DataDirect
TECHNOLOGIES

DataDirect Connect®

for SQL/XML

User's Guide

May 2003

DataDirect Technologies
3202 Tower Oaks Blvd. Suite 300
Rockville, Maryland 20852

Table of Contents

Preface

This book is your guide to using the DataDirect Connect® *for* SQL/XML product from DataDirect Technologies. Read on to find out more about DataDirect Connect *for* SQL/XML and how to use this book.

What Is DataDirect Connect *for* SQL/XML?

DataDirect Connect *for* SQL/XML is your solution for producing XML for data exchange. Developers need a way to build hierarchical XML structures using queries on a set of relational tables. Similarly, they need a way to update the content of their relational tables using data in XML hierarchies. Using Connect *for* SQL/XML, you can accomplish both of these tasks, and you can accomplish them by writing one Java application that works without change for any database DataDirect Technologies JDBC drivers support.

Connect *for* SQL/XML lets Java applications connect to databases using DataDirect Technologies JDBC drivers, return XML values in the columns of JDBC result sets, and access XML columns as JDOM, SAX, DOM, or text. See Chapter 1, "Connect for SQL/XML Overview" on page 19 for a more detailed explanation of Connect *for* SQL/XML.

Using This Book

This book assumes that you are familiar with your operating system and its commands; the concept of directories; the management of user accounts and security access; and your network protocol and its configuration. You should also be familiar with Java, the JDBC API, and SQL.

This book contains the following information:

- Chapter 1, "Connect for SQL/XML Overview" on page 19 provides an overview of the DataDirect Connect *for* SQL/XML product.
- Chapter 2, "Understanding Connect for SQL/XML" on page 39 describes some key features of Connect *for* SQL/XML and provides examples to help you decide how to use them.
- Chapter 3, "Creating Connect for SQL/XML Queries Using the Builder" on page 61 explains how to create SQL/XML and jXTransformer queries using the DataDirect Query Builder *for* SQL/XML (the Builder).
- Chapter 4, "Syntax of SQL/XML Queries" on page 141 describes the syntax of SQL/XML queries.
- Chapter 5, "Syntax of jXTransformer Queries" on page 153 describes the syntax of jXTransformer queries.
- Chapter 6, "Syntax of jXTransformer Write Statements" on page 173 describes the syntax of jXTransformer Insert, Update, and Delete statements.
- Chapter 7, "Using the SQL/XML JDBC Driver and JDBC API Extensions" on page 191 describes the SQL/XML JDBC driver and the classes it uses to process SQL/XML queries. It also provides information about connecting to the database and using SQL/XML queries in Java applications.

- Chapter 8, "Using the jXTransformer API" on page 207 describes the jXTransformer API and the classes it uses to process jXTransformer queries and write statements. It also provides information about connecting to the database and using jXTransformer queries and write statements in Java applications.
- Chapter 9, "Tutorial: Using SQL/XML Queries" on page 233 contains a step-by-step tutorial that show you how to create a SQL/XML query using the DataDirect Query Builder *for* SQL/XML. It also shows you how to embed that query in a Java application and use Connect *for* SQL/XML hints to specify processing options that are not supported through the standard SQL/XML query syntax.
- Chapter 10, "Tutorial: Using jXTransformer Queries" on page 253 contains a step-by-step tutorial that shows you how to create a jXTransformer query using the DataDirect Query Builder *for* SQL/XML. It also shows you how to embed that query in a Java application using the jXTransformer API.
- Appendix A "JDBC Data Types" on page 295 lists the JDBC data types supported by Connect *for* SQL/XML and the XML representations to which they are converted when building XML values.
- Appendix B "jXTransformer Query and Statement Processing" on page 297 provides additional information that you need to know about how Connect *for* SQL/XML processes jXTransformer queries and write statements.

In addition, this book contains a "Glossary" on page 299 that defines terms used in this book.

NOTE: This book refers the reader to Web URLs for more information about specific topics, including Web URLs not maintained by DataDirect Technologies. Because it is the nature of Web content to change frequently, DataDirect Technologies can guarantee only that the URLs referenced in this book were correct at the time of publishing.

Conventions Used in This Book

This section describes the typography, terminology, and other conventions used in this guide.

Typographical Conventions

This guide uses the following typographical conventions:

Convention	Explanation
italics	Introduces new terms that you may not be familiar with, and is used occasionally for emphasis.
bold	Emphasizes important information. Also indicates button, menu, and icon names on which you can act. For example, click **Next**.
UPPERCASE	Indicates the name of a file. For operating environments that use case-sensitive filenames, the correct capitalization is used in information specific to those environments. Also indicates keys or key combinations that you can use. For example, press the ENTER key.
`monospace`	Indicates syntax examples, values that you specify, or results that you receive.
`monospaced italics`	Indicates names that are placeholders for values you specify; for example, *`filename`*.
forward slash /	Separates menus and their associated commands. For example, Select File / Copy means to select Copy from the File menu.
vertical rule |	Indicates an OR separator to delineate items.
brackets [ ]	Indicates optional items. For example, in the following statement: SELECT [DISTINCT], DISTINCT is an optional keyword.

Convention	Explanation
braces { }	Indicates that you must select one item. For example, {yes | no} means you must specify either yes or no.
ellipsis . . .	Indicates that the immediately preceding item can be repeated any number of times in succession. An ellipsis following a closing bracket indicates that all information in that unit can be repeated.

About DataDirect Documentation

The DataDirect Connect *for* SQL/XML library consists of the following books:

- *DataDirect Connect for SQL/XML Installation Guide* details requirements and procedures for installing Connect *for* SQL/XML.
- *DataDirect Connect for SQL/XML User's Guide* provides detailed information about using DataDirect Connect *for* SQL/XML both to retrieve relational data and transform it into XML structures, and to transform XML data into relational data.
- *DataDirect Connect for JDBC User's Guide and Reference* provides detailed information about using DataDirect Connect *for* JDBC drivers to connect to the database.

For detailed information about DataDirect SequeLink *for* JDBC, refer to the Connect *for* SQL/XML product documentation on the DataDirect Technologies web site:

http://www.datadirect-technologies.com/download/docs/dochome.asp

NOTE: Depending on your configuration, you may have to configure the Web browser as described in "Changing the GUI's General Appearance" on page 74 before you can use the DataDirect Query Builder *for* SQL/XML help or the Web links in the Help menu of the Builder.

Contacting Technical Support

DataDirect Technologies provides technical support for registered users of this product, including limited installation support, for the first 30 days. Register online for your SupportLink user ID and password for access to the password-protected areas of the SupportLink web site at http://www.datadirect-technologies.com/support/support_index.asp. Your user ID and password are issued to you by email upon registration.

For post-installation support, contact us using one of the methods listed below or purchase further support by enrolling in the SupportLink program. For more information about SupportLink, contact your sales representative.

The DataDirect Technologies web site provides the latest support information through SupportLink Online, our global service network providing access to support contact details, tools, and valuable information. Our SupportLink users access information using the web and automatic email notification. SupportLink Online includes a knowledge base so you can search on keywords for technical bulletins and other information.

World Wide Web

http://www.datadirect-technologies.com/support/support_index.asp

E-Mail

USA, Canada, and Mexico	supportlink@datadirect-technologies.com
Europe, Middle East, and Africa	int.supportlink@datadirect-technologies.com
Japan	jpn.answerline@datadirect.co.jp
All other countries	http://www.datadirect-technologies.com/contactus/distributor.asp provides a list of the correct e-mail contacts.

Local Telephone Support

Local phone numbers can be found at:

http://www.datadirect-technologies.com/support/support_contact_aline.asp

SupportLink support is available 24 hours a day, seven days a week.

Fax Information

Fax US, Mexico, and Canada	1 919 461 4527
Fax EMEA	+32 (0) 15 32 09 19

When you contact us, please provide the following information:

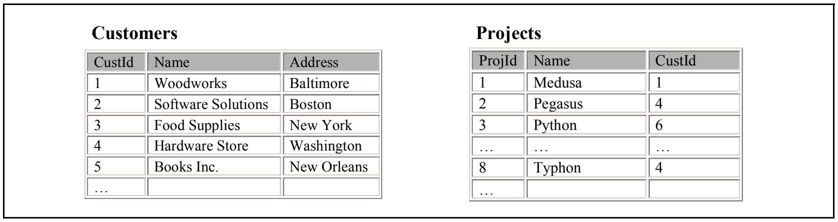
- The **product serial number** or a case number. If you do not have a SupportLink contract, we will ask you to speak with a sales representative.
- Your **name and organization**. For a first-time call, you may be asked for full customer information, including location and contact details.
- The **version number** of your DataDirect product.
- The type and version of your **operating system**.
- Any **third-party software or other environment information** required to understand the problem.

- A **brief description of the problem,** including any error messages you have received, **and the steps preceding the occurrence of the problem**. Depending on the complexity of the problem, you may be asked to submit an example so that we can recreate the problem.
- An assessment of the **severity level** of the problem.

1 Connect *for* SQL/XML Overview

Many Java applications exchange data as XML, but store and query data using a JDBC connection to a traditional relational database. Unfortunately, XML and SQL represent information in very different ways, and many developers spend significant effort converting information between the two. The following example shows the same information represented in relational tables and represented in XML. In the XML representation, hierarchy and sequence are the main ways used to represent relationships among data; in the relational representation, data is represented as unordered two-dimensional tables.

Relational Tables



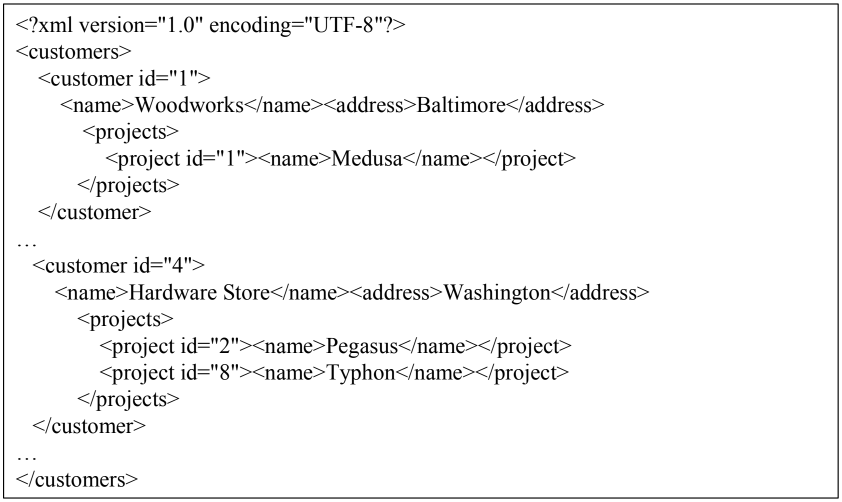
Customers

CustId	Name	Address
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Store	Washington
5	Books Inc.	New Orleans
…		

Projects

ProjId	Name	CustId
1	Medusa	1
2	Pegasus	4
3	Python	6
…	…	…
8	Typhon	4
…		

XML



```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
   <customer id="1">
      <name>Woodworks</name><address>Baltimore</address>
         <projects>
            <project id="1"><name>Medusa</name></project>
         </projects>
   </customer>
…
  <customer id="4">
     <name>Hardware Store</name><address>Washington</address>
        <projects>
           <project id="2"><name>Pegasus</name></project>
           <project id="8"><name>Typhon</name></project>
        </projects>
  </customer>
…
</customers>
```

When producing XML for data exchange, developers need a way to build hierarchical XML structures using queries on a set of unordered two-dimensional tables. Similarly, they need a way to update the content of their two-dimensional tables using data in XML hierarchies.

Almost all major RDBMS vendors now supply tools or product enhancements to help bridge the gap between XML and relational data. Unfortunately, their proprietary approaches are incompatible and are often ad hoc or awkward. A better, standards-based approach now exists. SQL/XML is an extension to the ANSI/ISO SQL standard that allows XML to be generated as the result of a query, and adds an XML data type to SQL so that XML query results can be returned in columns of normal SQL result sets. The final draft of SQL that includes these extensions is expected in mid-2003. Using standard SQL/XML instead of proprietary vendor extensions simplifies development, provides for more maintainable code, and provides portability across databases.

For most Java applications that need to exchange data in XML, the most efficient approach is to use SQL/XML to query relational data and build the appropriate XML structure. For example, suppose your company uses data stored in an Oracle database for billing purposes. You may want to make available some of the same data over your company's Intranet for project planning purposes. Using Connect *for* SQL/XML, you can create Connect *for* SQL/XML queries to structure that information any way you want it in the returned result set or XML. You can create one Connect *for* SQL/XML query that lists each project by employee and their billable hours, and you can create another Connect *for* SQL/XML query that lists each employee by their project assignment and billable hours.

DataDirect Connect *for* SQL/XML lets Java applications connect to databases using DataDirect Technologies JDBC drivers, return XML values in the columns of JDBC result sets, and access XML columns as JDOM, SAX, DOM, or text. This makes it easy to write applications that work without change for any database DataDirect Technologies JDBC drivers support.

Currently, the SQL/XML standard supports queries, but not updates. DataDirect Technologies is a member of the H2.3 Task Group that is responsible for SQL/XML. We hope that updates will be added to the standard; in the meantime, Connect *for* SQL/XML provides proprietary extensions to the SQL99 Insert, Update, and Delete statements that allow you to update data stored in relational databases with information extracted from XML.

DataDirect Connect *for* SQL/XML was originally introduced as DataDirect jXTransformer, which used a query language derived from the same paper that inspired SQL/XML. DataDirect Technologies is committed to supporting standards, so we have renamed the product to clearly convey our support for SQL/XML. The older, proprietary language and API are still available, and are particularly useful for updates. As SQL adds this functionality to the standard, we will provide it in our SQL/XML implementation. As a member of the SQL/XML task group, DataDirect Technologies wants the standard to cover all functionality required for common applications, allowing you to build your applications using standards-based components.

Types of Connect *for* SQL/XML Queries

DataDirect Connect *for* SQL/XML supports two types of queries: *SQL/XML queries* and proprietary queries known as *jXTransformer queries*. Using either type of query, you can create XML for data exchange or another purpose.

SQL/XML queries return JDBC result sets that can contain XML values. SQL/XML queries must be executed through the Connect *for* SQL/XML JDBC driver (the SQL/XML JDBC driver). SQL/XML is an extension to the ANSI/ISO SQL standard.

jXTransformer queries return XML results in the form of an XML document, a JDBC result set, or directly to your Java application. Typically, jXTransformer queries are executed through the proprietary jXTransformer API, but they also can be executed through the SQL/XML JDBC driver.

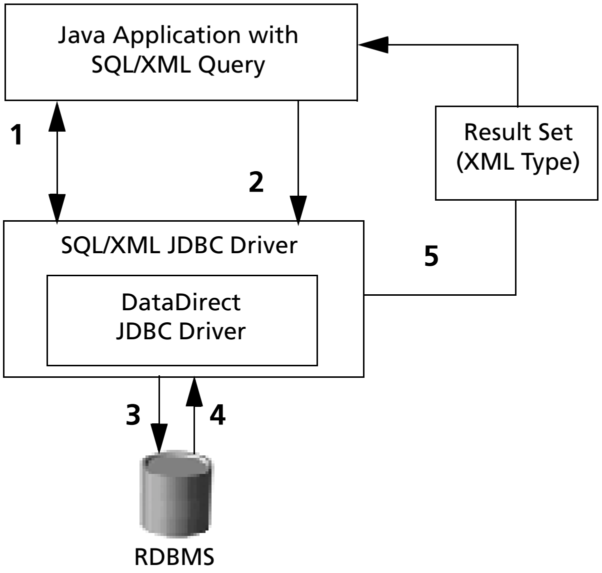
SQL/XML Queries

This section provides information about the architecture of a SQL/XML query used in a Java application and an example of a SQL/XML query.

Architecture of SQL/XML Queries in Java Applications

Figure 1-1 shows the components involved when a SQL/XML query is used in a Java application.

Figure 1-1. Architecture of a SQL/XML Query in a Java Application



1 The Java application establishes a JDBC connection with the database.

2 The Java application issues the SQL/XML query using the SQL/XML JDBC driver.

3 The SQL/XML JDBC driver processes the SQL/XML query and sends one or multiple SQL statements through to the database to retrieve the result set.

NOTE: As database vendors begin implementing the SQL/XML extensions to SQL, the SQL/XML JDBC driver may not need to process the SQL/XML query before sending it to the database. Whether the driver processes the SQL/XML query in the future will depend on performance advantages.

4 The database sends the requested result set to the SQL/XML JDBC driver.

5 The SQL/XML JDBC driver creates a JDBC result set containing the XML structure specified by the SQL/XML query and returns it to the application.

SQL/XML Query Example

SQL/XML queries contain two types of instructions for the SQL/XML JDBC driver to process. The first type of instruction is conventional SQL99, which is used to identify the data that will be retrieved. The second type includes XML constructors that are used to create the XML structures of the data the query will return.

For example, let us examine a simple SQL query:

```
SELECT
   e.EmpID,
   e.FirstName,
   e.LastName,
   e.Title,
   e.StartDate,
   e.HourlyRate
FROM Employees e WHERE e.Start-Date >= {d '2000-01-01'}
```

Now, let us compare the preceding SQL query with a SQL/XML query that uses the same Select statement and wraps XML constructors around the columns to define how to structure the JDBC result set:

```
SELECT
   XMLELEMENT (NAME "EMPLOYEES_INFO",
     XMLATTRIBUTES (e.EmpID AS "ID"),
     XMLELEMENT (NAME "name",
       XMLELEMENT (NAME "first", e.FirstName),
       XMLELEMENT (NAME "last", e.LastName)),
     XMLELEMENT (NAME "title", e.Title),
     XMLELEMENT (NAME "hiredate", e.StartDate),
     XMLELEMENT (NAME "salary", e.HourlyRate)) AS "result"
FROM Employees e WHERE e.StartDate >= {d '2000-01-01'}
```

Notice that, in this example, we used every column specified in the SQL query and wrapped XML constructors around the columns. The returned result set contains one column and two rows.

result

<Employees_Info ID='9'>
      <name>
         <first>Mike</first>
         <last>Johnson</last>
      </name>
      <title>Mr</title>
      <hiredate>2000-01-15 00:00:00.0</hiredate>
      <salary>95</salary>
</Employees_Info>

<Employees_Info ID='18'>
      <name>
         <first>Sonia</first>
         <last>Evans</last>
      </name>
      <title>Ms</title>
      <hiredate>2000-10-01 00:00:00:.0</hiredate>
       <salary>105</salary>
</Employees_Info>

To represent XML in result sets, SQL/XML introduces a data type called the XMLType and places instances of the XML type in columns. Connect *for* SQL/XML provides the com.ddtek.jdbc.jxtr.XMLType class to allow instances of the XMLType to be retrieved as DOM, JDOM, SAX, or text.

See "Syntax of SQL/XML Queries" on page 141 for more information about the syntax of SQL/XML queries.

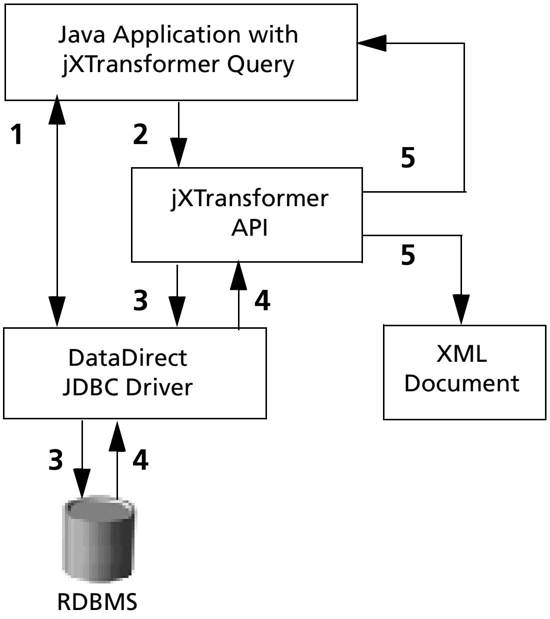
jXTransformer Queries

This section provides information about the architecture of a jXTransformer query used in a Java application and an example of a jXTransformer query.

