering data with 82, 92, 93, 97, 99, 100
 - grouping data with 104, 107
 - synchronizing parameters with 116
- graphical query editor 67, 158
 - See also* Information Object Query Builder
- Greater Than operator 91
- Greater Than or Equal to operator 91
- GreedyMatchChar attribute 246, 247
- GROUP BY clause
 - See also* SQL statements
 - adding expressions to 162, 165
 - creating 102–107
 - deleting 106
 - filtering aggregate data and 107
 - mapping 263
 - removing columns from 105, 107
- Group By page 104, 107

- GroupByClause declaration (SQL) 165
- GroupByClauseMapper element 263
- grouping data 102–107, 162
- grouping filter conditions 99, 100
- groups
 - changing column order for 104, 105
 - disabling automatic grouping and 105
 - nesting rows and 125
 - returning distinct values for 70
 - returning from queries 162
 - viewing available columns for 105
- > character code 214
- GT operator 242
- GT_ANY function 256

H

- Has Null property 75
- HAVING clause 107, 162, 165
 - See also* SQL statements
- Having page 108
- HavingClause declaration (SQL) 165
- Heading Key property 75, 114
- Heading property 75, 81, 114, 130
- Help Text Key property 76, 114
- Help Text property 75, 81, 114, 130
- help topics. *See* online documentation
- Hidden Messages command 7
- Hidden Messages list 6
- hidden parameters 114
- hidden strings 224
- hiding dialog messages 6
- hiding specific values 113
- hints 189
- Horizontal Alignment property 76, 114
- Host property 19
- hyphen (–) character 174, 194

I

- IANAAppCodePage property 18, 19, 23, 24, 222
- IBM DB2 databases. *See* DB2 databases
- IDENTIFIER token (SQL) 163
- identifiers (SQL) 163, 168, 169
- IEncryptionProvider interface 29, 30
- iHub servers
 - accessing BIRT reports and 18

- accessing Informix databases and 27
- accessing ODA data sources and 15, 17
- character encoding and 24
- configurable database types for 212
- creating profiles for 139
- deploying encryption plug-in to 38
- externalizing connection properties for 40
- installing drivers for 219, 220
- logging plug-in loading errors for 39
- preconfigured connection types for 17, 204
- publishing information objects to 141
- running information objects on 30, 197, 198
- viewing profiles for 6
- iHub System error messages 6
- iHub volumes. *See* Encyclopedia volumes
- illegal characters 169
- impact paths 152
- impact reports 152, 153, 155
- IN operator 91, 94, 102, 176, 256
- indexed columns 76, 77
- Indexed property 76
- indexes (SQL queries) 197
- Information Object command 66
- information object data sources 127, 130
- Information Object Designer
 - creating queries and 162
 - described 4
 - filtering data and 99, 100
 - grouping data and 104
 - overriding default values for 11
 - previewing data and 121
 - prompting for input and 111
 - publishing project files with 141, 143
 - saving passwords for 139
 - viewing queries and 122, 123, 127
- information object editors. *See* graphical information object editor; SQL editor
- information object files 66, 139, 140, 145
- Information Object Query Builder 73, 158, 189
- information objects
 - accessing multiple data sources and 123
 - building from information objects 70, 79, 115
 - building from map files 45, 47, 51, 54, 60
 - building queries for 67, 68, 89, 117, 158
 - changing items in 10
 - creating 4, 66–67
 - defining computed fields and 190
 - defining joins for 83–85, 86, 88, 89, 195
 - defining optional tables for 189–193
 - defining output columns for 69–71, 79, 138
 - defining parameters for 108–116
 - deleting joins for 86
 - deleting query plans for 130
 - disabling indexing for 197
 - displaying file dependencies for 150
 - displaying output for 6, 120, 121
 - displaying queries for 6
 - downloading 144, 145
 - filtering data for 81, 82, 90, 99, 107
 - grouping data for 102–107
 - localizing 130–135
 - naming 66
 - opening 69
 - publishing 140, 141, 142, 143
 - referencing 82, 169
 - removing dropped columns from 49
 - renaming 79
 - retrieving data and 4
 - running 30
 - saving query plans for 127, 128, 129
 - selecting 68
 - testing 126
- Information Objects folder 66
- Informix data types 206
- Informix databases
 - connecting to 19, 27, 204
 - externalizing connections for 41
 - mapping to 206
 - renaming stored procedures for 55
 - setting character encodings for 24
- inheriting properties 79–81, 115, 138
- initialization statements 260
- inner joins 85, 161
- input 81, 82, 102, 111
- input parameters 54, 56, 60, 114
 - See also* stored procedures
- installation
 - character sets 22
 - JDBC drivers 218, 219, 220