Architecture of jXTransformer Queries in Java Applications

Figure 1-2 shows the components involved when a jXTransformer query is used in a Java application.

Figure 1-2. Architecture of a jXTransformer Query in a Java Application



1 The Java application establishes a JDBC connection with the database.

2 The Java application issues the jXTransformer query using the jXTransformer API.

3 The jXTransformer API processes the jXTransformer query and sends one or multiple SQL99 statements through the DataDirect Technologies JDBC driver to the database to retrieve the result set.

4 The DataDirect Technologies JDBC driver returns the requested result set to the jXTransformer API.

5 The jXTransformer API formats the data in the XML structure specified by the jXTransformer query and returns it to the application (for example, in a DOM or JDOM representation) or writes it to a standalone XML document file.

jXTransformer Query Example

jXTransformer queries contain two types of instructions for the jXTransformer API to process. The first type of instruction is conventional SQL99, which is used to identify the data that will be retrieved. The second type includes XML constructors that are used to create the XML structures of the data the query will return.

For example, let us examine a simple SQL query:

```
SELECT
   e.EmpID,
   e.FirstName,
   e.LastName,
   e.Title,
   e.StartDate,
   e.HourlyRate
FROM Employees e WHERE e.StartDate >= {d '2000-01-01'}
```

Now, let us compare the preceding SQL99 query with a jXTransformer query that uses the same Select statement and adds jXTransformer syntax constructs about how to structure the data that the query retrieves in XML:

```
xml_document(xml_element('result',
 SELECT
    xml_element('Employees_Info',
       xml_attribute('ID', e.EmpID),
       xml_element('name',
           xml_element('first', e.FirstName),
           xml_element('last', e.LastName) ),
        xml_element('title', e.Title),
        xml_element('hiredate', e.StartDate),
        xml_element('salary', e.HourlyRate) )
 FROM Employees e WHERE e.StartDate >= {d '2000-01-01'} ))
```

The resulting XML document for this jXTransformer query example would look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<result>
   <Employees_Info ID='9'>
      <name>
         <first>Mike</first>
         <last>Johnson</last>
      </name>
      <title>Mr</title>
      <hiredate>2000-01-15 00:00:00.)</hiredate>
      <salary>95</salary>
   </Employees_Info>
   <Employees_Info ID='18'>
      <name>
         <first>Sonia</first>
         <last>Evans</last>
      </name>
      <title>Ms</title>
      <hiredate>2000-10-0 00:00:00.0</hiredate>
       <salary>105</salary>
   </Employees_Info>
</result>
```

See “Syntax of jXTransformer Queries” on page 153 for more information about the syntax of jXTransformer queries.

SQL/XML or jXTransformer?

We recommend that you use SQL/XML whenever you can. You may find, however, that the current SQL/XML extensions to the SQL standard do not contain the functionality that you need. In these cases, we still support our older jXTransformer language because many developers need to perform updates or to create processing instructions, CDATA Sections, comments, or DOCTYPEs in the results of queries. In the future, we hope that this functionality will all be available as part of the SQL standard.

Both SQL/XML and jXTransformer queries support the following features:

- Read access to any relational database supported by DataDirect Technologies JDBC drivers
- Support for constructing XML typed values based on relational data
- Creation of XML structures based on standard JDBC result sets
- DataDirect Query Builder *for* SQL/XML GUI tool that allows you to quickly create, modify, and test Connect *for* SQL/XML queries without having to know the details of the query syntax
- JAXP 1.2, DOM level 2, JDOM 1.0 Beta 7, SAX 2.0.1, writer object interfaces, and XPath 1.0
- Parameter markers, SQL Select expressions, and JDBC scalar functions within Connect *for* SQL/XML queries
- Reuse of prepared statements, batch updates, and other performance-enhancing techniques

The following list describes the features provided by jXTransformer statements that are not provided by SQL/XML queries:

- Write access to any relational database supported by DataDirect Technologies JDBC drivers
- Native XML results (transforming into XML from result sets is not required)
- Full support for XML, including:
  - DTDs, XML schemas, and namespaces
  - CDATA sections
  - Comments in the XML document header
  - Document-level processing instructions, such as the specification of XSL stylesheets

jXTransformer Write Statements

To write data stored in XML documents to relational databases, you create jXTransformer write statements (Insert, Update, and Delete statements). jXTransformer write statements allow you to perform the following tasks:

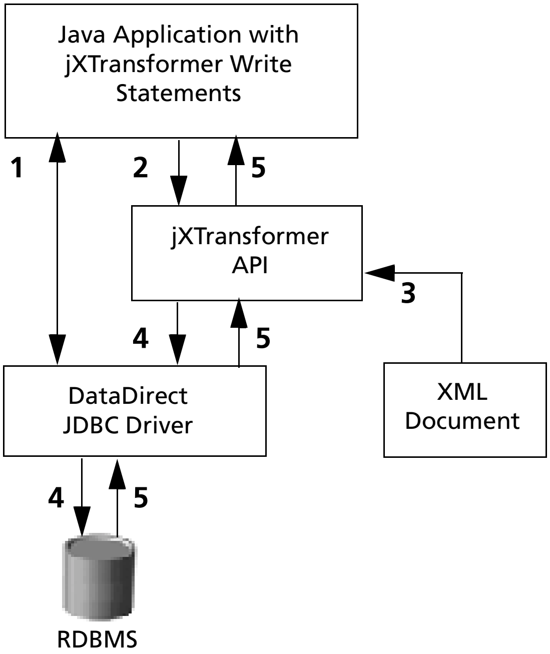
- jXTransformer Insert statements insert data from an XML document into new rows in relational database tables.
- jXTransformer Update statements update data in relational database tables with data from an XML document.
- jXTransformer Delete statements delete data in a relational database table. The rows that are deleted are specified by a Where clause in the jXTransformer Delete statement.

For example, suppose your company's Human Resources (HR) department uses a Web interface to enter and make changes to employee information. When a HR representative adds a new employee or changes an employee's existing information using the Web interface, the data is written to an XML document. Because employee data is stored in an Oracle database for payroll purposes, jXTransformer write statements are used in a Java application that runs automatically at the end of each day to change the employee data in the Oracle database.

Architecture of jXTransformer Write Statements in Java Applications

Figure 1-3 shows the components involved when jXTransformer write statements are used in a Java application.

Figure 1-3. Architecture of a jXTransformer Write Statement in a Java Application



1 The Java application establishes a JDBC connection with the database.

2 The Java application issues the jXTransformer write statement (Insert, Update, or Delete) using the jXTransformer API.

3 The jXTransformer API retrieves the data that is to be written to the database from the XML document.

4 The jXTransformer API sends one or multiple SQL99 statements through the DataDirect Technologies JDBC driver to the database and the database tables are updated.

5 The DataDirect Technologies JDBC driver returns the result of executing the jXTransformer write statements to the jXTransformer API, and the API returns update row count values to the application.

See "Using jXTransformer Queries and Write Statements in a Java Application" on page 223 for more information about using a jXTransformer write statement in a Java application.

jXTransformer Write Statement Example

The following example inserts new rows into the following database tables:

- Employees table: EmpID, FirstName, LastName, Title, StartDate, HourlyRate, and Resume columns
- EmpBenefits table: BenefitId, EmpId, Amount, and StartDate columns
- Assignments table: ProjId, EmpId, and Task columns

The values for the columns are retrieved from the following XML document named emp.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<insert>
  <employee ID="21" FirstName="Anne" LastName="Dodsworth"
    Title="Miss" StartDate="2001-10-24" HourlyRate="115">
    <resume><![CDATA[
      <a href='http://www.xesdemo/resume/21.htm'>
       A. Dodsworth</a>]]></resume>
    <benefits>
      <benefit ID="1" Amount="1"/>
      <benefit ID="2" Amount="175" StartDate="2001-11-01"/>
    </benefits>
    <projects>
      <project ID="8">
        <task>Analysis</task>
        <task>Development</task>
      </project>
      <project ID="9">
        <task>Analysis</task>
      </project>
    </projects>
  </employee>
</insert>
```

Insert statement:

```
insert xml_document('emp.xml', 1)
  into Employees (EmpId, FirstName, LastName, Title,
    StartDate, HourlyRate, Resume)
    xml_row_pattern('/insert/employee')
    values(xml_xpath('@ID', 0, 'Integer'),
           xml_xpath('@First'),
           xml_xpath('@LastName'),
           xml_xpath('@Title'),
           xml_xpath('@StartDate', 'Timestamp'),
           xml_xpath('@HourlyRate', 'Integer'),
           xml_xpath('resume[1]/text()')
          )
  into EmpBenefits (BenefitId, EmpId, Amount, StartDate)
    xml_row_pattern('/insert/employee/benefits/benefit')
    values(xml_xpath('@ID', 'Integer'),
           xml_xpath('../../@ID', 'Integer'),
           xml_xpath('@Amount', 'Integer'),
           xml_xpath('@StartDate', 'Timestamp')
          )
  into Assignments (ProjId, EmpId, Task)
  xml_row_pattern('/insert/employee/projects/project/task')
    values(xml_xpath('../@ID', 'Integer'),
           xml_xpath('../../../@ID', 'Integer'),
           xml_xpath('text()')
          )
```

Results:

One new row inserted into the Employees table:

EmpId	LastName	FirstName	Title	StartDate	EndDate	HourlyRate	Resume
21	Dodsworth	Anne	Miss	2001-10-24		115	<a href= 'http://www.xesdemo/ resume21.htm'> A.Dodsworth</a>

Two new rows inserted into the EmpBenefits table:

EmpId	BenefitId	StartDate	EndDate	Amount
21	1	2001-10-24		1
21	2	2001-22-01		175

Three new rows inserted into the Assignments table:

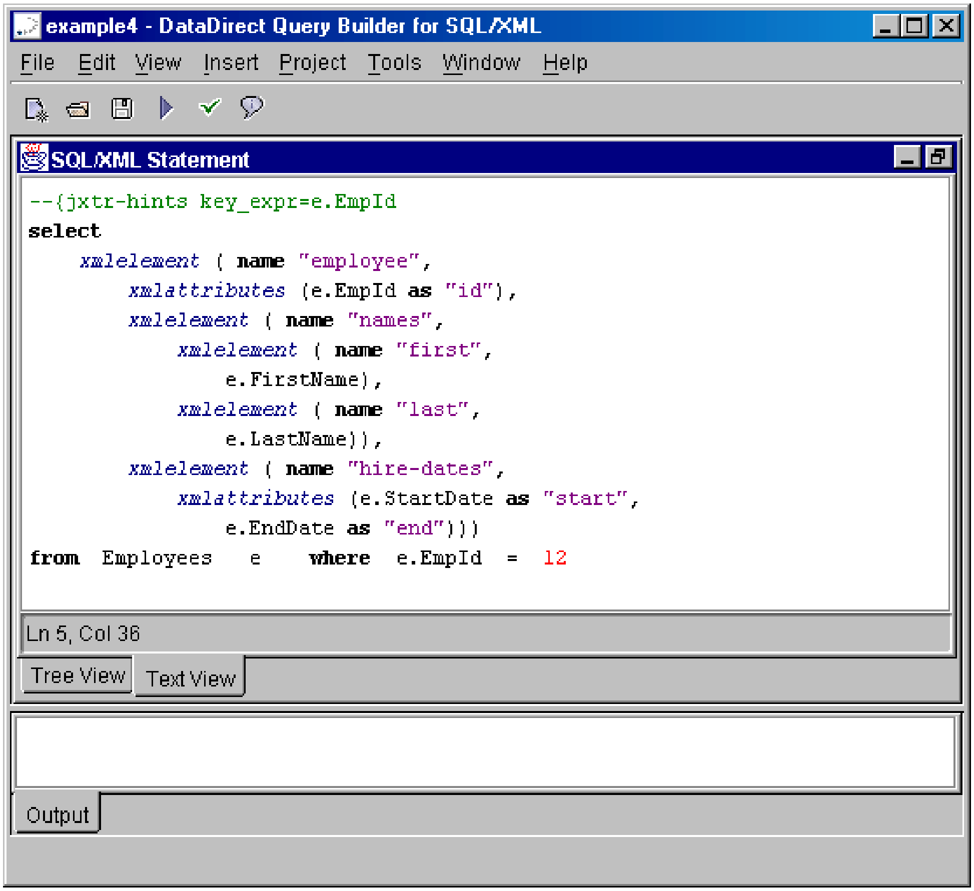
EmpId	ProjId	Task	StartDate	EndDate	TimeUsed	EstimatedDuration
21	8	Analysis				
21	8	Development				
21	9	Analysis				

See Chapter 6, "Syntax of jXTransformer Write Statements" on page 173 for more information about the syntax for writing jXTransformer write statements.

DataDirect Query Builder *for* SQL/XML

The *DataDirect Query Builder for SQL/XML* is a graphical user interface tool that helps you create and modify Connect *for* SQL/XML queries, both SQL/XML and jXTransformer queries. It also allows you to test jXTransformer write statements. The following example screen shows a SQL/XML query defined using the Builder.

Figure 1-4. DataDirect Query Builder for SQL/XML



example4 - DataDirect Query Builder for SQL/XML
File Edit View Insert Project Tools Window Help
SQL/XML Statement

```
--{jxtr-hints key_expr=e.EmpId
select
    xmlelement ( name "employee",
        xmlattributes (e.EmpId as "id"),
        xmlelement ( name "names",
            xmlelement ( name "first",
                e.FirstName),
            xmlelement ( name "last",
                e.LastName)),
        xmlelement ( name "hire-dates",
            xmlattributes (e.StartDate as "start",
                e.EndDate as "end")))
from  Employees   e    where  e.EmpId  =  12
```

Ln 5, Col 36
Tree View
Text View
Output

You also can use the DataDirect Query Builder *for* SQL/XML to check the syntax of queries and write statements before you use them in your Java applications. You can save a Connect *for* SQL/XML query as a Builder project file for future fine-tuning or reuse.

See Chapter 3, "Creating Connect for SQL/XML Queries Using the Builder" on page 61 for more information about using the DataDirect Query Builder *for* SQL/XML to create Connect *for* SQL/XML queries.

For a tutorial on using the DataDirect Query Builder *for* SQL/XML to create:

- SQL/XML queries, see Chapter 9, "Tutorial: Using SQL/XML Queries" on page 233
- jXTransformer queries, see Chapter 10, "Tutorial: Using jXTransformer Queries" on page 253

Connecting to the Database

The way you connect to the database differs slightly depending on whether you are using the SQL/XML JDBC driver or the jXTransformer API to execute statements. In both cases, an underlying DataDirect Technologies JDBC driver is used for the connection to the database. Connect *for* SQL/XML works only with DataDirect Technologies JDBC drivers.

For complete information about connecting to the database when you are using:

- SQL/XML JDBC driver, see "Connecting to the Database" on page 199
- jXTransformer API, see "Connecting to the Database" on page 216

2 Understanding Connect *for* SQL/XML

This chapter describes some key features of Connect *for* SQL/XML and provides examples to help you decide how to use them.

Using the Java Packages

Connect *for* SQL/XML contains two Java packages: com.ddtek.jdbc.jxtr and com.ddtek.jxtr. The com.ddtek.jdbc.jxtr package is mainly used for processing SQL/XML queries, and the com.ddtek.jxtr package is used for processing jXTransformer queries and jXTransformer write statements.

One interface, JXTRStatementFactory, within the com.ddtek.jdbc.jxtr package is used to create instances of com.ddtek.jxtr objects from a SQL/XML JDBC driver connection object.

Using Both SQL/XML and jXTransformer Statements in a Java Application

Both SQL/XML queries and jXTransformer statements can be used in the same Java application. In this case, you can either:

- Use a SQL/XML JDBC driver connection to process both types of statements. In this case, you must use the JXTRStatementFactory interface of the com.ddtek.jdbc.jxtr package to process the jXTransformer statements.
- Use a SQL/XML JDBC driver connection to process the SQL/XML queries and a DataDirect Technologies JDBC driver connection to process the jXTransformer statements.

When you are writing an application that uses both SQL/XML queries and jXTransformer statements, you may find it useful to use this interlace so that you only have one database connection in your application.

Using SQL/XML Queries

This section provides information to help you use SQL/XML queries to transform relational data into XML.

Using key_expr Hints

You can optimize processing of a SQL/XML query by specifying key_expr hints as comments in your SQL/XML query. A key_expr hint uniquely identifies each of the rows retrieved from the database. If you do not specify any key_expr hint, all the selected database columns concatenated together compose the key. Note that the key columns are used to link parent rows to child rows when you are using nested queries. In SQL/XML queries that do not contain nested queries, there is no need to specify key_expr hints. Notice that only the sorted_outer_union algorithm uses keys; see the next section for a definition of this algorithm.

See "Using Connect for SQL/XML Hints" on page 195 for more information about hints.

Using rewrite_algorithm Hints

A SQL/XML query cannot be passed directly to the underlying database system; therefore, the SQL/XML JDBC driver translates the query to one or multiple SQL Select statements. The SQL/XML JDBC driver uses one of the following rewrite algorithms to perform this translation:

- Nested loop
- Sorted outer union

You can specify which algorithm to use by specifying a rewrite_algorithm hint in your SQL/XML query. Hints are placed within comments in the SQL/XML query. If no algorithm is specified, the driver uses sorted outer union.

The following list explains why you would use one algorithm versus the other:

- If a SQL/XML query does not contain nested queries and you want the order of the data to be determined by the database system, use the nested loop algorithm. By default, the sorted outer union algorithm is used, and data is sorted according to key columns specified in the SQL/XML query.

- If you are connecting to a Sybase or an Informix database and your SQL/XML query contains nested queries where both parent and child queries contain SQL aggregate functions, you must use the nested loop algorithm.

See “Using Connect for SQL/XML Hints” on page 195 for more information about hints.

Creating Result Sets From SQL Queries

In addition to creating result sets from SQL/XML queries, the SQL/XML JDBC driver also allows you to generate result sets from normal SQL queries. These result sets do not contain XML values unless the values are stored in the database as XML data types. For example, if you process the following SQL query using the SQL/XML JDBC driver:

```
SELECT e.FirstName, e.LastName
FROM Employees e AS Employees
WHERE e.EmpId < 3
```

Result Set:

Employees	
Marc	Marshall
Brian	Ayers

This process is no different than returning a result set from a SQL query with any DataDirect Technologies JDBC driver. See “Connecting to the Database” on page 199 for instructions on connecting with the SQL/XML JDBC driver.

Using jXTransformer Queries

This section provides information to help you use jXTransformer queries to transform relational data into XML.

Creating Hierarchical XML Documents

Using the jXTransformer query syntax, you can create hierarchical XML documents and define parent and child relationships for data from one or multiple database tables. Also, you can use nested jXTransformer queries to link data from multiple database tables and hierarchically structure that data.