- INTEGER data type 172, 173, 229
- INTEGER_LITERAL token (SQL) 164
- integers 163, 172, 173, 176, 229
 - See also* numeric values
- Integration service
 - calculating data values and 176
 - comparing data values and 174, 175
 - configuring database types and 213
 - connecting to databases and 27, 218
 - defining joins and 88
 - mapping data types and 228, 230, 231
 - mapping SQL functions and 234, 237, 239, 257
 - ordering strings and 266, 267, 268, 269
 - running queries and 162, 203, 230, 235
 - setting collation for 269
- intersection operations 158
- intsrvsources.xml 41, 204
- IO Design perspective
 - building projects and 138, 148
 - creating information objects and 4
 - creating queries and 158
 - described 5
 - duplicating column names and 11
 - viewing column categories and 73
- IO Designs folder 140
- .job files 66
 - See also* information object files
- IP addresses 18, 23
- IS NOT NULL operator 91, 95, 179, 259
- IS NULL function 253
- IS NULL operator 91, 95, 179
- IteratorAsLeaf operator 126
- iterators 124, 125, 126

J

- Java Runtime Environment 219
- JDBC connection strings 202, 218
- JDBC data sources 221, 222
- JDBC drivers
 - See also* drivers
 - connecting to databases and 12, 212, 218, 221
 - creating map files and 56, 228, 229, 231
 - creating SQL statements and 234
 - installing 218, 219, 220

- jdbcCompliant function 219
- JDBCdriver element 220, 221, 222
- join algorithms 86, 88, 160, 196
- join control syntax 160
- join operators 124, 125, 126
- join types 85
- JoinCondition declaration (SQL) 165
- JoinElement declaration (SQL) 165
- JoinExpression declaration (SQL) 165
- joins
 - accessing multiple data sources and 86, 195
 - building subqueries and 171
 - creating 83–85, 89
 - defining algorithms for 87, 88, 160
 - defining cardinality of 87, 88, 89, 194
 - defining dependent 87, 124, 160
 - disabling cost-based optimization for 195, 196
 - optimizing 86, 88
 - removing conditions for 86
 - setting column properties for 88, 89
 - setting conditions for 84, 87, 88
 - specifying optional tables for 189
 - testing 123
- Joins page 84, 88
- JoinSpec declaration (SQL) 165
- JRE compatibility 219

K

- keywords (Actuate SQL) 168

L

- language code 131
- LE operator 242
- LE_ANY function 256
- leaf operators (queries) 126
- LEFT function 181, 246, 270
- LEFT OPTIONAL keywords 192
- LEN function 259
- Length declaration (SQL) 165
- length function 180
- Less Than operator 91
- Less Than or Equal to operator 92
- libraries 151

- LibraryPath element 222
- LIKE expressions 248
- LIKE operator
 - changing escape character for 248
 - customizing 246
 - disabling 247, 248
 - displaying blank values and 95
 - mapping to 246–248
 - matching characters and 92, 96, 247, 248
 - SQL queries and 159, 181
- LikeOpMapper element 246
- Linux servers 18, 24, 27
- lists
 - displaying at run time 81, 82, 110, 111
 - selecting values from 83, 94, 113
- literal characters 96, 163, 166, 181
- literal integers 164, 165, 166
- literal strings 159, 164, 262
- LiteralMapper element 262
- LiteralMapping element 262
- local parameters 115
- locale code 131
- locales
 - See also* localizing information objects
 - byte-based strings and 269
 - character encoding and 24, 27
 - column properties and 75
 - database collation and 266
 - date-and-time filters and 95
 - numeric comparisons and 96
 - selecting 132
- Localization button 132
- Localization dialog box 133, 134, 135
- Localization folder 130
- localization properties files 130, 131
- localizing information objects 130–135
- Location element 222
- logical operators 99, 179
- logical values. *See* Boolean values
- login IDs 18, 23
- LONG VARCHAR data type 229
- LOWER function 180, 245
- < character code 214
- LT operator 242
- LT_ANY function 256
- LTRIM function 182, 245

M

- Management Console 139
- Map command 45
- map files
 - See also* maps
 - displaying 68
 - naming 45, 52, 79
 - opening 69
 - setting directory location for 225
 - storing 45, 54, 60
- Map page 52
- Map stored procedures option 14
- Map tables button 17
- Map tables option 14
- mapping
 - data types 204, 228, 229, 231
 - database functions 234, 236, 238
 - database tables and views 45–48, 203, 228
 - literal strings 262
 - ODA result sets 17, 60–64
 - SQL functions 234, 236, 237
 - SQL operators 236, 237, 238, 258
 - SQL queries 51–54
 - stored procedures 54–60, 228
- mapping examples (SQL functions) 239
- mapping examples (SQL operators) 259–260
- mappings.xml 204, 214, 234
- maps
 - accessing databases and 228
 - adding to projects 45, 52, 55
 - changing 10, 47, 50
 - column property inheritance and 79, 81
 - creating 45–64
 - defining joins for 84, 86, 88, 90, 195
 - referencing 82, 169
 - removing items from 11
 - renaming output columns in 47
 - retrieving data from 84
 - retrieving type information for 228, 229, 231
 - running queries and 203
 - searching for 148
 - selecting 68
 - unsupported functions and 236
 - updating 48, 49, 51
 - viewing file dependencies for 150

- maps (*continued*)
 - viewing output for 6
- Maps page 46, 56, 57, 61
- masked property 29, 38
- masks 224
- MATCH operator 247
- matching character patterns 96, 181, 247, 248
- Materialize operator 125
- mathematical operators 176
- MAX function 187, 255
- Max memory per query parameter 88
- Max Value property 88
- maximum values 187
- MaxSize attribute 230
- measure analysis type 77
- memory 88, 197
- merge joins 87, 161
- MergeJoin operator 125
- Message Distribution Service 139
- messages 6, 7
- metadata 54, 60
- Microsoft applications. *See* specific application
- Microsoft Windows. *See* Windows systems
- MIN function 187, 255
- Min Value property 88
- minimum values 187
- MinRowsForIndexing pragma 197
- missing values 95
- MOD function 177, 244
- Move operator 125
- moving
 - column categories 72
 - output columns 48, 71, 72
 - project files 39
- MULT operator 243
- MultiAugment operator 125
- multibyte characters 270
- multiline comments (SQL) 169
- multiplication 176
- MultiplicativeExpression declaration (SQL) 165
- MultiRowBoolFuncMapper element 255
- MySQL Enterprise database type 212
- MySQL Enterprise databases 19, 41, 204
- MySQL Enterprise login accounts 19
- MySQL Enterprise named instances 20

N

- Name attribute
 - ConnectionParam element 223
 - ConnectionType element 220
 - DatabaseType element 225
 - DataType element 230
- Name property 76, 79, 114
- named parameters 51
- named servers 20, 23
- NamedParameter declaration (SQL) 165
- names
 - as SQL identifiers 169
 - changing 10
 - parameters and 53, 59
 - stored procedures and 55
- naming
 - connection types 220, 225
 - data connection definition files 12
 - database schemas 28
 - database types 225
 - external procedure object files 54, 60
 - information objects 66
 - JDBC drivers 221
 - map files 45, 52, 79
 - output columns 70
 - parameters 109, 114
 - projects 10
 - queries 61
 - resources 4
 - result sets 57
 - server profiles 139
- naming restrictions 4
 - See also* case sensitivity
- native SQL data types 53, 59
- Navigator view 5, 150
- NCHAR data type 52, 206
- NE operator 242
- NEG operator 243
- negation 179, 258, 260
- negation operator 243, 260
- negative (–) signs 258, 260
- NEQ function 259
- Nest operator 125
- nested loop joins 87, 126, 160
- NestedLoopJoin operator 126
- nesting joins 124, 125

- networked environments 17
- New Data Connection Definition wizard 12, 15
- New Information Object wizard 66
- new line characters 168
- New Maps wizard 45, 52, 55
- New Project wizard 10
- New Server Profile wizard 139
- node operators (queries) 124
- non-native data types 230
- NoOp operator 127
- NOT BETWEEN operator 92, 94
- Not Equal to operator 92
- NOT IN operator 92, 256
- NOT LIKE operator 92, 94, 95
- NOT operator 99, 179, 241, 242, 258
- NOT_IN function 256
- not-equal-to operator 259
- null collation 268
- NULL functions 252
- NULL keyword 260
- null test operators 179
- null values
 - assigning to parameters 109
 - concatenating strings and 260
 - defining output columns and 75
 - filtering 95
 - mapping 207, 252, 253
 - testing for 179
- null_sensitive collation 268
- NullFuncMapper element 252
- numbers
 - assigning to parameters 109, 116
 - calculating 176
 - comparing 96, 175
 - entering in SQL queries 163, 172
 - ordering 266
 - rounding 178
 - setting decimal precision for 230
 - setting default values for 109
- NUMERIC data type 229, 230
- numeric data types 173, 175, 205
- numeric functions 177, 244
- NumericFuncMapper element 239, 244
- NVARCHAR data type 206, 230
- NVARCHAR2 data type 207

O

- ODA connection types 44
- ODA data source query builder 61
- ODA data source types 15
- ODA data sources
 - connecting to 15
 - creating queries for 61, 62
 - enabling passthrough security and 40
 - externalizing connections for 44
 - mapping result sets for 60–64
 - retrieving parameters from 114
- ODA drivers 15
- ODA nodes (Query Profiler) 123
- ODA operator 127
- ODBC connection strings 202, 218
- ODBC databases 12, 234
- ODBC drivers 12, 212, 234
- ODBC escape sequences 212, 230, 234
- odbc.ini 24
- online documentation ix
- open data sources. *See* ODA data sources
- open database connectivity. *See* ODBC
- Open Editor command 153
- opening
 - Expression Builder 70
 - graphical information object editor 67
 - information objects 69
 - map files 69
 - SQL editor 117
- operands 238, 258
- OperandTypes attribute 238, 239
- operator mapping examples 259–260
- operators
 - customizing 241, 242, 243, 246, 255
 - filter conditions 91, 92, 94, 95, 99
 - join conditions 84
 - mapping 236, 237, 238, 258
 - ODBC data sources 238
 - SQL queries 124, 159, 175
- optimizing
 - joins 86, 88
 - queries 86, 88, 171, 189, 195
- OPTION clause 171
 - See also* SQL statements
- Optional attribute 224
- optional data filters 75