Let us look at a simple jXTransformer query that retrieves data from one database table and creates a hierarchical XML document with one parent XML element named Employees_info, and with one child element named name that is parent to two child elements named first and last.

```
SELECT
xml_element('Employees_Info',
    xml_attribute('Department', e.Dept),
    xml_element('name',
          xml_element('first', e.FirstName),
          xml_element('last', e.LastName) ) )
FROM Employees e WHERE e.Dept = 'QA'
```

XML Result:

```
<Employees_Info Department='QA'>
   <name>
      <first>Paul</first>
      <last>Steward</last>
   </name>
</Employees_Info>
<Employees_Info Department='QA'>
   <name>
      <first>John</first>
      <last>Jenkins</last>
   </name>
</Employees_Info>
```

Now, look at another query that uses nested jXTransformer queries. In this example, the nested query uses the e.EmpId column to link the rows selected in the parent query with the rows selected in the nested, or child, query. Using xml_attribute_key in the parent query creates an XML attribute and specifies that the e.EmpId column in the database uniquely identifies the rows retrieved from the first query. Also, it specifies that the attribute value uniquely defines the parent element named employee.

```
SELECT
  xml_element('employee',
    xml_attribute_key('ID', e.EmpId),
    xml_attribute('name',
   {fn concat({fn concat(e.FirstName, ' ')},e.LastName)}),
    (SELECT
      xml_element('project',
        xml_attribute('name',p.Name),
        xml_attribute('task',a.Task))
    FROM Projects p, Assignments a
    WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
FROM Employees e WHERE e.EmpId < 3
```

XML Result:

```
<employee ID='1' name='Marc Marshall'
  <project name='Medusa' task='Analysis'></project>
    <project name='Medusa' task='Documentation'></project>
    <project name='Medusa' task='Planning'></project>
    <project name='Medusa' task='Testing'></project>
    <project name='Phoenix' task='Analysis'></project>
    <project name='Phoenix' task='Documentation'></project>
</employee>
<employee ID='2' name='Brian Ayers'
  <project name='Python' task='Analysis'></project>
    <project name='Python' task='Development'></project>
    <project name='Hydra' task='Analysis'></project>
    <project name='Hydra' task='Documentation'></project>
</employee>
```

NOTE: When you use nested queries, you must use an explicit key, specified through one, or a combination, of xml_attribute_key, xml_element_key, and xml_hide_key; otherwise, Connect *for* SQL/XML creates an implicit key by making all the columns specified in the top-level jXTransformer query part of the key, in which case, the query results could be incorrect.

See "Choosing an XML Document Structure" on page 53 for more information about choosing an XML document structure.

Using Keys

You can optimize processing of a jXTransformer query by specifying key columns using the "key" keyword. If you do not specify any key columns, Connect *for* SQL/XML automatically assumes that all the selected columns concatenated together compose the key. Notice that the key columns are used to link parent rows to child rows when you are using jXTransformer nested queries. In jXTransformer queries that do not contain nested queries, there is no need to specify key columns.

The following jXTransformer query constructors support the "key" keyword:

- xml_element_key
- xml_attribute_key
- xml_hide_key

Look at a jXTransformer query that uses the jXTransformer constructor xml_hide_key. In this example, xml_hide_key is used to relate parent rows to child rows.

```
SELECT
  xml_element('employee',
    xml_hide_key(e.EmpId),
    xml_attribute('name',
   {fn concat({fn concat(e.FirstName, ' ')},e.LastName)}),
    (SELECT
      xml_element('project',
        xml_attribute('name',p.Name),
        xml_attribute('task',a.Task))
    FROM Projects p, Assignments a
    WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
FROM Employees e WHERE e.EmpId < 3
```

XML Result:

```
<employee name='Marc Marshall'>
    <project name='Medusa' task='Analysis'></project>
    <project name='Medusa' task='Documentation'></project>
    <project name='Medusa' task='Planning'></project>
    <project name='Medusa' task='Testing'></project>
    <project name='Phoenix' task='Analysis'></project>
    <project name='Phoenix' task='Documentation'></project>
</employee>
<employee name='Brian Ayers'>
    <project name='Python' task='Analysis'></project>
    <project name='Python' task='Development'></project>
    <project name='Hydra' task='Analysis'></project>
    <project name='Hydra' task='Documentation'></project>
</employee>
```

NOTE: When you use nested queries, you must use an explicit key, specified through one, or a combination, of xml_attribute_key, xml_element_key, and xml_hide_key; otherwise, Connect *for* SQL/XML creates an implicit key by making all the columns specified in the top-level jXTransformer query part of the key, in which case, the query results could be incorrect.

Hiding Information

The jXTransformer query xml_hide[_key] constructor allows you to select and "hide" database columns you do not want to display in the resulting XML document. For example, suppose you have a database table that contains the following employee information:

SSN (primary key)
LastName
FirstName
JobTitle
Department
Salary
HireDate
ParentSSN

From this database table, you want to retrieve the names and salaries of the employees in the "QA" department as well as list dependents for those employees. In this case, to optimize processing, you want to write a jXTransformer query that selects the SSN column (a primary key), but you do not want to display the SSN information in the XML document, so you use the xml_hide_key construct to select the SSN data. The jXTransformer query might look like:

```
SELECT
  xml_element('QAemployees',
    xml_hide_key(e.SSN),
    xml_element('Salaries',
      xml_element('first', e.FirstName),
      xml_element('last', e.LastName),
      xml_element('salary', e.Salary) )
      (SELECT
       xml_element('children',
          xml_attribute('first-name', c.first),
          xml_attribute('last-name', c.last) )
       FROM Employees C WHERE c.ParentSSN=e.SSN) )
FROM Employees e WHERE e.Department='QA'
```

This query retrieves information about employees in the QA department and lists the names of their dependents. In this query, using xml_hide_key optimizes processing of the query by linking information about the parents with information about their children through a common denominator, the SSN information.

Creating ID/IDREFS Links

The jXTransformer query syntax allows you to create IDREFS as values for XML attributes. IDREFS are references to unique attribute ID values for XML elements in the same XML document.

You can use the jXTransformer query constructor xml_attribute to create IDREFS using the following syntax:

```
xml_attribute ('xml_attribute_name', sql99_select)
```

When you specify a SQL query, a space-separated concatenation of the complete result set is created; these values typically are IDREFS for an ID attribute of another XML element in the same XML document.

Using the xml_attribute query constructor, in combination with multiple top-level queries within a jXTransformer query, you can create ID/IDREFS links in the resulting XML document. To specify multiple top-level queries, separate the queries with a semicolon (;). The result is a single XML document that contains the concatenation of the results of the different top-level queries.

Let us take a look at an example that creates ID/IDREFS links.

```
                           SELECT
                             xml_element('employees',
                               xml_attribute_key('emp-id',
                                {fn concat('e-',{fn convert(e1.EmpId,VARCHAR)})}),
                               xml_attribute('emp-name',e1.LastName),
First query                    xml_attribute('emp-benefits',
                                 SELECT
                                   {fn concat('b-',
Creates IDREFS                         {fn convert(eb1.BenefitId,VARCHAR)})}
                                 FROM EmpBenefits eb1 WHERE eb1.EmpId=e1.EmpId))
                           FROM Employees e1 WHERE exists
                             (SELECT * from EmpBenefits eb2 WHERE e1.EmpId=eb2.EmpId)
Semicolon separating       ;
queries                    SELECT
                             xml_element('benefits',
                               xml_attribute_key('benefit-id',
                                 {fn concat('b-',
                                   {fn convert(b2.BenefitId,VARCHAR)})}),
                               xml_attribute('benefit-description',b2.Description),
                               xml_attribute('benefit-employees',
Second query                     SELECT
                                   {fn concat('e-',
                                       {fn convert(eb3.EmpId,VARCHAR)})}
Creates IDREFS                   FROM EmpBenefits eb3
                                 WHERE eb3.BenefitId=b2.BenefitId))
                           FROM Benefits b2
                           WHERE exists
                             (SELECT * FROM EmpBenefits eb4
                             WHERE b2.BenefitId=eb4.BenefitId)
```

XML Result:

```
<employees emp-id="e-1" emp-name="Marshall"
  emp-benefits="b-1 b-3" />
<employees emp-id="e-12" emp-name="Steward"
  emp-benefits="b-3" />
<employees emp-id="e-2" emp-name="Allen"
  emp-benefits="b-1 b-4" />
<benefits benefit-id="b-1" benefit-description="Bonus"
  benefit-employees="e-1 e-2" />
<benefits benefit-id="b-3" benefit-description="Car"
  benefit-employees="e-1 e-12" />
<benefits benefit-id="b-4" benefit-description="Commission"
  benefit-employees="e-2" />
```

Creating XML Document Fragments

The jXTransformer query syntax allows you to create complete XML documents and XML document fragments. Unlike a document fragment, a complete XML document contains an XML root element; in addition, it can contain processing instructions, comments, and a reference to a private or public external DTD.

Typically, you create XML document fragments to insert into an existing DOM or JDOM document. For example, suppose your company wants to produce an XML document that contains a sales report. This sales report contains a message to all sales representatives as well as sales data that is retrieved from a relational database. In this scenario, you first create a DOM document that contains the message. Then, you use a jXTransformer query to retrieve the sales data from the database and to place the XML results into the existing DOM document.

The Java application code required for this scenario might look like:

```
import java.io.OutputStreamWriter;
import java.sql.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;

import com.ddtek.jxtr.*;
{
//Load properties from the resource
loadProperties();

//Create JDBC connection
jdbcConnect();

//Build jXTransformer query
StringBuffer jxtrQ = new StringBuffer();
jxtrQ.append ( "select ");
jxtrQ.append ( "  xml_element('Order-by-SalesID', ");
jxtrQ.append ( "     xml_attribute('Salesperson', o.SalesID), ");
jxtrQ.append ( "     xml_attribute('CustomerID', o.CustomerID), ");
jxtrQ.append ( "  xml_element('OrderDate', o.OrderDate), ");
jxtrQ.append ( "  xml_element('ShipDate', o.ShipDate), ");
jxtrQ.append ( "  xml_element('OrderAmount', o.OrderAmt) ) ");
jxtrQ.append ( "from Orders o order by o.SalesID " );

//Construct new JXTRQuery object
JXTRQuery jxtrQuery = JXTRQuery (conn, new String ( jxtrQ ) )

// Create DOM Document through JAXP
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance ();
DocumentBuilder db = dbf.newDocumentBuilder ();
Document doc = db.newDocument ();

// Create root node and place it in DOM document
domRoot = doc.createElementNS ( null, "sales-root" );
doc.appendChild ( domRoot );
// Create a child node for the root node and append it
Node domS1 = doc.createElementNS ( null, "sales-query-result" );
domRoot.appendChild ( domS1 );

// Create text content for the subelement
domS1.appendChild ( doc.createTextNode ( "Text message. . ." ) );

// Execute and place the result into the previously created DOM node
jxtrQuery.executeDOM ( domS1 );

// Close JDBC connection
jdbcDisconnect();

}
```

Creating XML in Result Sets

In addition to creating complete XML documents and XML document fragments, you can also return XML values in result sets from jXTransformer queries. For example, you may want to return a large XML result of a jXTransformer query as a result set so that you can process it row by row. To create a result set, use the executeQuery method in the JXTRQuery class. Once you have returned the XML as a result set, you can use the methods in the JXTRColInfo class to work with those XML results.

Generating DTDs and Schemas

You can generate a DTD or schema that describes the structure of the XML result of a jXTransformer query using the jXTransformer API or the DataDirect Query Builder *for* SQL/XML. For example, you may want to distribute a DTD or schema you have generated describing the XML results of a jXTransformer query to a business partner.

The JXTRQuery class implements the methods you can use to generate DTDs or schemas based on the XML result of a jXTransformer query.

See "Generating DTDs and XML Schemas" on page 133 for instructions on generating DTDs or XML schemas using the DataDirect Query Builder *for* SQL/XML.

Choosing an XML Document Structure

jXTransformer queries provide the flexibility to define a complex hierarchical structure for XML documents. jXTransformer queries also allow you to generate XML documents from SQL queries. In this case, the rows and columns returned from the SQL queries are transformed into either to a flat-attribute or element-centric XML structure (see the following sections for definitions).

Hierarchical XML

For hierarchical XML structures, all data retrieved from the database is structured as defined by parent and child relationships, and nested queries, using the jXTransformer query syntax. The JXTRQuery class implements the methods to transform a jXTransformer query into an XML structured document. See "Creating Hierarchical XML Documents" on page 43 for more information about structured XML.

Flat Attribute and Element-Centric Structures

The following XML documents are generated from executing a SQL query, rather than a jXTransformer query:

- For flat-attribute XML structures, all data retrieved from the database is structured inside attributes, one attribute for each column. One element is created for each row retrieved and the column values are defined as its attributes. The attribute names are derived from the database column names.

- For element-centric XML structures, all data retrieved from the database is structured inside elements. One element is created for each row retrieved and the column values are defined as its subelements. The element names are derived from the database column names.

The JXTRResultSetWrapper class implements the methods required to execute a SQL query to generate attribute- and element-centric XML documents.

In the following example, data is retrieved from the FirstName and LastName table columns and is placed in an attribute-centric or element-centric XML document.

```
SELECT e.FirstName, e.LastName
FROM Employees e
WHERE e.EmpId < 3
```

Attribute-Centric Result:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jxtr-result>
    <row FirstName='Marc' LastName='Marshall'/>
    <row FirstName='Brian' LastName='Ayers'/>
</jxtr-result>
```

Element-Centric Result:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jxtr-result>
     <row>
          <FirstName>Marc</FirstName>
          <LastName>Marshall</LastName>
     </row>
     <row>
          <FirstName>Brian</FirstName>
          <LastName>Ayers</LastName>
     </row>
</jxtr-result>
```

Choosing an XML Output Format

The JXTRQuery and JXTRResultSetWrapper classes implement the methods listed in Table 2-1 for generating XML documents. These two classes are in the com.ddtek.jxtr package.

Table 2-1. jXTransformer Query XML Output Options

XML Document Output Options	Methods
Write the XML document as a character stream to a Writer object	executeWriter
Return the XML document as a DOM level 2 document object	executeDOM
Create the XML document under an existing DOM level 2 element node	executeDOM
Return the XML document as a JDOM tree	executeJDOM
Create the XML document under an existing JDOM tree element node	executeJDOM
Invoke SAX2 callbacks as XML document results are being retrieved from the database	executeSAX
Create the XML within a result set	executeQuery

Choosing a Rewrite Algorithm

A jXTransformer query cannot be passed to the underlying database system; therefore, Connect *for* SQL/XML translates a jXTransformer query to one or multiple SQL Select statements or uses one of the following jXTransformer rewrite algorithms to perform this translation:

- Nested loop
- Sorted outer union
- Outer union
- For XML explicit

For a complete description of each rewrite algorithm, refer to the Javadoc shipped with this product.

In most cases, you do not need to know the rewrite algorithm Connect *for* SQL/XML uses to translate a jXTransformer query; however, you may find it helpful to know the following information about the rewrite algorithms:

- If a jXTransformer query does not contain nested queries, the nested loop algorithm is used for the translation. When the nested loop algorithm is used, the order of the data is determined by the database system.

- For Informix and Sybase only: If your jXTransformer query contains nested queries where both parent and child queries contain aggregate functions, Connect *for* SQL/XML uses the nested loop algorithm for translation. The nested loop algorithm must be used.

- Do not use the outer union algorithm if your jXTransformer query contains Distinct clauses.

The JXTRQuery class contains methods that allow you to specify which rewrite algorithm you want to use for the translation of your jXTransformer query.

Using jXTransformer Write Statements

This section provides information you need to know to use jXTransformer write statements to write data from XML documents into your relational database.

jXTransformer Write Statement Processing Overview

When the jXTransformer API executes a jXTransformer write statement (Insert, Update, or Delete statements), the following events occur:

1 The jXTransformer API validates the jXTransformer write statement syntax.

2 One or multiple SQL99 Insert, Update, or Delete statements are parsed from the jXTransformer write statements.

3 For each of the SQL99 Insert, Update, or Delete statements resulting from the previous action, the following tasks are performed:

  a A JDBC preparedStatement is created.

  b The xml_row_pattern XPath expressions in the statements are evaluated against the input XML document and the nodes specified by the XPath expressions are returned.

  c For each of the nodes returned from the xml_row_pattern XPath expressions, the following tasks are performed:

  - Each xml_xpath expression is evaluated and the appropriate set*XXX* method is invoked on the JDBC preparedStatement using the result of the xml_xpath expression.

- The java.sql.PreparedStatement.addBatch method is invoked on the JDBC prepared statement.
- The JDBC prepared statement is executed.

See Appendix B, "jXTransformer Query and Statement Processing" on page 297 for more information about jXTransformer write statement processing.

Disabling Autocommit Mode

When using jXTransformer write statements in Java applications, we recommend that Autocommit mode be disabled. Invoke con.setAutoCommit(false), execute the jXTransformer write statement, and then, invoke con.commit() or con.rollback().

Choosing an XML Input Format

The JXTRUpdate and JXTRSingleTableUpdate classes implement the methods listed in Table 2-2 for writing data from an XML document to a relational database.

Table 2-2. jXTransformer Write XML Document Options

XML Document Input Options	Methods
Read the XML document input as a character stream to a Reader object	setReader
Read the XML document input as a DOM level 2 document object	setDOM
Read the XML document input from a DOM level 2 element node	setDOM
Read the XML document input as a JDOM tree	setJDOM

Table 2-2. jXTransformer Write XML Document Options(cont.)

XML Document Input Options	Methods
Read the XML document input from a JDOM tree element node	setJDOM
Read the XML document input from a SAX2 input source	setSAX

3 Creating Connect *for* SQL/XML Queries Using the Builder

This chapter explains how to create SQL/XML and jXTransformer queries using the DataDirect Query Builder *for* SQL/XML (the Builder). The Builder is a Java application that makes it easy for you to create SQL/XML and jXTransformer queries without having detailed knowledge of the query syntax.

For more information about:

- SQL/XML query syntax, see Chapter 4, "Syntax of SQL/XML Queries" on page 141
- jXTransformer query syntax, see Chapter 5, "Syntax of jXTransformer Queries" on page 153

HTML-based online help for the Builder is placed on your system during the installation of your DataDirect product. To access help for the Builder, you must have Internet Explorer 5.x or higher, or Netscape 4.x or higher installed. Open the Builder help by selecting **Help / Contents**.

NOTE: Depending on your configuration, you may have to configure the Web browser as described in "Changing the GUI's General Appearance" on page 74 before you can use the Builder help or the Web links in the Help menu.

For installation requirements and instructions for Connect *for* SQL/XML and the Builder, refer to the *Connect for SQL/XML Installation Guide*.

Working with the Builder

Using the Builder, you can create and store SQL/XML and jXTransformer queries in a SQL/XML Builder project file. Builder project files for SQL/XML queries have the file extension .cfb; Builder project files for jXTransformer queries (and jXTransformer write statements) have the file extension .jxb.

The Builder allows you to create SQL/XML or jXTransformer queries, execute the queries, and view the resulting XML using the following types of views:

- **Tree view** uses nodes in a project tree to represent SQL/XML and jXTransformer query constructs. To construct SQL/XML and jXTransformer queries, you can add, modify, delete, and move these nodes using position-sensitive menu commands and drag-and-drop functionality. Detailed knowledge of the SQL/XML or jXTransformer query syntax is not required to construct a query in this view. To work in Tree view, select the Tree View tab at the bottom of the SQL/XML Statement window or jXTransformer Statement window.

- **Text view** allows you to construct SQL/XML and jXTransformer queries using their respective query syntax. To work in Text view, select the Text View tab at the bottom of the SQL/XML Statement window or jXTransformer Statement window. For more information about:

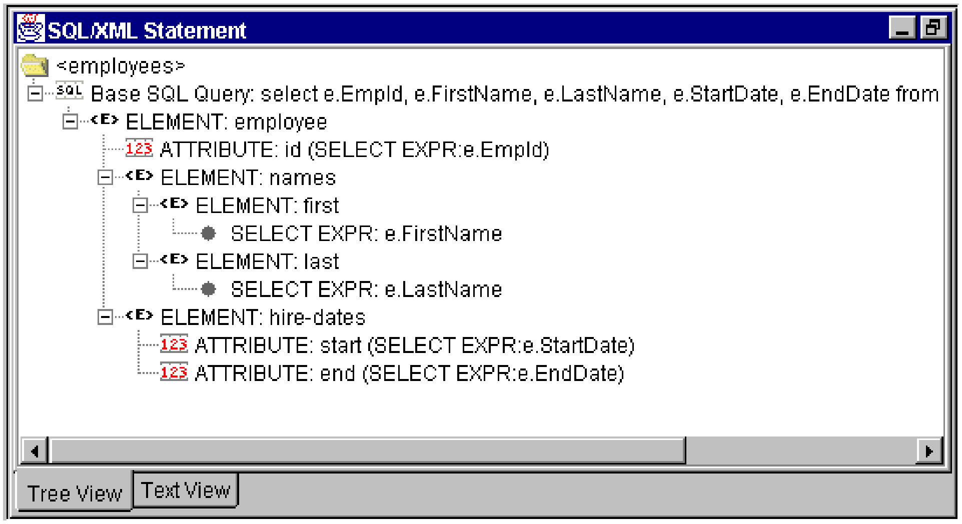
  - SQL/XML query syntax, see Chapter 4, "Syntax of SQL/XML Queries" on page 141
  - jXTransformer query syntax, see Chapter 5, "Syntax of jXTransformer Queries" on page 153

  Text view also allows you to execute jXTransformer write statements. See "Executing jXTransformer Write Statements" on page 137 for instructions.

Using Tree and Text Views

Using Tree view, you can create and modify SQL/XML and jXTransformer queries without having to know the syntax of their respective query syntax. The SQL/XML and jXTransformer query constructs are represented as nodes in a Builder project tree, similar to a DOM tree, as shown in Figure 3-1. You can structure the data you retrieve from the database any way you want it in the resulting XML by adding, modifying, deleting, and moving nodes. The following figure shows a Tree view of a SQL/XML query.

Figure 3-1. jXTransformer Query in Tree View



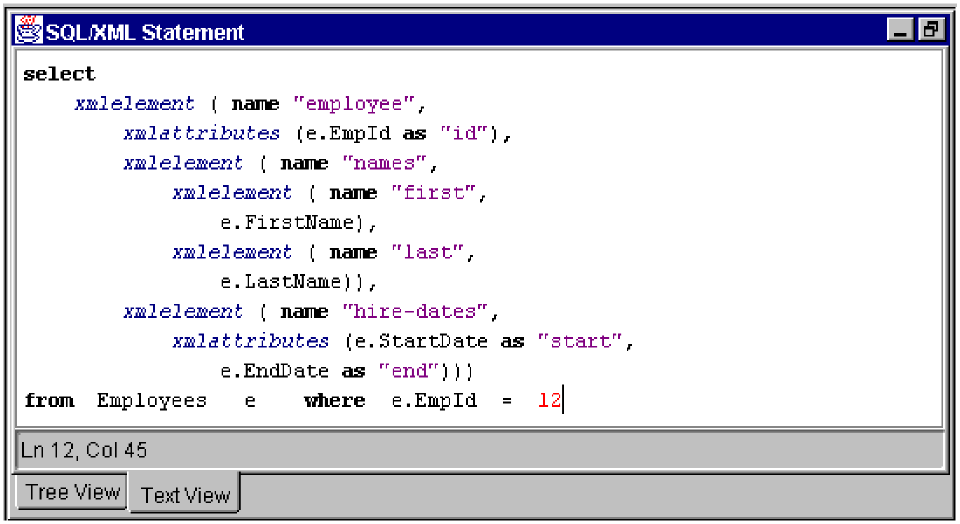
```
<employees>
  Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate from
    ELEMENT: employee
      ATTRIBUTE: id (SELECT EXPR:e.EmpId)
      ELEMENT: names
        ELEMENT: first
          SELECT EXPR: e.FirstName
        ELEMENT: last
          SELECT EXPR: e.LastName
      ELEMENT: hire-dates
        ATTRIBUTE: start (SELECT EXPR:e.StartDate)
        ATTRIBUTE: end (SELECT EXPR:e.EndDate)
```

Each node in the Builder project tree indicates the type of query construct that node represents. See "Using Project Tree Nodes" on page 65 for a list of the node types you can use in the Builder project tree.

When you create SQL/XML and jXTransformer queries using a Tree view, you can easily see the query syntax used in the source of the query by switching to a Text view. The following figure shows a Text view of a SQL/XML query.

The same query shown in Tree view in Figure 3-1 is displayed in Text view in Figure 3-2.

Figure 3-2. jXTransformer Query in Text View



```
select
    xmlelement ( name "employee",
        xmlattributes (e.EmpId as "id"),
        xmlelement ( name "names",
            xmlelement ( name "first",
                e.FirstName),
            xmlelement ( name "last",
                e.LastName)),
        xmlelement ( name "hire-dates",
            xmlattributes (e.StartDate as "start",
                e.EndDate as "end")))
from  Employees   e    where  e.EmpId  =  12
```

Conversely, if you want to work in Text view and create a query using the SQL/XML or jXTransformer query syntax, you can view the resulting changes in the Builder project tree by switching to Tree view.

For more information about:

- SQL/XML query syntax, see Chapter 4, “Syntax of SQL/XML Queries” on page 141
- jXTransformer query syntax, see Chapter 5, “Syntax of jXTransformer Queries” on page 153

NOTE: Before you can switch between Tree view and Text view, the syntax of your query must be correct. The Builder generates a message if an error is encountered. If an error is encountered, you cannot switch views until you correct the error.

Using Project Tree Nodes

The node types you can create in a Builder project in Tree view depend on whether you are creating a SQL/XML or jXTransformer query.

SQL/XML Queries

Table 3-1 lists the SQL/XML query node types you can create in a Builder project in Tree view and provides a description of each node type. Notice that most of the node types correspond to operators in the SQL/XML query syntax. See Chapter 4, “Syntax of SQL/XML Queries” on page 141 for information about SQL/XML query syntax.

Table 3-1. Builder Project Tree Nodes for SQL/XML Queries

SQL/XML Nodes	Description
project node	Specifies the parent node for the SQL/XML query. This node is required and is created when you create a Builder project. It cannot be modified, moved, or deleted. This node is used only in the Tree view and does not correspond to a SQL/XML operator.

***Table 3-1. Builder Project Tree Nodes for SQL/XML Queries** (cont.)*

SQL/XML Nodes	Description
Base SQL Query node	Specifies a SQL query on which all or part of the SQL/XML query is based. This node is required. When specifying table names, you must use unique table aliases. This node is used only in Tree view and does not correspond to a SQL/XML operator. See "Using Base SQL Query Nodes" on page 71 for more information.
ELEMENT node	Specifies an XML element. This node can have multiple values and mixed content. This node corresponds to the XMLELEMENT operator.
ATTRIBUTE node	Specifies an XML attribute associated with an ELEMENT node. This node corresponds to a single attribute specified by the XMLATTRIBUTES operator.
SELECT EXPR node	Specifies a Select expression value for an ELEMENT node.
CONSTANT node	Specifies a constant value for an ELEMENT node.
SELECT EXP KEY	Specifies that a value that is a Select expression for an ELEMENT node is also a key. This node corresponds to a Connect *for* SQL/XML hint; it does not correspond to a SQL/XML operator.
ATTRIBUTE KEY	Specifies that a value that is a Select expression for an ATTRIBUTE node is also a key. This node corresponds to a Connect *for* SQL/XML hint; it does not correspond to a SQL/XML operator.
CONCAT node	Specifies a forest of elements that is produced by concatenating a list of XML values. This node corresponds to the XMLCONCAT operator.

Table 3-1. Builder Project Tree Nodes for SQL/XML Queries (cont.)

SQL/XML Nodes	Description
FOREST node	Specifies a forest of elements that is produced from a list of arguments. This node corresponds to the XMLFOREST operator.
FOREST ELEMENT node	Specifies an element that belongs to a forest of elements. This node corresponds to an argument accepted by the XMLFOREST operator.
AGG node	Specifies a forest of elements that is produced from a collection of XML elements. This node corresponds to the XMLAGG operator.

See "Creating and Modifying SQL/XML Queries in Tree View" on page 80 for instructions on using these nodes to create SQL/XML queries in Tree view of the Builder.

jXTransformer Queries

Table 3-2 lists the jXTransformer node types you can create in a Builder project in Tree view and provides a description of each node type. Notice that most of the node types correspond to keywords in the jXTransformer query syntax. See Chapter 5, "Syntax of jXTransformer Queries" on page 153 for information about jXTransformer query syntax.

Table 3-2. Builder Project Nodes for jXTransformer Queries

jXTransformer Nodes	Description
project node	Specifies the parent node for the jXTransformer query, which can contain one or multiple sub-queries. This node is required and is created when you create a jXTransformer Builder project. It cannot be modified, moved, or deleted. This node is used only in the Tree view and does not correspond to a jXTransformer keyword.
ROOT ELEMENT node	Specifies the root element to be used for the resulting XML document. When you turn on the document header, a ROOT ELEMENT node is inserted in the project tree with the default root element name jxtr-result. This node corresponds to the xml_element keyword when the keyword is used to specify a root element.
COMMENT node	Specifies comments in the XML document header. This node corresponds to the xml_comment keyword.
PROCESSING INSTRUCTION node	Specifies an XML processing instruction. This node corresponds to the xml_pi keyword.
EXTERNAL DTD node	Specifies an external public or private DTD. This node corresponds to the xml_external_dtd keyword.

Table 3-2. Builder Project Nodes for jXTransformer Queries (cont.)

jXTransformer Nodes	Description
Base SQL Query node	Specifies a SQL query on which all or part of the jXTransformer query is based. This node is required. When specifying table names, you must use unique table aliases. This node is used only in Tree view and does not correspond to a jXTransformer keyword. See "Using Base SQL Query Nodes" on page 71 for more information.
ELEMENT node	Specifies an XML element. This node can have multiple values and mixed content. This node corresponds to the xml_element keyword.
ATTRIBUTE node	Specifies an XML attribute associated with an ELEMENT node or a ROOT ELEMENT node. This node corresponds to the xml_attribute keyword.
HIDE node	Specifies a node that allows you to retrieve information from the database you do not want to include in the resulting XML document. This node corresponds to the xml_hide keyword.
CDATA node	Specifies an XML CDATA section. This node corresponds to the xml_cdata keyword.
NAMESPACE node	Specifies an XML namespace associated with an ELEMENT node or a ROOT ELEMENT node. This node corresponds to the xml_namespace keyword.
SELECT EXPR node	Specifies Select expression value for an ELEMENT node.
CONSTANT node	Specifies a constant value for an ELEMENT node.
SELECT EXP KEY	Specifies that a value that is a Select expression for an ELEMENT node is also a key. This node corresponds to the xml_element_key keyword.

Table 3-2. Builder Project Nodes for jXTransformer Queries (cont.)

jXTransformer Nodes	Description
ATTRIBUTE KEY	Specifies that a value that is a Select expression for an ATTRIBUTE node is also a key. This node corresponds to the xml_attribute_key keyword.
HIDE KEY	Specifies that a value that is a Select expression for a HIDE node is also a key. This node corresponds to the xml_hide_key keyword.

See "Creating and Modifying jXTransformer Queries in Tree View" on page 96 for instructions on using these nodes to create jXTransformer queries in Tree view of the Builder.

Working with Project Tree Nodes

You can expand or collapse any node in the project tree by double-clicking that node or by single-clicking the + (plus sign) or - (minus sign) for that node. You can also expand or collapse any node and its child nodes by right-clicking the node and selecting **Expand Branch** or **Collapse Branch**.

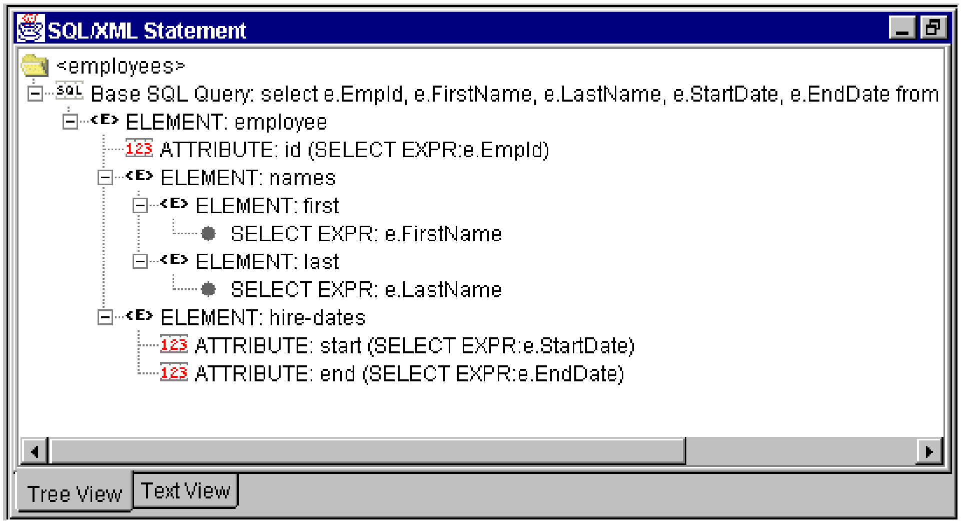
Builder menu commands in the Tree view are position-sensitive. When you create a node, the new node becomes a child of the selected node. For example, if you have an ELEMENT node selected when you select **Insert / Attribute Node**, the ATTRIBUTE node will be created as a child of that ELEMENT node.

Also, to help you create a valid query that complies with the SQL/XML or jXTransformer query syntax, the Builder menu commands that are available depend on the selected node.

Using Base SQL Query Nodes

The Base SQL Query node is a SQL query that forms a base for all or part of your query. It is a required node in the Builder project tree. It allows you to decide which data you want to retrieve from the database before you create the query and facilitates the process of constructing the query. Figure 3-3 shows a SQL/XML query in Tree view. Notice the Base SQL Query node is the first node under the project node named employees.

Figure 3-3. Base SQL Query in SQL/XML Query



```
<employees>
  Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate from
    ELEMENT: employee
      ATTRIBUTE: id (SELECT EXPR:e.EmpId)
      ELEMENT: names
        ELEMENT: first
          SELECT EXPR: e.FirstName
        ELEMENT: last
          SELECT EXPR: e.LastName
      ELEMENT: hire-dates
        ATTRIBUTE: start (SELECT EXPR:e.StartDate)
        ATTRIBUTE: end (SELECT EXPR:e.EndDate)
```

Starting the Builder

How you start the Builder depends on your platform:

- **On Windows**: Run the builder.bat file located in the Connect *for* SQL/XML installation directory.
- **On UNIX**: Run the builder.sh shell script located in the Connect *for* SQL/XML installation directory.

During a normal installation, the builder.bat file and builder.sh script are automatically customized to include the path to your JDK. If the installer was unable to detect a required JDK or you want to change the JDK to be used, refer to the *DataDirect Connect for SQL/XML Installation Guide* for instructions on configuring the startup file for the Builder.

Customizing the Builder

You can customize the following settings in the Builder:

- General appearance of the GUI, including:
  - Java Look and Feel theme
  - Font size of the text in Builder menus and dialog boxes
  - Web browser that is used to display the online help and access the Web links in the Help menu

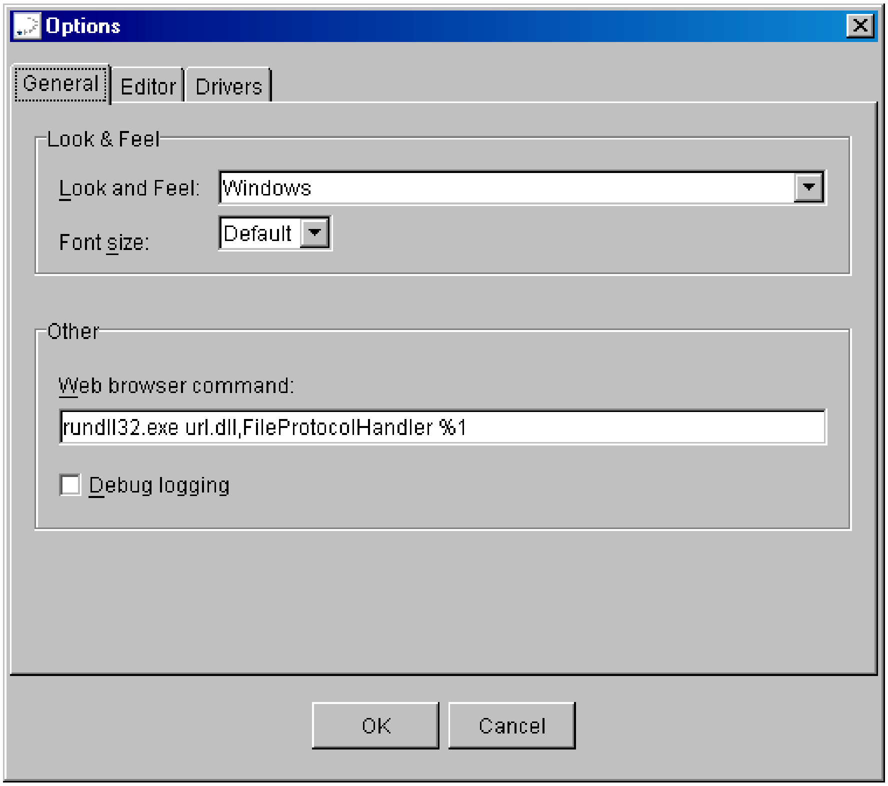
    NOTE: You may have to configure this setting before you can access the online help for the Builder and use the Web links in the Help menu. See "Changing the GUI's General Appearance" on page 74 for instructions.
  - Turn on and off debug logging

- Editor in the Text view of the SQL/XML Statement window and jXTransformer Statement window, as well as the XML window. For example, you may want SQL/XML query operators or jXTransformer keywords to appear in a blue color or bold type.
- DataDirect Technologies JDBC drivers that are not shipped with Connect *for* SQL/XML. A named driver for each of the Connect *for* JDBC drivers and the SequeLink JDBC driver shipped with Connect *for* SQL/XML is configured by default. To use this feature, select **Tools** / **Options**. Then, select the Drivers tab.

  NOTE: This feature is reserved for DataDirect Technologies use; only use when instructed to do so by DataDirect Technologies.

Changing the GUI's General Appearance

1 Select **Tools** / **Options**. The General tab appears.



Look and Feel: Windows

Font size: Default

Web browser command:

rundll32.exe url.dll,FileProtocolHandler %1

□ Debug logging

2 In the Look and Feel group, complete the following information:

Look and Feel: From the drop-down list, select the Java theme you want to use. The default is Metal. The screenshots in this book use the Windows Java theme.

Font Size: From the drop-down list, select the font size you want the Builder to use for text in menus and dialog boxes. The default is 12 points. The font size for the text and tree views in the SQL/XML Statement window and jXTransformer Statement window can be specified on the Editor tab.

3 In the Other group, complete the following information:

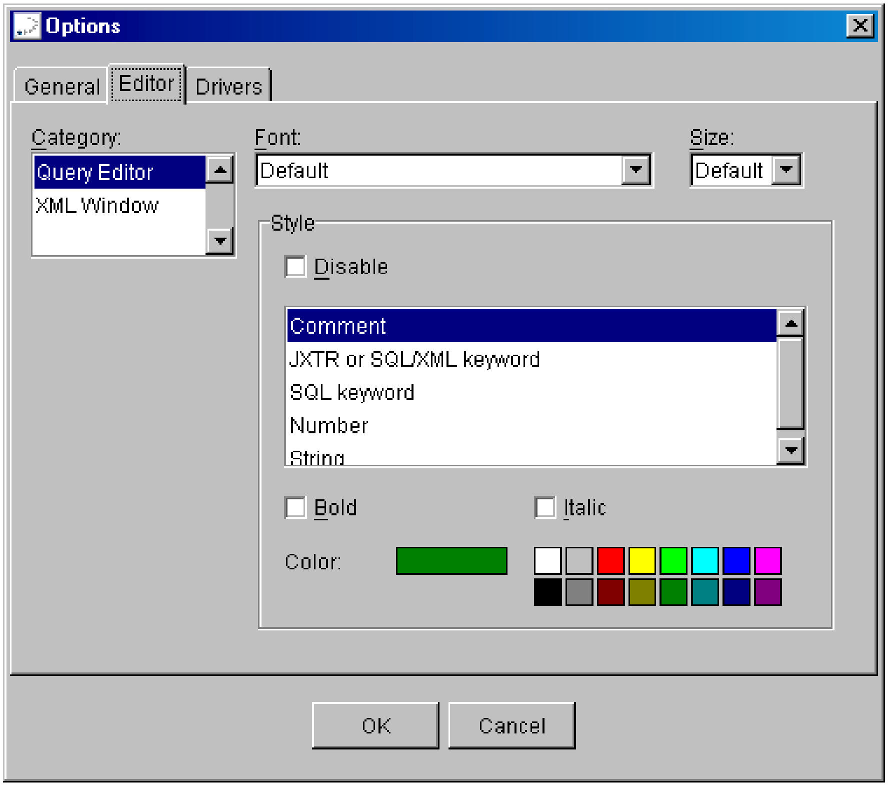
Web Browser Command: Type the full path of the Web browser executable to be used to display the online help and access the Web links from the Help menu. Append a space and `%1` to the executable. For example:

```
C:\Program Files\Internet Explorer\iexplore.exe %1
```

Debug logging: Debug logging is used for troubleshooting and dumps debug messages to your standard error stream. It is turned off by default. Check the Debug logging check box to turn debug logging on.