- OPTIONAL keyword
 - aggregate functions and 193
 - computed fields and 190
 - joins and 86, 189, 190, 191
- optional parameters 224
- Optional value 75
- OR keyword 117
- OR operator 99, 179, 241, 258
- Oracle data types 208
- Oracle databases
 - connecting to 20, 204
 - creating queries for 203, 207
 - externalizing connections for 41
 - mapping to 51, 207
- ORDER BY clause 51, 119, 161, 165, 263
 - See also* SQL statements
- OrderByClause declaration (SQL) 165
- OrderByClauseMapper element 263
- OS/400 operating systems 18
- OSGi extensions framework 28
- outer joins 84, 85, 161
- output 6, 76, 120, 121
 - See also* result sets
- output columns
 - categorizing 71–72
 - changing order of 48, 71, 72
 - changing properties for 80
 - defining 69–71
 - deleting 71
 - displaying categories 72
 - displaying data types for 119
 - displaying multi-line text in 76
 - excluding from map files 47
 - filtering 47, 71, 75
 - formatting values in 75
 - grouping data and 105
 - inheriting property values and 79, 81
 - naming 70
 - renaming 47, 53, 57, 62
 - setting display lengths for 75
 - setting properties for 48, 71, 74, 120
 - truncating text in 76
 - viewing 47, 53, 62, 119
- Output Columns page 47, 57, 62
- output formats 75
- output parameters 54, 56, 59, 60, 114
 - See also* stored procedures

- overloaded functions 238
- overloaded stored procedures 55
- overriding default values 11

P

- Page Size list box 121
- Parameter Mode property 114
- parameter passing conventions 160
- Parameter Values dialog box 121, 122, 128
- ParameterDeclaration declaration (SQL) 165
- parameterized queries
 - See also* stored procedures
 - creating 159, 173
 - defining parameters for 108, 159
 - joining tables with 161
 - mapping to 51, 54, 60
 - renaming parameters for 53, 63
- parameters
 - adding to joins 124
 - assigning data types to 109, 113
 - assigning values to 111, 113, 116, 121, 170
 - changing 151
 - defining connection 223
 - defining local 115
 - defining source 115–116
 - deleting 11, 110
 - displaying 120
 - filtering data and 82, 93, 101
 - hiding 114
 - localizing information objects and 130, 132, 134, 135
 - mapping stored procedures and 54, 56, 59
 - mapping to ODA data sources and 60, 63
 - mapping to SQL databases and 209
 - mapping to SQL queries and 51, 53
 - naming 109, 114
 - prompting for 109, 110, 111
 - propagating values for 10, 138
 - querying information objects and 108, 109, 159, 173
 - renaming 10, 53, 59, 63
 - resetting properties for 115
 - setting default values for 109, 113
 - setting properties for 109, 113
 - specifying as required 114
 - synchronizing 116

- viewing data types for 120
- Parameters For Stored Procedure dialog box 55, 56
- Parameters page
 - defining parameters and 109, 115
 - defining prompts and 111
 - deleting parameters and 110
 - updating information on 116
- Parameters page (SQL editor) 53, 59, 63, 120
- ParamPlaceholder declaration (SQL) 165
- parentheses () characters 100, 191, 258
- passthrough security 39
- Passthrough value 13
- Password property
 - ActuateOne for e.Reports data source connections 17
 - database connections 18, 19, 20, 21, 22, 23
 - server profiles 139
- passwords
 - database connections and 13, 18, 20, 23
 - MySQL Enterprise login accounts and 19
 - PostgreSQL login accounts and 21
 - run-time connections and 40
 - saving 139
 - server profiles and 139
 - SQL Server login accounts and 22
- paths. *See* directory paths
- pattern characters 181
- pattern-matching operator 159
- percent (%) character 96, 181, 222, 247
- performance 17, 90, 203, 237, 267
- period (.) character 96, 109
- permissions 141
- perspectives 5
- pick lists. *See* drop-down lists
- pipe-separated values files 19
- Plain Old Java Objects. *See* POJO data sources
- platform-independent queries 234
- plugin.xml 44
- POJO data sources 15, 21
- Port number property 139
- Port property
 - DB2 data connections 18
 - Informix data connections 19
 - MySQL Enterprise connections 19
 - ODBC data connections 13
 - Oracle data connections 20
 - PostgreSQL connections 21
 - SQL Server connections 22
 - Sybase data connections 23
- ports 139
- POSITION function 183, 236, 241, 246, 270
- PostgreSQL databases 21, 28, 41, 204
- PostgreSQL login accounts 21
- PostgreSQL named instances 21
- POWER function 178, 205, 239, 244
- power function 211
- Pragma declaration (SQL) 164, 165
- pragmas 161, 195
- precedence 258
- precision 172, 173, 230
- Precision declaration (SQL) 165
- preconfigured connection types 17, 41
- preconfigured database types 12, 202, 204, 238
- preconfigured ODA data source types 15
- predefined filters 75, 81
- Predefined value 75, 81
- previewing output 6, 120, 121
- primary keys 77
- PrimaryExpression declaration (SQL) 166
- privileges 141
- Problems view 6, 138
- production databases 28
- Profile name property 139
- profiles (iHub server) 6, 139
- programmers 4
- Project command 10
- project dependencies 148, 150, 154
- project model diagrams 150, 152, 153
- Project operator 126
- projects
 - changing locations of 10
 - compiling resources for 138
 - creating 10
 - downloading files for 144, 145, 153
 - moving 39
 - naming 10
 - propagating property values for 138
 - publishing 138
 - searching resources in 148
 - selecting translation locale for 132
 - testing 138
 - viewing contents 5