Changing the Text Editor

1 Select **Tools / Options**. Then, select the Editor tab.



Options

General | Editor | Drivers

Category: Query Editor

Font: Default

Size: Default

Style

□ Disable

Comment

□ Bold □ Italic

Color: 🟩

2. To change the appearance of the editor in the:

   - Text view of the SQL/XML Statement window and jXTransformer Statement window, in the Category list, select **Query Editor**. Continue with the next step.

   - XML window, in the Category list, select **XML Window**. The selections that are unavailable for the XML window become disabled. Continue with Step 4.

3. To change the appearance of the editor in the Text view of the SQL/XML Statement window and jXTransformer Statement window, complete the following information:

   Font: From the drop-down list, choose the font you want to use in the text editor.

   Size: From the drop-down list, choose the size of the font you want to use in the text editor. The default is 12 points.

   If you want to define a style for an item in the text editor, which include comments, SQL/XML query operators or jXTransformer query keywords, SQL keywords, numerical values, and strings, select that item in the Style group. You can distinguish any listed item by color, bold type, italic type, or any combination of these characteristics.

   NOTE: You can disable any style without deleting its definition by selecting the **Disable** check box in the Style group.

   Select your choices, and click **OK**.

4. To change the appearance of the editor in the XML window, complete the following information:

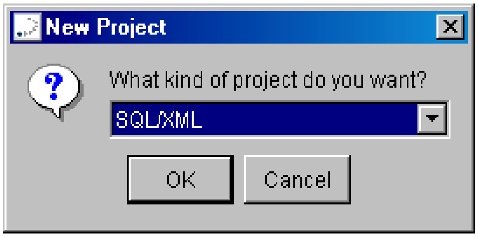
   Font: From the drop-down list, choose the font you want to use in the text editor.

   Size: From the drop-down list, choose the size of the font you want to use in the text editor. The default is 12 points.

   Select your choices, and click **OK**.

Creating a Builder Project

From the main menu, select **File** / **New Project**. The New Project dialog box appears, prompting you to select which type of project to create: SQL/XML or jXTransformer.



New Project

What kind of project do you want?

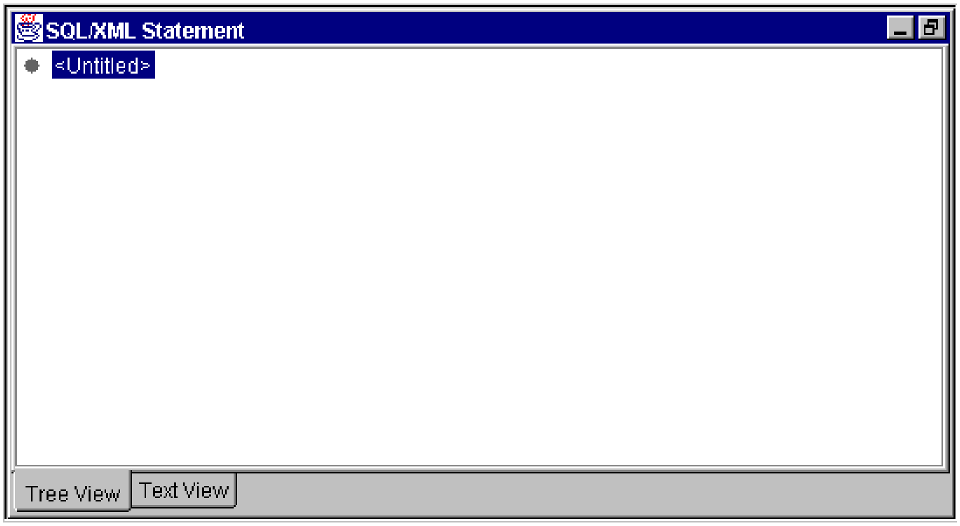
SQL/XML

From the drop-down list, select the type of Builder project to create:

- Select **SQL/XML** to create a Builder project for SQL/XML queries (.cfb).
- Select **jXTransformer** to create a Builder project for jXTransformer queries or write statements (.jxb).

An untitled project node appears in Tree view of the SQL/XML Statement window or jXTransformer Statement window, depending on the type of Builder project you chose to create.

The following figure shows a newly created SQL/XML project in the SQL/XML Statement window.



When you save the Builder project to a file, the project node is renamed to the name of the Builder project file. You cannot modify, move, or delete this node.

For instructions on creating and modifying:

- SQL/XML queries in Tree view, see "Creating and Modifying SQL/XML Queries in Tree View" on page 80
- jXTransformer queries in Tree view, see "Creating and Modifying jXTransformer Queries in Tree View" on page 96
- SQL/XML queries and jXTransformer queries and write statements in Text view, see "Creating and Modifying SQL/XML and jXTransformer Statements in Text View" on page 114

When you are ready to save your Builder project, select **File / Save Project As**. Specify the filename of the project, and click **Save**.

Opening a Builder Project

From the main menu, select **File** / **Open Project**. A dialog box allows you to browse and select a project to open. Select the Builder project you want to open, and click **OK**.

NOTE: Remember that Builder projects for SQL/XML queries have a file extension of .cfb; Builder projects for jXTransformer queries and jXTransformer write statements have a file extension of .jxb.

A SQL/XML query or jXTransformer query appears in the Tree view or the Text view of the Builder, depending on which view was active when the project was last saved. A jXTransformer write statement appears in Text view only.

Closing a Builder Project

When you are ready to close a Builder project, select **File** / **Close Project**. You are prompted to confirm whether you want to save the project before it is closed. Click **OK** to save the project. When this project is opened again in the Builder, it will appear in the view that was active when the project was last saved.

Creating and Modifying SQL/XML Queries in Tree View

This section describes how to create and modify SQL/XML queries in Tree view and provides instructions on performing the following tasks:

- "Creating Base SQL Query Nodes" on page 80
- "Creating XML Elements" on page 82
- "Creating XML Attributes" on page 84
- "Assigning Select Expression Values to ELEMENT Nodes" on page 87
- "Assigning Constant Values to ELEMENT Nodes" on page 88
- "Creating a Forest of XML Elements" on page 89
- "Modifying Nodes" on page 94
- "Moving Nodes" on page 94
- "Deleting Nodes" on page 95

NOTE: Builder menu commands are position-sensitive. When creating Builder project tree nodes, the resulting hierarchy depends on the node you have selected when you perform the menu command.

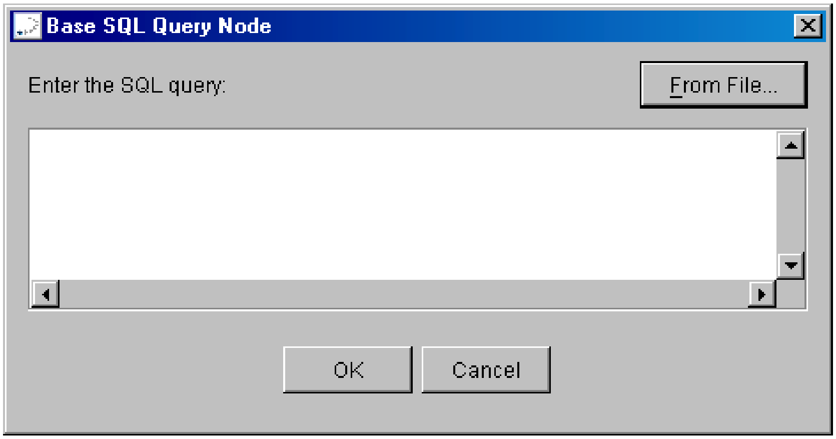
See "Creating and Modifying SQL/XML and jXTransformer Statements in Text View" on page 114 for instructions on creating and modifying queries and statements in Text view of the Builder.

Creating Base SQL Query Nodes

A Base SQL Query node contains the SQL query on which the SQL/XML query is based. A Base SQL Query node is required for each SQL/XML sub-query within a Builder project. See "Using Base SQL Query Nodes" on page 71 for information about using Base SQL Query nodes.

To create a Base SQL Query node:

1 Right-click the project node, and select **Insert / Base SQL Query node**. The Base SQL Query Node dialog box appears.



2 Using a simple Select statement, type a SQL query or click the **From File** button to navigate to a text file that contains a SQL query.

3 Click **OK**.

The Builder checks the SQL query for syntax. If an error is detected, a dialog box appears describing the error. If no errors are detected, a Base SQL Query node appears in the Builder project tree.

Once a Base SQL Query node is created, you can construct the SQL/XML query based on the Base SQL Query node, creating other types of nodes for XML elements, attributes, and so on.

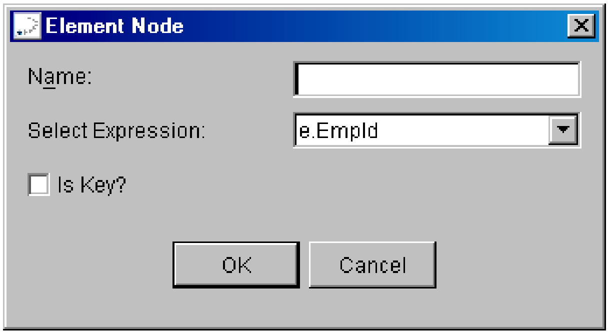
Creating XML Elements

You specify XML elements in the Tree view of the Builder using ELEMENT nodes. Because ELEMENT nodes can have multiple values and mixed content, values of ELEMENT nodes are shown as child nodes. You can create an ELEMENT node with any of the following values:

- A Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- A constant, which can be any character string literal (for example, a literal such as 8 or TRUE).
- Empty (no value). Use this value if you want the node to contain only XML subelements such as other XML elements or a nested SQL/XML query.

Select Expression

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / as Select Expression**. The Element Node dialog box appears.


Element Node

Name:

Select Expression: e.EmpId

□ Is Key?

2 In the Name field, type the name of the XML element.

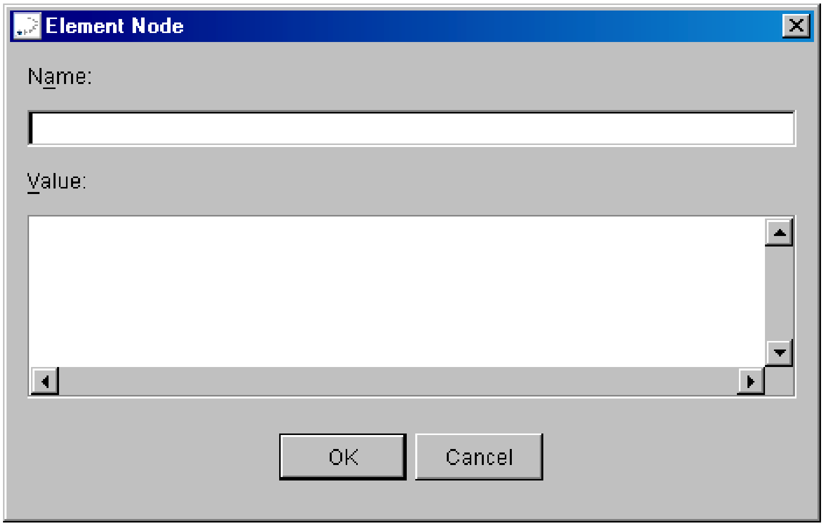
3 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

4 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

5 Click **OK**. The ELEMENT node appears in the project tree with a Select expression node as its child.

Constant

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / as Constant**. The Element Node dialog box appears.



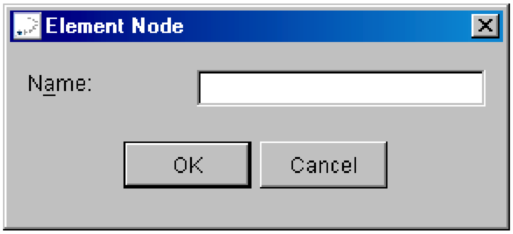
2 In the Name field, type the name of the XML element.

3 In the Value field, type a constant value. This can be any character string literal.

4 Click **OK**. The ELEMENT node appears in the project tree with a CONSTANT node as its child.

Empty

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / Empty**. The Element Node dialog box appears.



2 In the Name field, type the name of the XML element.

3 Click **OK**. The ELEMENT node appears in the project tree.

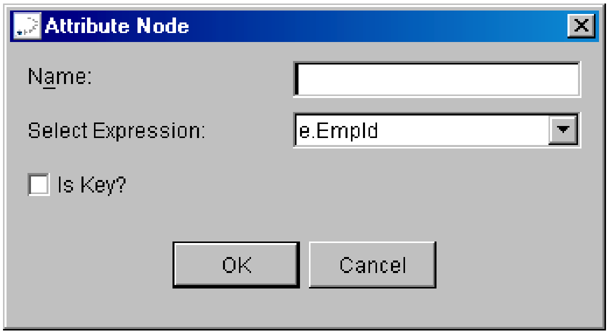
Creating XML Attributes

You specify XML attributes in the Tree View of the Builder using ATTRIBUTE nodes. You can create an ATTRIBUTE node with any of the following values:

- A Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- A constant, which can be any character string literal (for example, a literal such as 8 or TRUE).

Select Expression

1 Right-click an ELEMENT node, and select **Insert / Attribute Node / as Select Expression**. The Attribute Node dialog box appears.



Attribute Node

Name:

Select Expression: e.EmpId

□ Is Key?

2 In the Name field, type the name of the XML attribute.

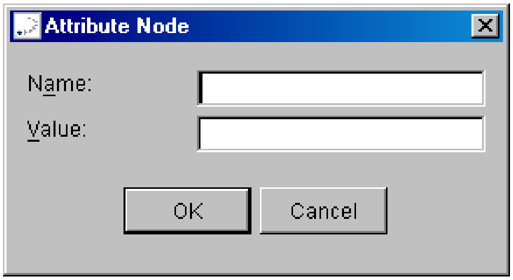
3 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

4 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

5 Click **OK**. The ATTRIBUTE node appears in the project tree.

Constant

1 Right-click an ELEMENT node and select **Insert / Attribute Node / as Constant**. The Attribute Node dialog box appears.



2 In the Name field, type the name of the XML attribute.

3 In the Value field, type a constant value. This can be any character string literal.

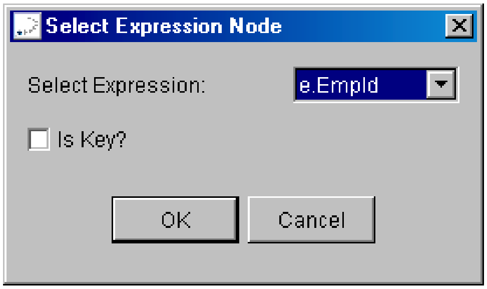
4 Click **OK**. The ATTRIBUTE node appears in the project tree.

Assigning Select Expression Values to ELEMENT Nodes

Because XML elements can have multiple values and mixed content, you can assign a Select expression value to an empty ELEMENT node.

To assign a Select expression value to an ELEMENT node:

1 Select **Insert / Select Expression Node**. The Select Expression Node dialog box appears.



Select Expression: e.EmpId

□ Is Key?

2 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

3 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

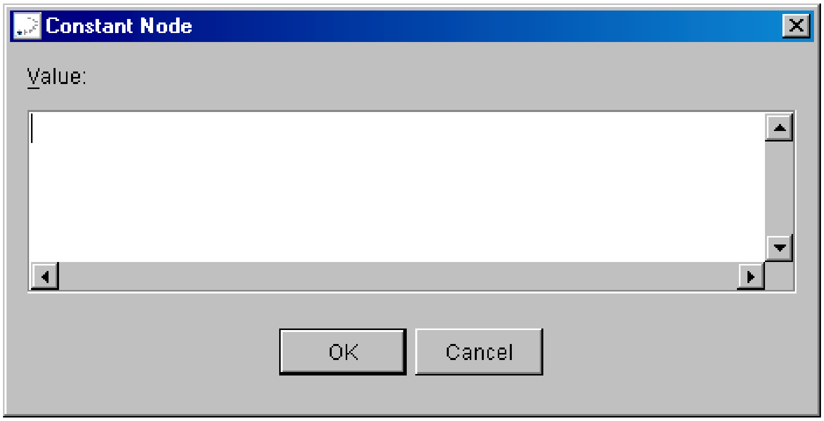
4 Click **OK**. The SELECT EXPR node appears in the project tree as a child of the selected ELEMENT node.

Assigning Constant Values to ELEMENT Nodes

Because XML elements can have multiple values and mixed content, you can assign a constant value to an empty ELEMENT node.

To assign a constant value to an ELEMENT node:

1 Select **Insert / Constant Node**. The Constant Node dialog box appears.



2 In the Value field, type a constant value. This can be any character string literal.

3 Click **OK**. The CONSTANT node appears in the project tree as a child of the selected ELEMENT node.

Creating a Forest of XML Elements

There are multiple ways to create a forest of XML elements, which is a collection of XML elements. The Builder allows you to create a forest of XML elements in Tree view using the following types of nodes:

- FOREST nodes produce a forest of XML elements from a given list of arguments. A FOREST node is parent to one or multiple FOREST ELEMENT nodes. A FOREST ELEMENT node corresponds to a single argument in the list of arguments.
- CONCAT nodes produce a forest of XML elements by concatenating a list of XML values. The CONCAT node is parent to one or multiple of the following nodes that compose the list of XML values:
  - ELEMENT nodes
  - FOREST nodes
  - Other CONCAT nodes
  - AGG nodes
- AGG nodes produce a forest of XML elements from a collection of XML elements. An AGG node is parent to only one of the following nodes:
  - ELEMENT nodes
  - FOREST nodes
  - CONCAT nodes
  - AGG nodes

XML Forest

A FOREST node produces a forest of XML elements from a given list of arguments. To create a FOREST node, select **Insert / XML Forest Node**. The FOREST node appears in the project tree. Add one or more Forest Elements as described in the next section.

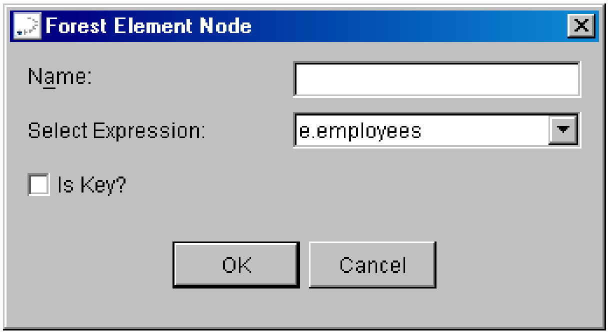
Forest Elements

You can create an FOREST ELEMENT node with any of the following values:

- A Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- A constant, which can be any character string literal (for example, a literal such as 8 or TRUE).
- Empty (no value). Use this value if you want the node to contain only XML subelements such as other XML elements or a nested SQL/XML query.

Select Expression

1 Right-click a FOREST node, and select **Insert** / **Forest Element Node / as Select Expression**. The Forest Element Node dialog box appears.



Forest Element Node

Name:

Select Expression: e.employees

□ Is Key?

2 In the Name field, type the name of the XML forest element.

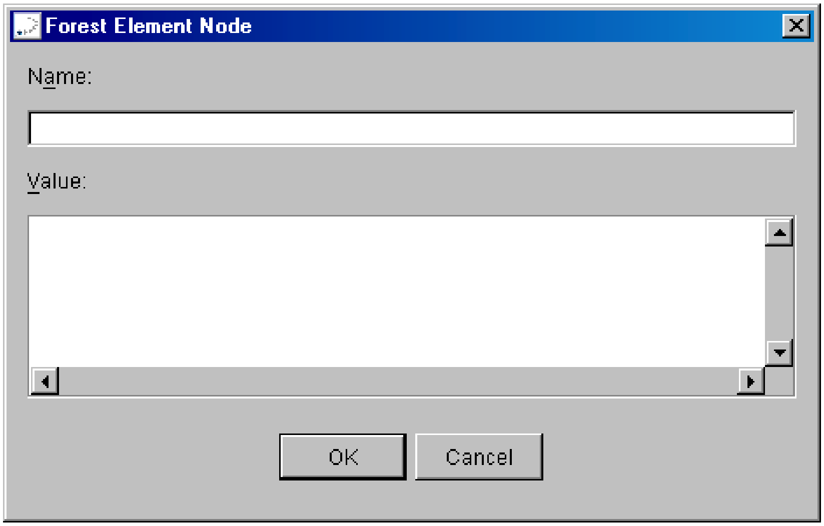
3 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

4 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

5 Click **OK**. The FOREST ELEMENT node appears in the project tree with a Select expression node as its child.

Constant

1 Right-click a FOREST node, and select **Insert / Forest Element Node / as Constant**. The Forest Element Node dialog box appears.



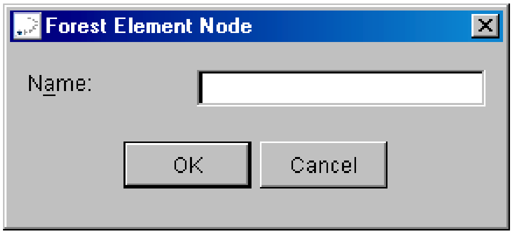
2 In the Name field, type the name of the XML forest element.

3 In the Value field, type a constant value. This can be any character string literal.

4 Click **OK**. The FOREST ELEMENT node appears in the project tree with a CONSTANT node as its child.

Empty

1 Right-click a FOREST node, and select **Insert / Forest Element Node / Empty**. The Forest Element Node dialog box appears.


Forest Element Node

Name:

2 In the Name field, type the name of the XML forest element.

3 Click **OK**. The FOREST ELEMENT node appears in the project tree.

XML Concatenation

CONCAT nodes produce a forest of XML elements by concatenating a list of XML values.

To create a forest of XML elements using XML concatenation:

1 Select **Insert / XML Concatenation Node**. The CONCAT node appears in the project tree.

2 Select the CONCAT node and add one or multiple nodes of the following types to specify the list of XML values:

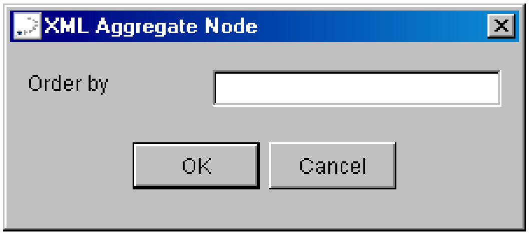
- ELEMENT nodes as described in "Creating XML Elements" on page 82
- FOREST nodes as described in "XML Forest" on page 89
- CONCAT nodes as described in this procedure
- AGG nodes as describe in "XML Aggregate" on page 93

XML Aggregate

AGG nodes produce a forest of XML elements from a collection of XML elements that is specified using a single XML value.

To create a forest of XML elements using XML aggregate:

1 Select **Insert / XML Aggregate Node**. The XML Aggregate Node dialog box appears allowing you to optionally specify an Order by condition.



2 If you want to sort your results, in the Order by field, type a value by which you want the returned result set to be sorted. You can specify any alternative for a sort list as defined in the document *ISO/IEC 9075-2*.

If you do not want to sort your results, leave the Order by field empty.

3 Click **OK**. The AGG node appears in the project tree.

4 Select the AGG node and add one node of the following node types to specify a collection of XML elements:

- ELEMENT nodes as described in "Creating XML Elements" on page 82
- FOREST nodes as described in "XML Forest" on page 89
- CONCAT nodes as described in "XML Concatenation" on page 92
- AGG nodes as described in this procedure

Modifying Nodes

Right-click the node you want to modify, and select **Edit**. Change the information in the dialog box as necessary. When you are satisfied with your changes, click **OK**.

NOTES:

- You cannot modify a project node.
- Changes to a validated Base SQL Query node may invalidate nodes in the project tree.
- For ELEMENT nodes, you can only change the name of the node because ELEMENT nodes can contain multiple value nodes.

Moving Nodes

Select the node you want to move. Then, drag and drop the node to the location you want it to appear in the project tree.

NOTES:

- You cannot move the project node.
- The Builder restricts moves to valid choices as defined by the SQL/XML syntax.

Deleting Nodes

Right-click the node you want to delete and select **Delete**. If the selected node has children, a window appears asking you to confirm the deletion of the node and all its children. To confirm, click **OK**.

NOTES:

- When you delete a parent node with children nodes, the children nodes are deleted also.
- You cannot delete the project node.

Creating and Modifying jXTransformer Queries in Tree View

This section describes how to create and modify jXTransformer queries in Tree view and provides instructions on performing the following tasks:

- "Turning on the Document Header" on page 97
- "Specifying Comments" on page 97
- "Specifying Processing Instructions" on page 98
- "Specifying an External DTD" on page 100
- "Creating Base SQL Query Nodes" on page 80
- "Creating XML Elements" on page 82
- "Creating XML Attributes" on page 84
- "Specifying Hide Information" on page 107
- "Creating XML CDATA Sections" on page 108
- "Specifying XML Namespaces" on page 110
- "Assigning Select Expression Values to ELEMENT Nodes" on page 111
- "Assigning Constant Values to ELEMENT Nodes" on page 112
- "Modifying Nodes" on page 112
- "Moving Nodes" on page 113
- "Deleting Nodes" on page 113

NOTE: Builder menu commands are position-sensitive. When creating Builder project tree nodes, the resulting hierarchy depends on the node you have selected when you perform the menu command.

See "Creating and Modifying SQL/XML and jXTransformer Statements in Text View" on page 114 for instructions on creating and modifying queries in Text view of the Builder.

Turning on the Document Header

To generate a complete XML document instead of an XML document fragment from a jXTransformer query, you must turn on the document header by selecting **Insert / Document Header**. If you do not turn on the document header, an XML document fragment is generated when the query is executed.

Turning on the document header automatically inserts a ROOT ELEMENT node in the project tree with a default root element name of jxtr-result. You can change the default root element name by editing the ROOT ELEMENT node. See "Modifying Nodes" on page 112 for instructions. Only one XML root element is allowed in each XML document.

NOTE: When you turn off the document header, the ROOT ELEMENT node, and any COMMENT, PROCESSING INSTRUCTION, and EXTERNAL DTD nodes are deleted.

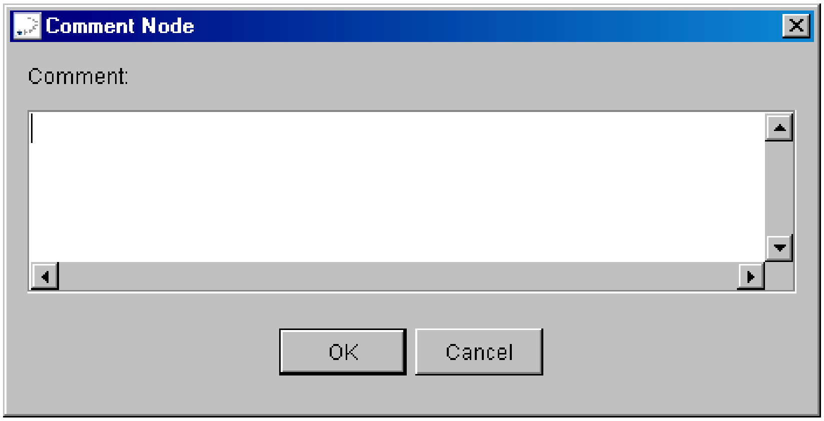
Specifying Comments

You can specify one or multiple comments in the document header only. Comments are specified in the Tree view of the Builder by using COMMENT nodes, with the project node as the parent. To specify multiple comments, create multiple COMMENT nodes, one for each comment.

NOTE: To create a COMMENT node, you first must turn on the document header. See "Turning on the Document Header" on page 97 for instructions.

To specify a comment:

1 Select the project node, and select **Insert / Comment**. The Comment Node dialog box appears.



2 In the Comment field, type a comment.

3 Click **OK**. A COMMENT node with the specified comment appears in the project tree.

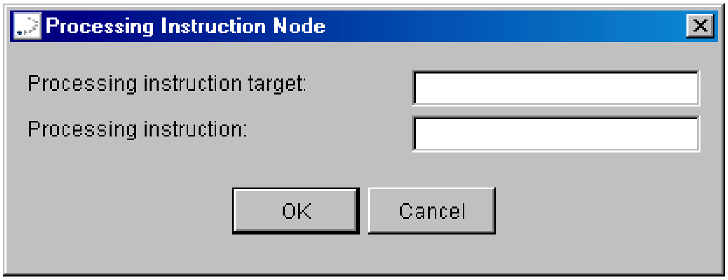
Specifying Processing Instructions

You can create one or multiple processing instructions in the resulting XML document. For example, if the generated XML document will be viewed in a browser, you may want to specify a processing instruction so that a specific XSL stylesheet is used when the XML document is viewed. Processing instructions are specified in the Tree view of the Builder using PROCESSING INSTRUCTION nodes, with the project node as the parent. To specify multiple processing instructions, create multiple PROCESSING INSTRUCTION nodes, one node for each processing instruction.

NOTE: To create a PROCESSING INSTRUCTION node, you first must turn on the document header. See "Turning on the Document Header" on page 97 for instructions.

To specify a processing instruction:

1 Select the project node, and select **Insert / Processing Instruction**. The Processing Instruction Node dialog box appears.



2 In the Processing instruction target field, type the target of the processing instruction. For example, if you are specifying a processing instruction for an XSL stylesheet, you may want to type `xsl-stylesheet`.

3 In the Processing instruction field, type a valid XML processing instruction. For example, if you are specifying a processing instruction for an XSL stylesheet named common.xsl, you would type:

```
type="text/xsl" href="file://common.xsl"
```

4 Click **OK**. A PROCESSING INSTRUCTION node with the specified processing instruction appears in the project tree.

Specifying an External DTD

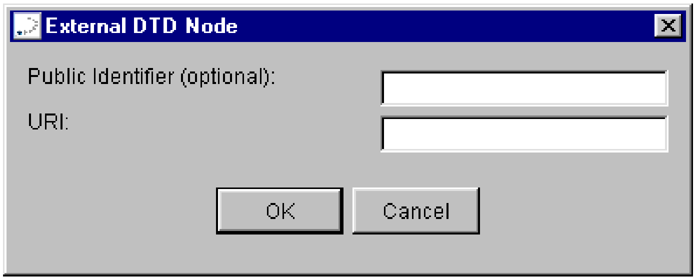
You can specify an external private or public DTD in the document header. External DTDs are specified in the Tree view of the Builder using EXTERNAL DTD nodes, with the project node as the parent.

NOTE: To create an EXTERNAL DTD node, you first must turn on the document header. See "Turning on the Document Header" on page 97 for instructions.

To view a DTD, you can open the DTD in the Open DTD window. See "Opening an XML DTD" on page 129 for instructions.

To specify an external DTD:

1 Select the project node, and select **Insert / External DTD**. The External DTD Node dialog box appears.



2 If specifying a public external DTD, in the Public Identifier field, type the public identifier of the public external DTD. Do not specify anything in this field if specifying a private external DTD.

3 In the URI field, type the Uniform Resource Identifier that identifies the external DTD.

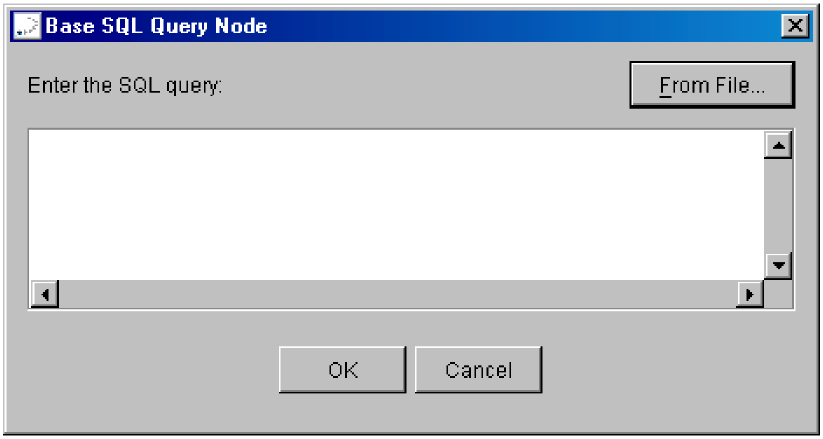
4 Click **OK**. An EXTERNAL DTD node appears in the project tree.

Creating Base SQL Query Nodes

A Base SQL Query node contains the SQL query on which the jXTransformer query is based. A Base SQL Query node is required for each jXTransformer sub-query within a Builder project. See "Using Base SQL Query Nodes" on page 71 for information about using Base SQL Query nodes.

To create a Base SQL Query node:

1 Right-click the project node or a ROOT ELEMENT node, and select **Insert / Base SQL Query node**. The Base SQL Query Node dialog box appears.



2 Using a simple Select statement, type a SQL query or click the **From File** button to navigate to a text file that contains a SQL query.

3 Click **OK**.

The Builder checks the SQL query for syntax. If an error is detected, a dialog box appears describing the error. If no errors are detected, a Base SQL Query node appears in the project tree.

Once a Base SQL Query node is created, you can construct the jXTransformer query based on the Base SQL Query node, creating other types of nodes for jXTransformer elements, attributes, and so on.

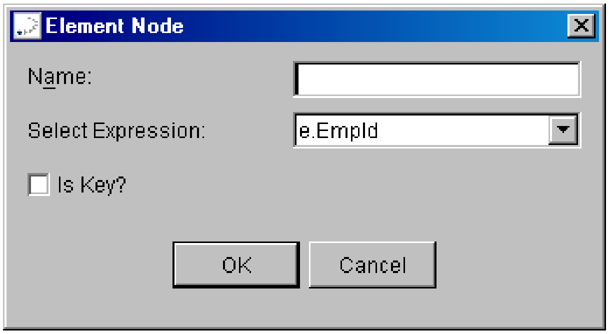
Creating XML Elements

You specify XML elements in the Tree view of the Builder using ELEMENT nodes. Because ELEMENT nodes can have multiple values and mixed content, values of ELEMENT nodes are shown as child nodes. You can create an ELEMENT node with any of the following values:

- A Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- A constant, which can be any character string literal (for example, a literal such as 8 or TRUE).
- Empty (no value). Use this value if you want the node to contain only XML subelements such as other XML elements, mixed content, or a nested jXTransformer query.

Select Expression

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / as Select Expression**. The Element Node dialog box appears.



Element Node

Name:

Select Expression: e.EmpId

☐ Is Key?

2 In the Name field, type the name of the XML element.

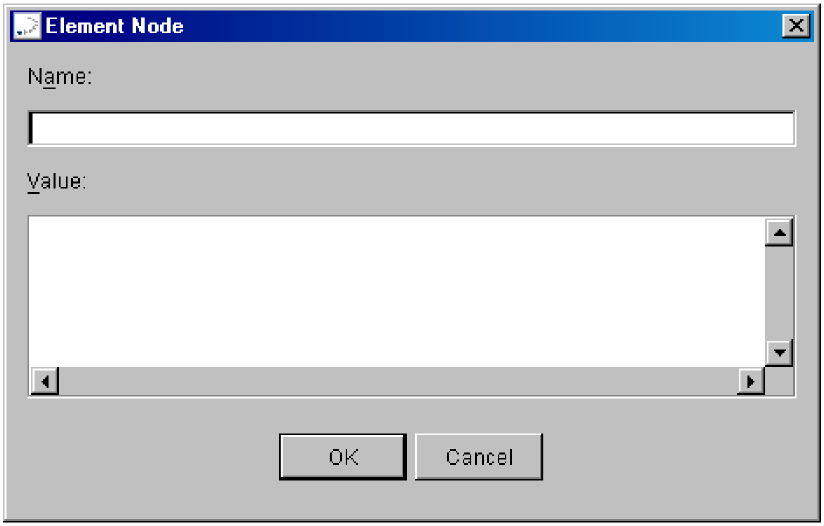
3 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

4 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

5 Click **OK**. The ELEMENT node appears in the project tree with a Select expression node as its child.

Constant

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / as Constant**. The Element Node dialog box appears.



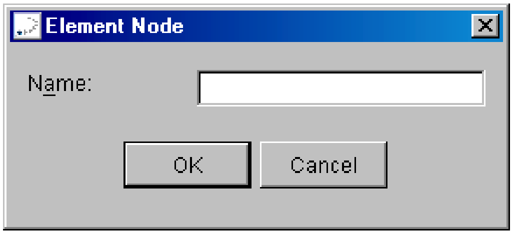
2 In the Name field, type the name of the XML element.

3 In the Value field, type a constant value. This can be any character string literal.

4 Click **OK**. The ELEMENT node appears in the project tree with a CONSTANT node as its child.

Empty

1 Right-click a Base SQL Query node or an ELEMENT node, and select **Insert / Element Node / Empty**. The Element Node dialog box appears.



2 In the Name field, type the name of the XML element.

3 Click **OK**. The ELEMENT node appears in the project tree.

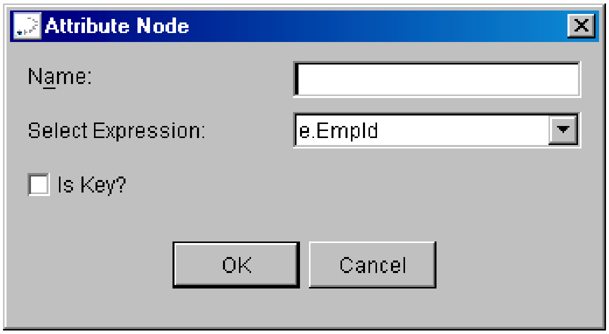
Creating XML Attributes

You specify XML attributes in the Tree View of the Builder using ATTRIBUTE nodes. You can create an ATTRIBUTE node with any of the following values:

- A Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- A constant, which can be any character string literal (for example, a literal such as 8 or TRUE).
- A SQL query (typically used to construct IDREF values for an XML attribute). This query must select only one table column.

Select Expression

1 Right-click an ELEMENT node, and select **Insert / Attribute Node / as Select Expression**. The Attribute Node dialog box appears.


Attribute Node

Name:

Select Expression: e.EmpId

□ Is Key?

2 In the Name field, type the name of the XML attribute.

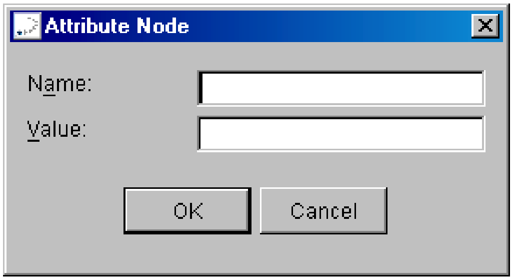
3 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

4 Select the **Is Key?** checkbox if the Select expression associated with this node is the key or part of a multi-value key in the database. Keys uniquely identify each row selected by the base SQL query.