- Prompt editor 80, 81, 82, 110, 111
- Prompt editor button 111
- prompt properties 109, 110, 111
- prompting for input 81, 82, 102, 111
- Propagate Property Values command 138
- propagating property values 10, 138
- properties
 - connection types 17
 - data source connections 11, 13, 17, 40
 - displaying 6
 - externalized connections 39–44
 - iHub profiles 139
 - inheriting 79–81, 115, 138
 - joins 88, 89
 - ODA data sets 64
 - ODA data sources 15, 16, 44
 - ODBC databases 12
 - output columns 48, 71, 74, 80, 120
 - parameters 109, 111, 113
 - propagating values for 10, 138
 - resetting 80, 115
 - translated strings 130, 134
- properties files (locale-specific) 130, 131
- Properties view 6
- Proxy value 13
- PSV files 19
- Publish Files button 144
- Publish Information Objects command 143
- Publish Information Objects dialog box 143
- publish locations 143
- Publish to Server command 141
- publishing
 - information objects 140, 141, 142, 143
 - localization properties files 131
 - projects 138
- PushComplexExprs attribute 263, 264

Q

- queries
 - See also* SQL statements; subqueries
 - adding comments to 169, 258
 - adding parameters to. *See* parameterized queries
 - aggregating data and 103, 193
 - building expressions for 67, 173, 175

- building for
 - information objects 67, 68, 89, 117, 158
 - ODA data sources 61, 62
 - Oracle databases 207
 - remote data sources 175, 176
 - SQL Server databases 209
 - Sybase databases 211
- building function templates for 239
- comparing strings and 174, 207
- configuring database types and 213
- converting functions for 234, 235, 238
- creating 5, 67, 117, 158, 234
- customizing mappings for 262–264
- disabling cost-based optimization for 195, 196
- filtering data with 81, 83, 90, 107
- grouping data with 102–107, 162
- initializing 260
- limitations for 158, 162, 171
- mapping data types for 204, 228, 229, 231
- mapping literal strings for 262
- mapping to 17, 51–54, 60
- naming 61
- not returning values 108
- optimizing 86, 88, 171, 189, 195, 237
- ordering strings for 266, 267, 268, 269
- prompting for input and 110, 111, 114
- retrieving data with 70, 81
- running 83, 203, 230, 235
- saving 117
- sorting data with 161
- specifying database collation for 266, 269
- updating map files and 49
- viewing execution plan for 6
- viewing information about 122, 127
- viewing parameters in 120
- query definition view 5
- query execution plans 123, 127, 128, 129, 130
- query extensions 159
- query operators. *See* operators
- Query Profiler view 6, 122, 123, 127
- QueryParameterDeclaration declaration (SQL) 166
- question mark (?) character 51, 247
- " character code 214