5 Click **OK**. The ATTRIBUTE node appears in the project tree.

Constant

1 Right-click an ELEMENT node and select **Insert / Attribute Node / as Constant**. The Attribute Node dialog box appears.



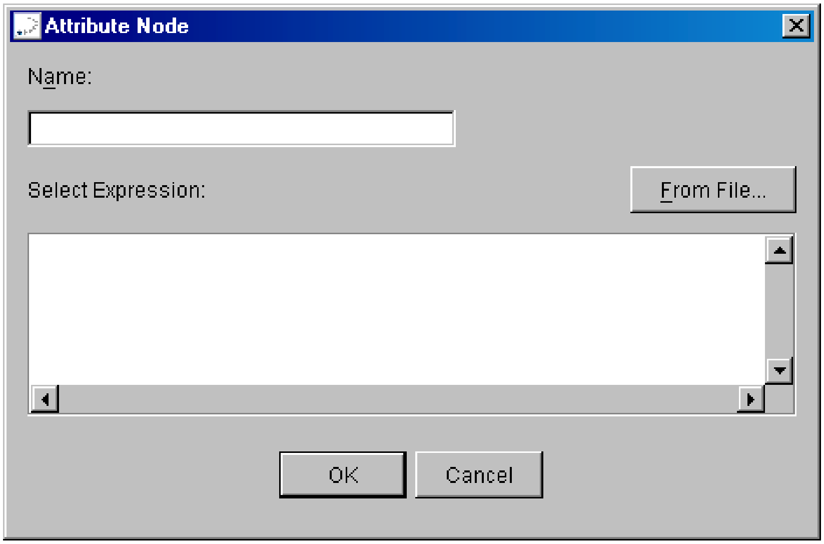
2 In the Name field, type the name of the XML attribute.

3 In the Value field, type a constant value. This can be any character string literal.

4 Click **OK**. The ATTRIBUTE node appears in the project tree.

SQL Query

1 Right-click an ELEMENT node and select **Insert / Attribute Node / as SQL Query**. The Attribute Node dialog box appears.



2 In the Name field, type the name of the XML attribute.

3 From the Select Expression field, type a SQL query or click the **From File** button to navigate to a text file that contains a SQL query. This query must select only one table column.

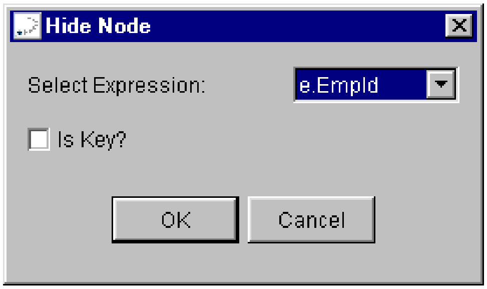
4 Click **OK**. The ATTRIBUTE node appears in the project tree.

Specifying Hide Information

You can retrieve information from the database that you do not want to include in the resulting XML document using HIDE nodes. Hide information typically is used to select database columns that are a key or part of a multi-value key that you do not want to display in the resulting XML document.

To specify hide information:

1 Right-click the Base SQL Query node or an ELEMENT node, and select **Insert / Hide Node**. The Hide Node dialog box appears.



Hide Node

Select Expression: e.EmpId

☐ Is Key?

2 From the Select Expression drop-down list box, select a Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

3 Select the **Is Key?** checkbox if the select expression associated with this node is the key or part of a multi-value key. Keys uniquely identify each row selected by the base SQL query.

4 Click **OK**. The HIDE node appears in the project tree.

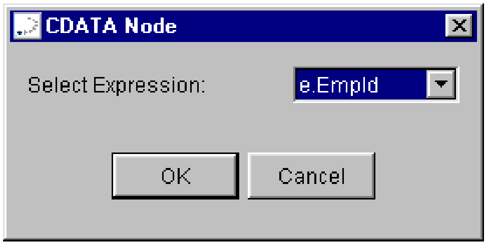
Creating XML CDATA Sections

You can create CDATA sections in the Tree view of the Builder using CDATA nodes. You can create a CDATA node with any of the following values:

- Select expression, which can be any Select expression specified in the Base SQL Query node (for example, a database column such as e.LastName).
- Constant, which can be any character string literal (for example, a literal such as 8 or TRUE).

Select Expression

1 Right-click the Base SQL Query node or an ELEMENT node, and select **Insert / CDATA Node / as Select Expression**. The CDATA Node dialog box appears.

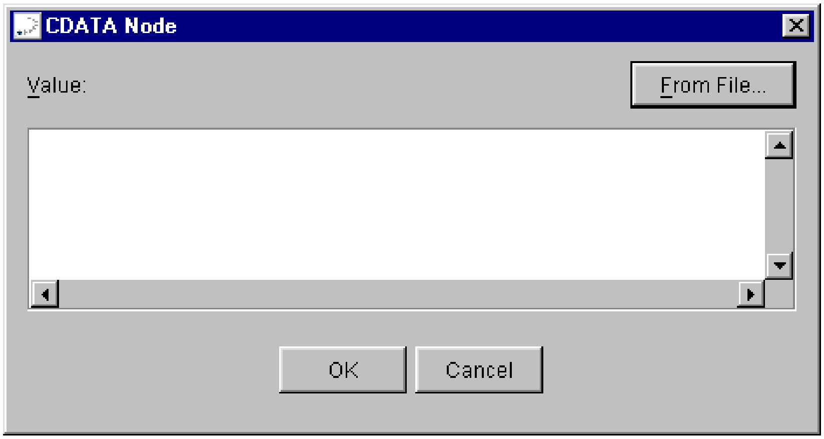


2 From the Select Expression drop-down list, select a Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

3 Click **OK**. The CDATA node appears in the project tree.

Constant

1 Right-click the Base SQL Query node or an ELEMENT node, and select **Insert / CDATA Node / as Constant**. The CDATA Node dialog box appears.



2 In the Value field, type a constant value or click the **From File** button to navigate to a file containing the constant value. This can be any character string literal.

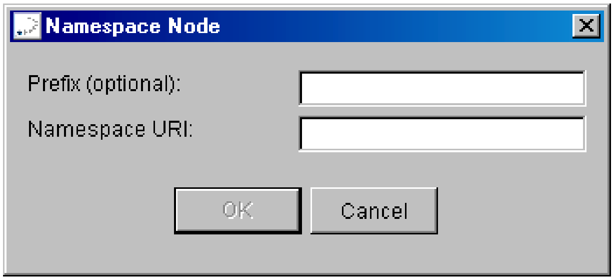
3 Click **OK**. The CDATA node appears in the project tree.

Specifying XML Namespaces

An XML namespace allows you to create a namespace definition for an XML element. You can create XML namespace definitions in the Tree view of the Builder using NAMESPACE nodes.

To specify a namespace:

1 Right-click an ELEMENT node or a ROOT ELEMENT node, and select **Insert / Namespace Node**. The Namespace Node dialog box appears.



2 (Optional) In the Prefix field, type a prefix for the namespace. If you do not specify a prefix, the default XML namespace will be defined.

3 In the Namespace URI field, type a URI to identify the namespace.

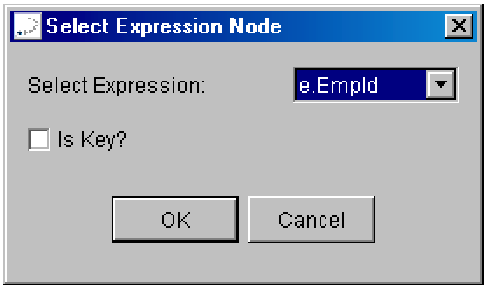
4 Click **OK**. The NAMESPACE node appears in the project tree.

Assigning Select Expression Values to ELEMENT Nodes

Because XML elements can have multiple values and mixed content, you can assign a Select expression value to an empty ELEMENT node.

To assign a Select expression value to an ELEMENT node:

1 Select **Insert / Select Expression Node**. The Select Expression Node dialog box appears.



Select Expression Node

Select Expression: e.EmpId

☐ Is Key?

2 From the Select Expression drop-down list, select an available Select expression. This list is populated with the Select expressions specified in the Base SQL Query node.

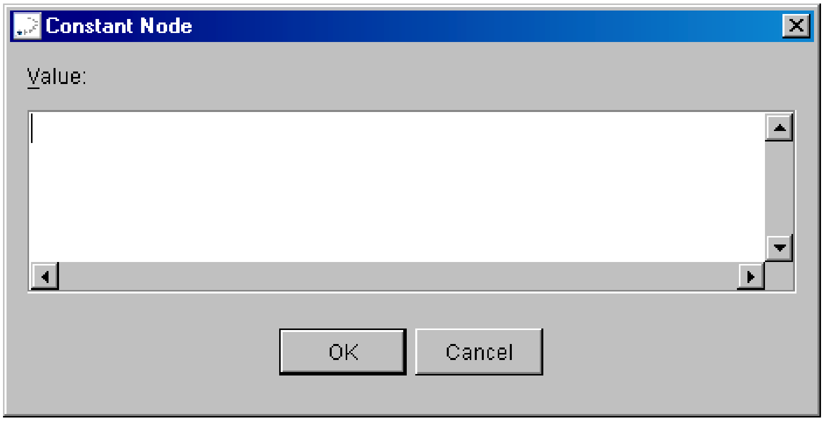
3 Click **OK**. The SELECT EXPR node appears in the project tree as a child of the selected ELEMENT node.

Assigning Constant Values to ELEMENT Nodes

Because XML elements can have multiple values and mixed content, you can assign a constant value to an empty ELEMENT node.

To assign a constant value to an ELEMENT node:

1 Select **Insert / Constant Node**. The Constant Node dialog box appears.



2 In the Value field, type a constant value. This can be any character string literal.

3 Click **OK**. The CONSTANT node appears in the project tree as a child of the selected ELEMENT node.

Modifying Nodes

Right-click the node you want to modify, and select **Edit**. Change the information in the dialog box as necessary. When you are satisfied with your changes, click **OK**.

NOTES:

- You cannot modify a project node.
- Changes to a validated Base SQL Query node may invalidate nodes in the project tree.
- For ELEMENT nodes, you can only change the name of the node because ELEMENT nodes can contain multiple value nodes.

Moving Nodes

Select the node you want to move. Then, drag and drop the node to the location you want it to appear in the project tree.

NOTES:

- You cannot move the project node.
- The Builder restricts moves to valid choices as defined by the jXTransformer query syntax.

Deleting Nodes

Right-click the node you want to delete and select **Delete**. If the selected node has children, a window appears asking you to confirm the deletion of the node and all its children. To confirm, click **OK**.

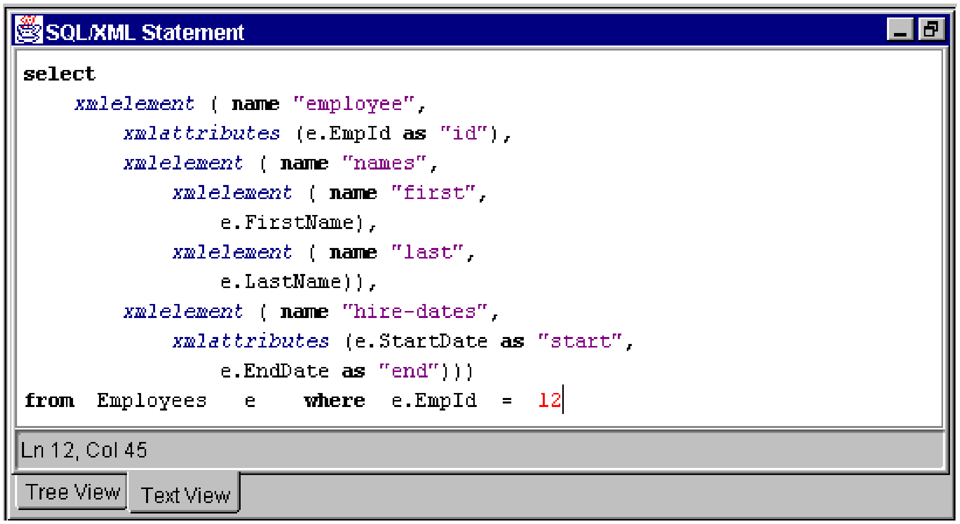
NOTES:

- When you delete a parent node with children nodes, the children nodes are deleted also.
- You cannot delete the project node.

Creating and Modifying SQL/XML and jXTransformer Statements in Text View

To create and modify SQL/XML and jXTransformer statements in Text view, you must be familiar with their respective syntax. Using valid syntax, you can type the statement in the Text view of the Builder. Figure 3-4 shows a SQL/XML query in the Text view of the SQL/XML Statement window.

Figure 3-4. SQL/XML Query in Text View



```
select
    xmlelement ( name "employee",
        xmlattributes (e.EmpId as "id"),
        xmlelement ( name "names",
            xmlelement ( name "first",
                e.FirstName),
            xmlelement ( name "last",
                e.LastName)),
        xmlelement ( name "hire-dates",
            xmlattributes (e.StartDate as "start",
                e.EndDate as "end")))
from  Employees   e    where  e.EmpId  =  12
```

You can also copy or import an existing SQL/XML or jXTransformer query or statement into Text view of the Builder and modify them as needed. See “Importing a Query or Statement” on page 115 for instructions.

For more information about:

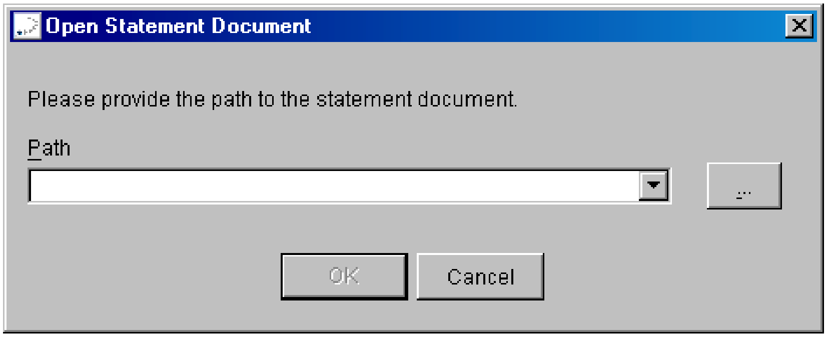
- SQL/XML query syntax, see Chapter 4, "Syntax of SQL/XML Queries" on page 141
- jXTransformer query syntax, see Chapter 5, "Syntax of jXTransformer Queries" on page 153
- jXTransformer write statement syntax, see Chapter 6, "Syntax of jXTransformer Write Statements" on page 173

Importing a Query or Statement

You can only import a query or statement into Text view of the Builder.

To import a query or statement:

1 Select **File / Import Statement**. The Open Statement Document dialog box appears allowing you to specify a path to an existing query or statement.



Open Statement Document

Please provide the path to the statement document.

Path

2 In the Path field, type the path to the file containing the query or statement, or click the **Browse (...)** button to navigate to the file.

3 Click **OK**. The contents of the specified file appear in the SQL/XML Statement window or jXTransformer Statement window.

Checking Query or Statement Syntax

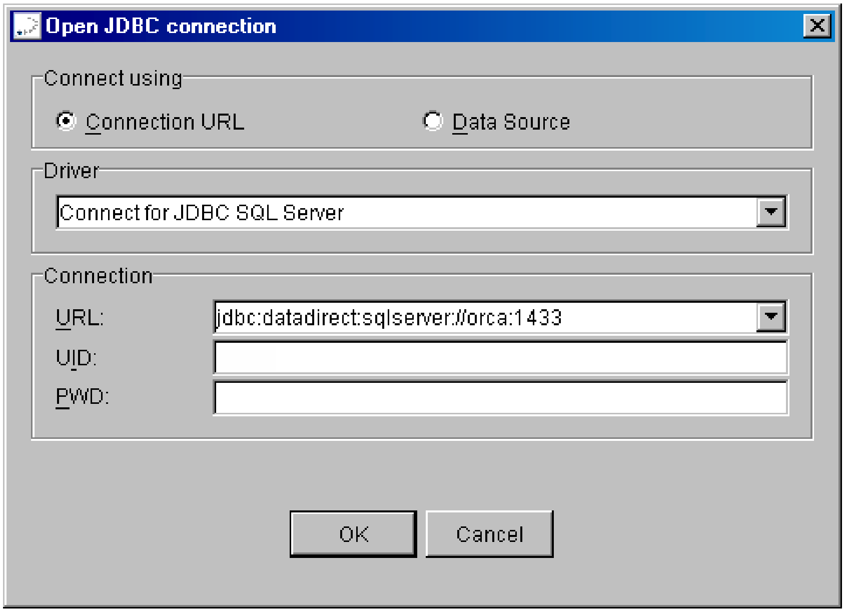
When you switch between the Tree and Text views, the Builder automatically checks the syntax of your query or statement. If an error is encountered, a message is generated. You can also check the syntax of a query or statement by selecting **Project** / **Check Statement Syntax**.

Connecting to the Database

To browse the database or execute a query or statement, a JDBC connection to the database using a DataDirect Technologies JDBC driver is required. The JDBC connection can be accomplished using a JDBC connection URL or JDBC data source.

To connect to the database:

Select **Project / Connect to Database**. The Open JDBC Connection dialog box appears.



Open JDBC connection

Connect using

● Connection URL ○ Data Source

Driver

Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

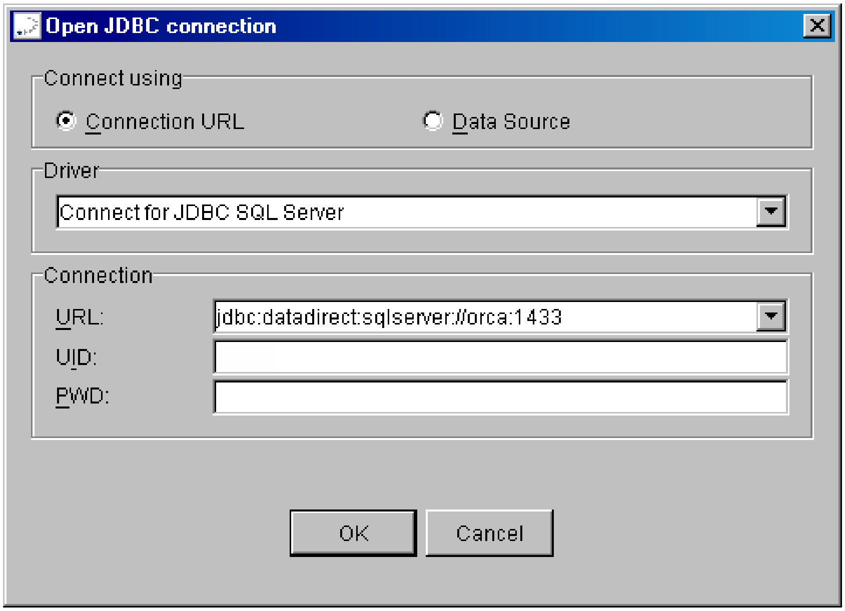
PWD:

The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, continue with "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, continue with "Connecting Using JDBC Data Sources" on page 119

Connecting Using JDBC Connection URLs

When you connect to the database, the Open JDBC Connection dialog box appears with the Connection URL option selected by default.



Open JDBC connection

Connect using

● Connection URL ○ Data Source

Driver: Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

PWD:

To connect using a JDBC connection URL:

1. In the Driver group, type or select the driver you want to use for the connection from the drop-down list. You can enter any driver class specified in your classpath or enter any named driver that you have explicitly configured.

2. In the Connection group, complete the following information:

   URL: Select the URL you want to use for the connection from the drop-down list or type a URL.

   UID: Type a user name for the database.

   PWD: Type a password for the database.