R

- radio buttons 75
- range of values 94, 175
- range test operator 175
- REAL data type 229
- records. *See* rows
- references
 - aliases and 162
 - database views and 159
 - information objects and 82, 169
 - pattern-matching operator and 159
 - SQL queries and 159
 - table names and 159
- Refine Database dialog box 149
- Regenerate Diagram command 152
- relational databases 234
 - See also* databases
- RelationalOperator declaration (SQL) 166
- relative paths 82, 119, 169
- remainders 177
- Remember Password option 139
- remote data sources 175, 176
- removing. *See* deleting
- renaming
 - column categories 72
 - columns 10, 47, 48
 - information objects 79
 - maps 79
 - parameters 10, 53, 59, 63
 - stored procedures 55
- report design files 139, 145
- report designs
 - creating filters for 75, 81, 82
 - creating queries and 169
 - downloading 144, 145
 - externalizing connections and 40
 - file dependencies and 148, 150, 153
 - localizing information objects and 130
 - publishing 139
 - specifying control type for 75
- Report document path property 18
- report documents 15, 18
- report files 144, 148, 150
- report libraries 151
- reports 40, 76, 130
- repository. *See* Encyclopedia volumes
- Required property 76, 114
- reserved words (Actuate SQL) 168
- resource names 4
- resources 138, 140
- Result Set Name dialog box 57
- result sets
 - categorizing columns in 71–72
 - changing column order in 71, 72
 - changing output columns names 53, 57, 62
 - defining output columns for 69–71
 - displaying multi-line text in 76
 - filtering data for 47, 71, 75
 - joins and 90, 160
 - mapping to 17, 54, 55, 60, 61
 - naming 57
 - removing output columns in 71
 - returning distinct values for 70
 - truncating text in 76
 - viewing output columns in 47, 53, 62
- RIGHT function 181, 246, 270
- RIGHT OPTIONAL keywords 192, 193
- ROUND function 178, 205, 244
- rows
 - adding to joins 84
 - copying iterators for 125
 - excluding duplicate 70, 103
 - filtering blank values in 95
 - filtering null values in 95
 - grouping 103, 125
 - previewing output and 121
 - resetting iterators for 125
 - restricting number returned 90, 99, 101, 107
 - retrieving from queries 124, 125, 126
 - specifying threshold values for 197, 198
 - viewing blank values in 95
 - viewing null values in 95
- RPAD function 51
- RTRIM function 182, 245
- running queries 83, 203, 230, 235
- run-time connections 12, 15, 40
- Runtime element 40
- run-time queries 159
- run-time values 83, 110, 113

S

- saving
 - connection definition files 11
 - passwords 139
 - queries 117
 - query execution plans 127, 128, 129
 - resources 138
- scalar subqueries 160
- scalar values 160, 173, 176
- ScalarDataType declaration (SQL) 166
- Scale declaration (SQL) 166
- Schema (optional) property 28
- Schema name prefix option 46, 56, 149
- schemas 24, 28, 46, 56, 149
- scripts (impact reports) 153
- search function 183
- search operations 148, 150
- Search view 150
- security 13, 39, 139
- SELECT clause 103, 105
 - See also* SELECT statements
- Select operator 126
- SELECT statements 159, 161, 162
 - See also* SQL statements
- selection lists. *See* drop-down lists
- SelectItem declaration (SQL) 166
- SelectList declaration (SQL) 166
- SelectStatement declaration (SQL) 166
- SelectWithoutFrom declaration (SQL) 166
- SelectWithoutOrder declaration (SQL) 166
- semicolon-separated values files 19
- separators (numeric values) 96, 109
- serial values 186, 252
- Server Explorer view 6
- Server name property 20, 21, 22, 23
- server ports 13, 18
- server profiles (iHub) 6, 139
- Server property 18, 139
- Server URL property 17
- servers 18, 19, 20, 23
 - See also* iHub servers
- Service property 19
- set difference operations 158
- SetClause declaration (SQL) 166
- Shared Resources folder 140
- Show Categories icon 6
- Show categories in graphical editor option 73
- Show Impact command 152
- Show map properties button 90
- SID property (Oracle System Identifier) 20
- SignedLiteral declaration (SQL) 166
- SimpleCondition declaration (SQL) 167
- SINGLE EXEC keywords 171
- single quotation mark (') character 95, 109, 163
- SingleMatchChar attribute 246, 247
- Size property 114
- slash characters. *See* backward slash character; forward slash character
- .sma files 45
 - See also* map files
- SMALLINT data type 229
- SOAP end point property 23
- sort keys 126
- sort operations 266
- Sort operator 126
- SortedOuterUnion operator 127
- source columns 49
- source parameters 54, 60, 115–116
- space characters. *See* white space characters
- special characters. *See* characters
- SQL compiler 86, 121
- SQL conventions (Actuate) 162, 164
- SQL data types
 - casting rules for 173
 - DB2 databases and 205
 - Informix databases and 206
 - listed 172, 228
 - mapping to 204, 228, 229, 231
 - Oracle databases and 207
 - SQL databases and 209
 - Sybase databases and 211
- SQL editor 117, 118, 120
- SQL editor icon 117
- SQL expressions 67, 173, 175
- SQL functions
 - aggregation 187, 193, 254
 - compatibility with 236
 - converting 234, 235
 - custom mappings for 238, 244, 245, 256
 - DB2 databases and 205
 - disabling default mappings for 236, 241
 - mapping 234, 236, 237, 239

- null values and 252
 - numeric data types and 177
 - Oracle databases and 207
 - SQL databases and 209
 - string data types and 179
 - substrings and 181, 183
 - Sybase databases and 211
 - timestamp values and 184
 - volume user and 188
- SQL identifiers 163, 168, 169
- SQL keywords 168
- SQL nodes (Query Profiler) 123
- SQL operator 127
- SQL operators 175
 - See also* operators
- SQL operators (mapping examples) 259–260
- SQL parameters 108–110
- SQL Preview view 6, 117, 158
- SQL Server databases
 - connecting to 22, 204
 - creating queries for 209
 - externalizing connections for 41
 - mapping to 209
- SQL Server login accounts 22
- SQL Server named instances 22
- SQL statements
 - See also* queries
 - adding
 - column names to 169
 - escape sequences to 234
 - functions to 68, 159
 - parameters to 109, 159, 173
 - table names to 169
 - case conversions and 180, 206
 - displaying 6, 122
 - editing 117, 118
 - entering at run time 159
 - mapping GROUP BY clause for 263, 264
 - mapping ORDER BY clause for 263
 - referencing
 - aliases in 162
 - information objects in 169
 - parameters in 159
 - tables or views in 159
 - removing columns from 105–107
 - selecting columns for 70, 169
 - selecting tables for 86, 159, 172, 189
 - setting filter conditions in 90, 93, 97
 - setting join conditions in 84, 89, 160, 195
 - unassigned parameters and 108
- SQL text editor. *See* SQL editor
- SQL type codes 212
- SQL-92 keywords 168, 169
- SQL-92 specifications 236
- square brackets ([]) characters. *See* brackets
- characters
- SSV files 19
- Start Auto suggest option 83, 113
- static data sources 15, 22
- Stored procedure name prefix option 56
- stored procedures
 - changing parameter names for 59
 - filtering 56
 - mapping to 54–60, 228
 - renaming 55
 - retrieving list of 219
 - returning parameter type for 114
- string data types 172
- string functions 179, 181, 182, 183, 245
- string operators 179
- strings
 - assigning to parameters 109, 116, 173
 - casting rules for 174
 - comparing 96, 174, 175, 207
 - concatenating 163, 180, 207, 260
 - converting case 180, 206
 - entering in SQL statements 172
 - filtering blank values in 95
 - filtering data and 81
 - filtering tables and views and 46
 - getting length of 180
 - localizing information objects and 130, 133, 134
 - mapping literal 262
 - mapping to data types and 230
 - mapping to queries and 51
 - masking 224
 - matching characters in 96, 181, 247, 248
 - maximum size for 230
 - missing values in 95
 - ordering 266, 269
 - padding 206
 - parsing 181
 - returning substrings in 181, 183

- strings (*continued*)
 - searching tables and views and 149
 - setting maximum length for 172
 - trimming white space in 182
 - XML parsers and 215
- SUB operator 243
- subcategories (output columns) 71, 72
- subqueries
 - creating 159, 171–172
 - grouping data and 162
 - matching character patterns and 159
 - optimizing 171
 - returning scalar values and 160
- SubQuery declaration (SQL) 167
- subroutines 124
- SUBSTR function 237
- SUBSTRING function 182, 237, 246, 270
- substring functions 181, 245
- SubStringFuncMapper element 245
- substrings 181, 183, 245
- subtraction operator 176, 243, 260
- SUM function 187, 188, 193, 255
- summary tables 74, 76, 77
- summary values 187
 - See also* aggregation
- SWITCH statements 254
- Sybase data types 211
- Sybase databases
 - connecting to 22, 204
 - creating queries for 211
 - externalizing connections for 41
 - mapping to 55, 210
 - setting character encodings for 24
- Sybase named servers 23
- symbolic operators 258
- synchronizing source parameters 116
- syntax (Actuate SQL) 162, 164
- synthesized data 126

T

- tab characters 168
- tab-delimited formats 126
- table aliases 118
- table names 169
- Table/View name prefix option 46, 149
- TableParameter declaration (SQL) 167

- TableParameters declaration (SQL) 167
- tables
 - See also* summary tables
 - adding to queries 86, 159, 172
 - changing 48
 - filtering 46
 - getting 219
 - joining 161
 - mapping to database 45–48, 61, 203, 228
 - searching 149
 - specifying as optional 86, 189–193
- tab-separated values files 19
- templates (SQL functions) 234, 236, 237, 239, 257
- testing
 - connections 14, 17
 - for null values 179
 - for range of values 175
 - for set of values 176
 - information objects 126, 134
 - joins 123
 - output 121
 - projects 138
- text 76, 163, 181
- text boxes 75
- text editors 153
- text file data sources 126
- Text Format property 76
- text strings. *See* strings
- textual information object editor. *See* SQL editor
- textual queries
 - building expressions for 67
 - creating 67, 117–119
 - displaying output columns for 119
 - displaying parameters for 120
 - saving 117
- textual query editor 53, 67
 - See also* SQL editor
- THEN keyword 254
- ThenString attribute 254
- threshold values (SQL indexes) 197, 198
- time stamps
 - as Actuate SQL token 164
 - as literal strings 262
 - assigning to parameters 116
 - comparing 175

- delta values and 185
- mapping 248, 249, 251, 252
- returning current 184
- returning current date for 185
- scalar values and 173
- specifying default 109
- time values 95, 184
- TIMESTAMP data type 172, 173, 229
- timestamp functions 184
- TIMESTAMP keyword 262
- TIMESTAMP_STRING token (SQL) 164
- TINYINT data type 229
- TNS names file property 20
- TNS server name property 20
- Toggle categories view icon 73
- tokens (SQL grammar) 163
- totals 107, 188
- trailing spaces 96
- Transact-SQL data types 209
- Transact-SQL functions 209
- translation keys 130, 133, 134, 135
- translation strings 130, 133, 134
- translations 130
- TRIM function 182
- true values. *See* Boolean values
- truncation 76, 173, 230
- trusted connections 17
- TSV files 19
- Type attribute 40, 224
- type casting 159, 173, 256
- type declarations 164, 166
 - See also* data types

U

- UnaryExpression declaration (SQL) 167
- UnaryLogicalExpression declaration (SQL) 167
- unassigned parameters 108
- UNC paths 18
- underscore (_) character 96, 181, 247
- Unicode binary collation 209, 266, 267, 268
- Unicode code point order 266
- Unicode data 22
- Unicode strings 163, 172, 262
- Unicode_BIN collation 266
 - See also* Unicode binary collation

- unicode_bin collation 268
 - See also* Unicode binary collation
- UNION keyword 117
- Union operator 126
- UNION statements 158, 166
- unique values 70, 88
 - See also* DISTINCT keyword
- UNIX systems. *See* Linux servers
- unknown data types 75
- unnamed parameters 51, 159
- UnsignedLiteral declaration (SQL) 167
- Update Map command 49
- Update Maps command 51
- updating
 - column names 10, 48
 - maps 48, 49, 51
- UPPER function 180, 245
- URL of the XML schema property 24
- URL of the XML source property 23
- URLs
 - localized information objects and 131
 - WSDL files 23
 - XML data streams 23
- Use Precompiled Query Plan at runtime property 128
- Use Trusted Connection property 17
- UseCharStringImplByDefault attribute 270
- User name property
 - DB2 databases 18
 - iHub profiles 139
 - Informix databases 19
 - MySQL Enterprise databases 20
 - Oracle databases 21
 - PostgreSQL databases 21
 - SQL Server databases 22
 - Sybase databases 23
- user names
 - database connections and 13, 18, 19, 23
 - MySQL Enterprise login accounts and 20
 - PostgreSQL login accounts and 21
 - returning 188
 - server profiles and 139
 - SQL Server login accounts and 22
- User property 17
- UseSelectItemIndexes attribute 263, 264

V

ValueExpression declaration (SQL) 167

ValueIsCaseSensitive attribute 224

values

See also data

aggregating. *See* aggregation

assigning to parameters 109, 111, 116, 121, 170

comparing. *See* comparisons

counting non-null 187

creating list of 81, 82, 110, 111

filtering empty or blank 95

filtering on multiple 94

filtering range of 94

hiding 113

inheritance and blank 81

mapping data types and 207, 211, 230

returning largest 177

returning smallest 177

rounding 178

selecting at run time 83, 110, 113

setting default 109, 113

testing for null 179

testing range of 175

testing sets of 176

ValueSelectItem declaration (SQL) 167

ValueSelectList declaration (SQL) 167

VARCHAR data type

as generic type 229

as literal strings 262

assigning to parameters 113, 114

casting numeric types to 205

declaring 230

defining output columns and 76

SQL queries and 172, 173

VARCHAR2 data type 207

variables 108

variant strings 173, 262

variant types 207

See also NVARCHAR; VARCHAR data types

View Error Log command 39

viewing

column categories 72

column names 75

data types 49, 77, 119, 120

error messages 6, 138

hidden messages 7

impact reports 153

information objects 68

map files 68

output columns 47, 53, 62, 119

parameters 120

project model diagrams 152

SQL queries 6, 122, 127

views

changing 48

filtering 46

mapping to database 45–48, 203, 228

searching 149

virtual tables 172

Volume property 17, 139

volumes. *See* Encyclopedia volumes

W

web services data sources 15, 23

Web Services Description Language file 23

WHEN clause 167

WHEN keyword 254

WhenClause declaration (SQL) 167

WhenString attribute 254

WHERE clause 89, 90, 167, 211

See also SQL statements

WhereClause declaration (SQL) 167

white space characters

in queries 164, 168, 169

in strings 182

wildcard characters 96, 247

Windows systems 18, 27

WITH clause 82, 108, 159, 166

See also SQL statements

Word Wrap property 76

WSDL descriptor property 23

X

XML connection types 44

XML data source connections 23, 24, 44

XML data sources 15

XML data streams 23

XML elements 215

XML files 24, 214

See also configuration files

XML parsers 215
XML schemas 24

Z

z/OS operating systems 18
zero-length variants 207