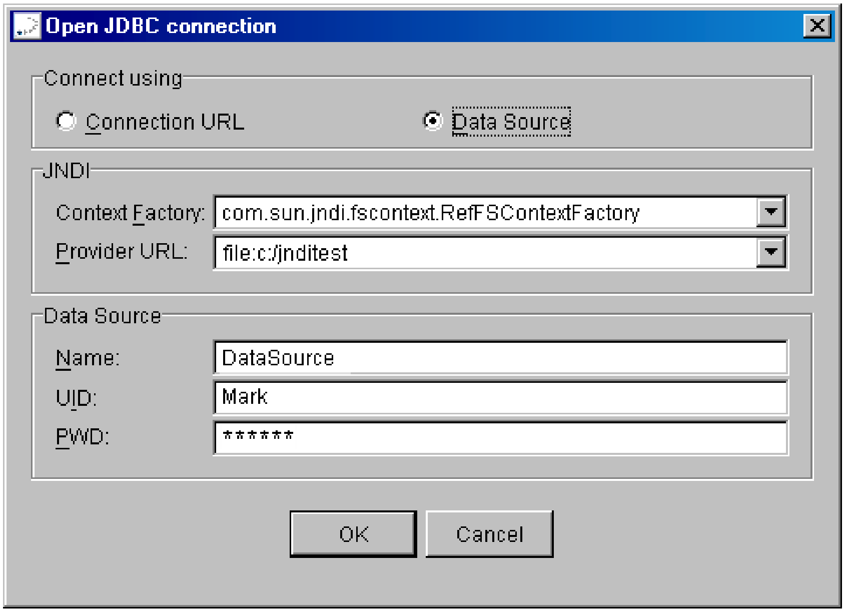
3 Click **OK**. When you are connected, a message confirming the connection appears in the Output window.

Connecting Using JDBC Data Sources

NOTE: To use JDBC data sources with the Builder if you are not using JDK 1.4 or higher, you must install the JNDI service providers used by your data sources and you must install the following Java packages and include them in your classpath:

- JDBC 2.0 Optional Package
- Java Naming and Directory Interface Package

To connect to the database using a JDBC data source, select the data source option on the Open JDBC Connection dialog box. The fields on the dialog box change to accommodate information required for a data source.



Open JDBC connection

Connect using

○ Connection URL ● Data Source

JNDI

Context Factory: com.sun.jndi.fscontext.RefFSContextFactory

Provider URL: file:c:/jnditest

Data Source

Name: DataSource

UID: Mark

PWD: ******

To connect using a JDBC data source:

1. In the JNDI group, complete the following information:

   Context Factory: Type the name of the JNDI context factory to be used, or select the name from the drop-down list.

   Provider URL: Type the URL that locates the JNDI provider to be used, or select a URL from the drop-down list.

2. In the Data Source group, complete the following information:

   Name: Type the name of the data source you want to use for the connection.

   UID: Type a user name for the database.

   PWD: Type a password for the database.

3. Click **OK**. When you are connected, a message confirming the connection appears in the Output window.

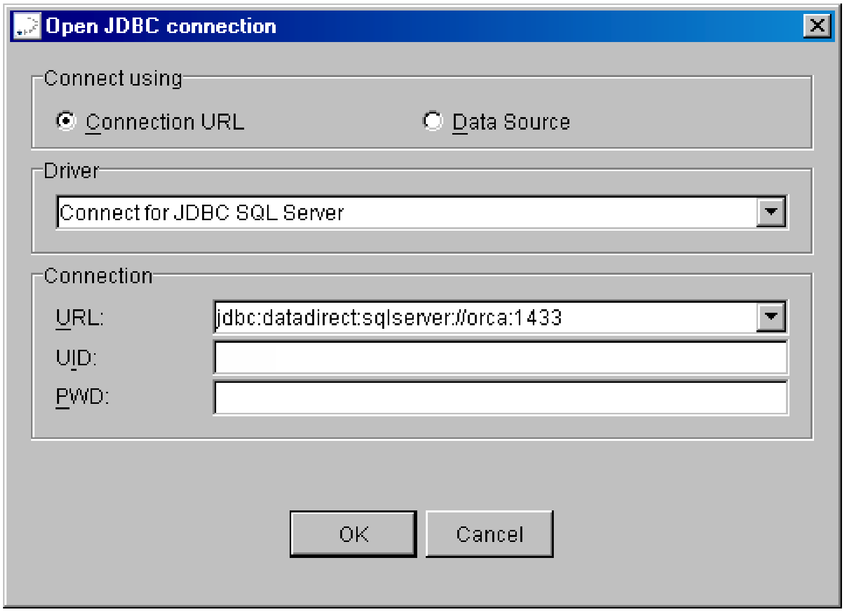
Executing a Query or Statement

Once you execute a query or statement, you can view the results in Tree view or Text view by selecting the appropriate tab at the bottom of the results window. The type of results that are returned depend on the type of query or statement being executed:

- A SQL/XML query returns a result set, where one or multiple columns in the result set can contain XML type values. The Builder displays these result sets by concatenating all columns or rows after converting the values to strings.
- A jXTransformer query returns an XML document or XML document fragment.
- A jXTransformer write statement returns a list of update counts.

To execute a query or statement:

1 Select **Project / Execute Statement**. If you are not connected to the database, you first must make a JDBC connection. The Open JDBC Connection dialog box appears.



Driver: Connect for JDBC SQL Server

URL: jdbc:datadirect:sqlserver://orca:1433

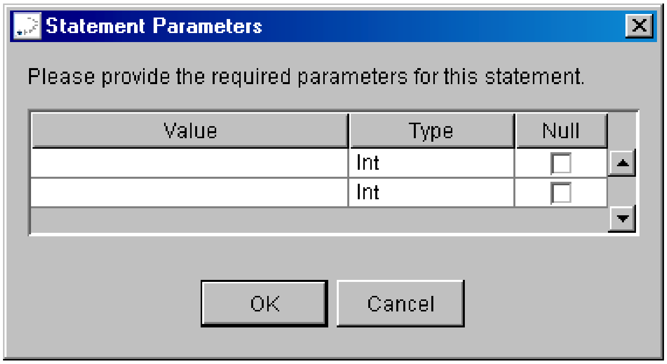
UID:

PWD:

The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

2 Parameter markers are placeholders for values, represented by question marks (?). If a query contains parameter markers, the Statement Parameters dialog box appears, prompting you for the parameter marker values.

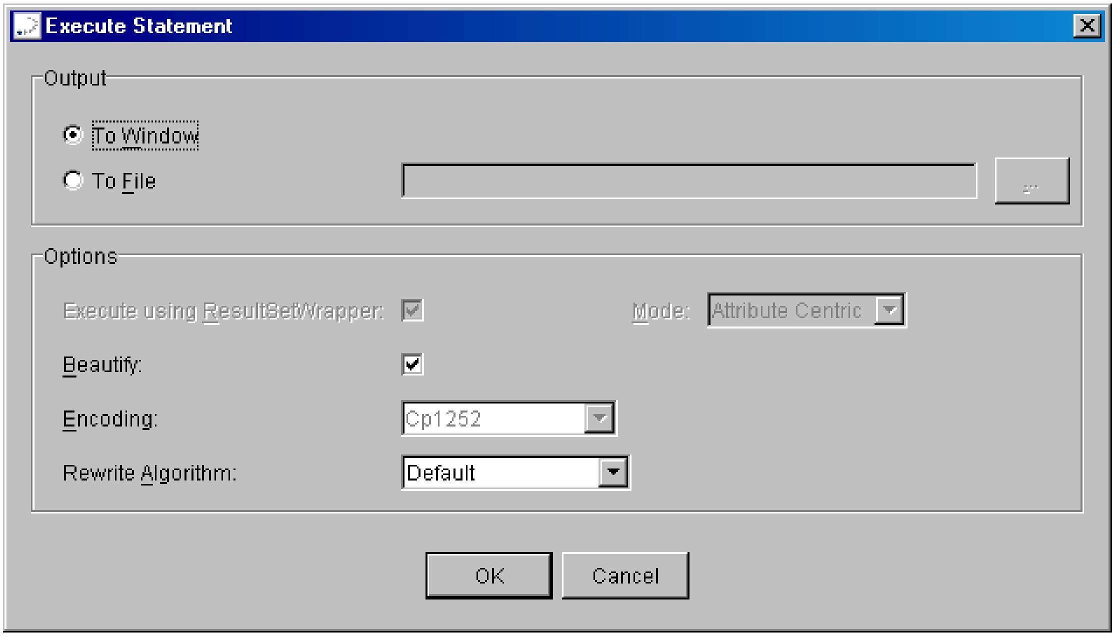

Statement Parameters

Please provide the required parameters for this statement.

Value	Type	Null
	Int	□
	Int	□

In the Value field, type the value of the parameter marker or select the Null checkbox to set the parameter marker to a null value. Then, click **OK**.

The Execute Statement dialog box appears.


Execute Statement

Output

● To Window

○ To File

Options

Execute using ResultSetWrapper: ☑

Mode: Attribute Centric

Beautify: ☑

Encoding: Cp1252

Rewrite Algorithm: Default

3 In the Output group, select one of the following options:

- **To Window** causes the results to display in a separate window. You can save the results to a file from this window by selecting **File / Save as**.

- **To File** causes the results to be saved directly to a file instead of displaying it in the Builder. Type the path and name of the file in the associated field, or click the **Browse (...)** button to browse to a file.

4 In the Options group, complete the following information:

Execute using ResultSetWrapper: This option is enabled only for jXTransformer query projects that execute a SQL query; it is disabled for all SQL/XML queries. Check this option to execute the SQL query using an attribute-centric or element-centric formula.

Mode: From the drop-down list, select **attribute-centric** or **element-centric** to choose a document structure for the SQL query.

See "Choosing an XML Document Structure" on page 53 for information about attribute-centric and element-centric document structures.

Beautify: This option is turned on by default. The Beautify option formats the content of XML with standard indents and line breaks. Uncheck the **Beautify** checkbox to turn off this option. The content of XML will be formatted without indents and line breaks.

Encoding: Select the type of encoding to use for the XML. The default depends on your platform.

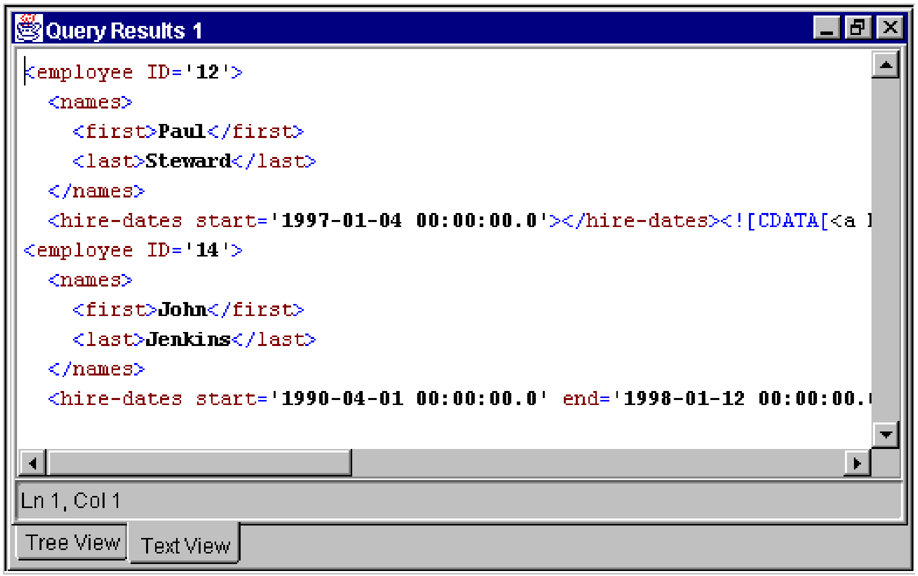
5 From the Rewrite Algorithm drop-down list, select a rewrite algorithm. In most cases, you should use the default.

For more information about choosing rewrite algorithms for:

- SQL/XML queries, see "Using rewrite_algorithm Hints" on page 41
- jXTransformer queries, see "Choosing a Rewrite Algorithm" on page 56

6 Click **OK** to execute the query or statement. If you selected the:

- **To Window** option, results appear in a separate window. Select **File** / **Save as** to save the results to a file. The following figure shows an XML document fragment, the result of a jXTransformer query.



```
<employee ID='12'>
  <names>
    <first>Paul</first>
    <last>Steward</last>
  </names>
  <hire-dates start='1997-01-04 00:00:00.0'></hire-dates><![CDATA[<a 
<employee ID='14'>
  <names>
    <first>John</first>
    <last>Jenkins</last>
  </names>
  <hire-dates start='1990-04-01 00:00:00.0' end='1998-01-12 00:00:00.
```

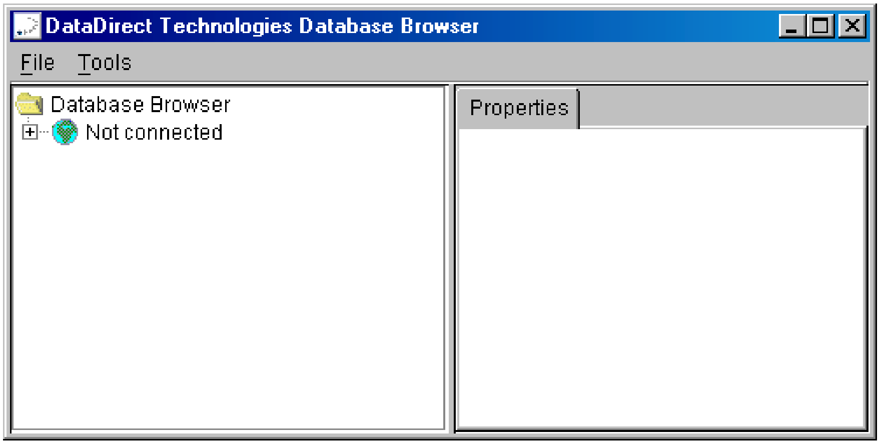
- **To File** option, the results are saved to the specified file.

Browsing the Database

To make sure that you know the correct columns and table names to retrieve from the database and use in your query, you can browse the database using a tool within the Builder named the DataDirect Technologies Database Browser (or Database Browser). A JDBC connection is required to browse the database. The Database Browser also allows you to customize the JDBC filter settings. See "Customizing JDBC Filter Settings" on page 127 for instructions.

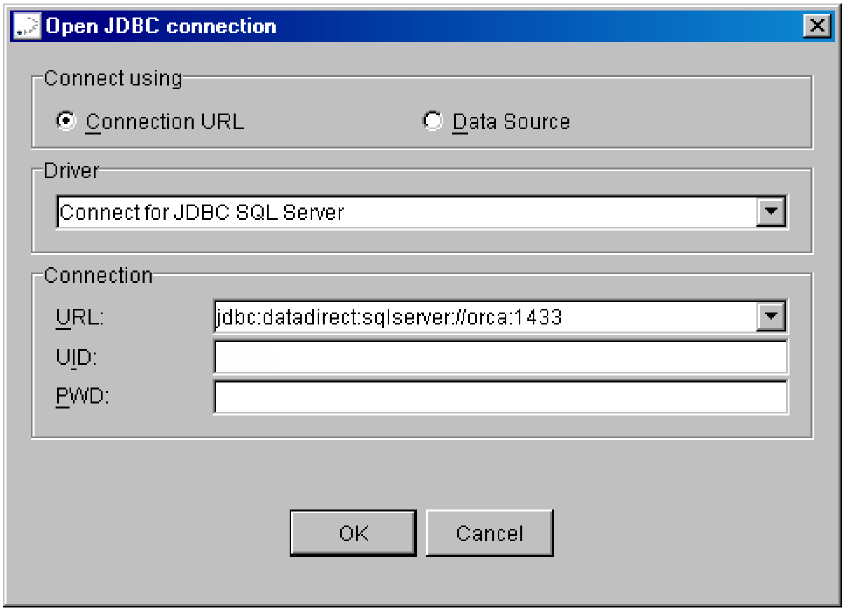
Using the Database Browser

1 To open the Database Browser, select **Tools** / **Database Browser**. The Database Browser appears.



This example shows the Database Browser in a non-connected state. If you were already connected to the database when you opened the Database Browser, you would see the database tree in the left pane.

2 If you are already connected to the database, continue with Step 3. Otherwise, make a JDBC connection. Right-click the **Not connected** node in the Database Browser, and select **Connect to Database**. The Open JDBC Connection dialog box appears.


Open JDBC connection

Connect using

● Connection URL ○ Data Source

Driver

Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

PWD:

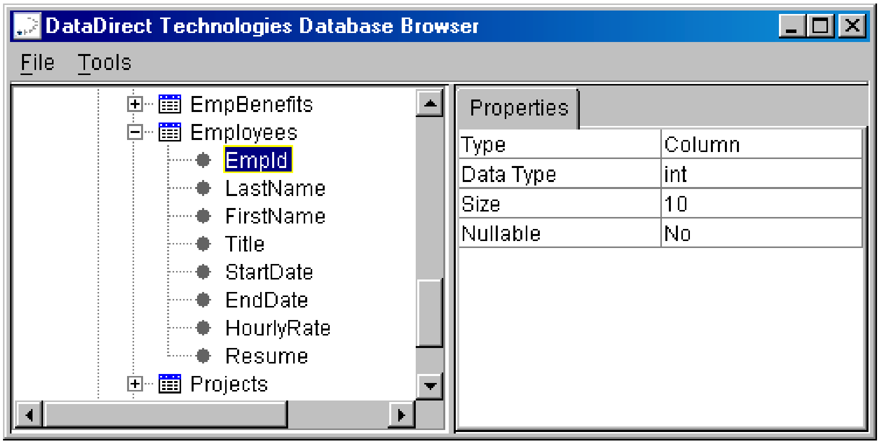
The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

Then, continue with the next step.

3 Browse the database as needed. You can expand any node in the database tree by double-clicking that node or by single-clicking the + (plus sign) for that node. When you select an entity in the database tree, its properties appear in the

right pane of the Database Browser as shown in the following example:.



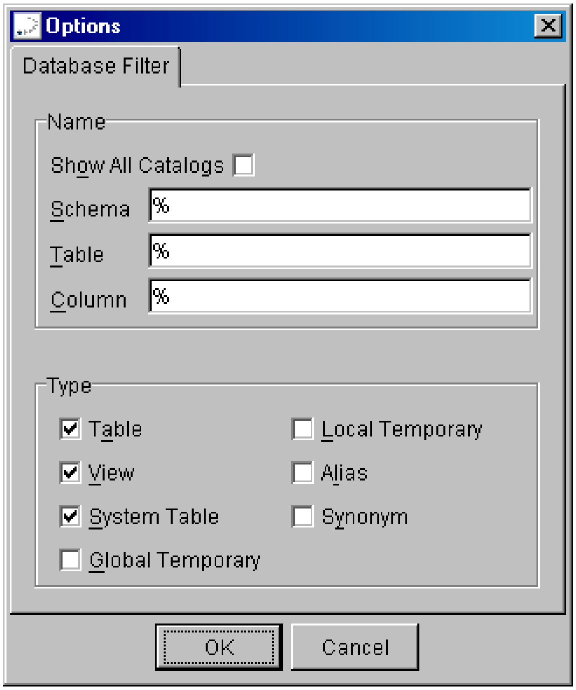
Type	Column
Data Type	int
Size	10
Nullable	No

Customizing JDBC Filter Settings

You can customize JDBC filter settings in the Database Browser to show specific database tables. For example, you may want to customize the filter settings to view database tables that begin with the characters "Emp."

To customize the JDBC filter settings:

1 In the Database Browser, select **Tools** / **Options**. The Options dialog box appears.



Name

Show All Catalogs □

Schema %

Table %

Column %

Type

☑ Table □ Local Temporary

☑ View □ Alias

☑ System Table □ Synonym

□ Global Temporary

2 In the Name group, complete the following information:

Show All Catalogs: Select this checkbox if you want to show all catalogs in the database.

Schema: Using the % character as a wildcard, set a filter to only show schemas named with the characters you specify.

Table: Using the % character as a wildcard, set a filter to only show tables named with the characters you specify.

Column: Using the % character as a wildcard, set a filter to only show columns named with the characters you specify.

3 In the Type group, select the type of information you want to see when you browse the database.

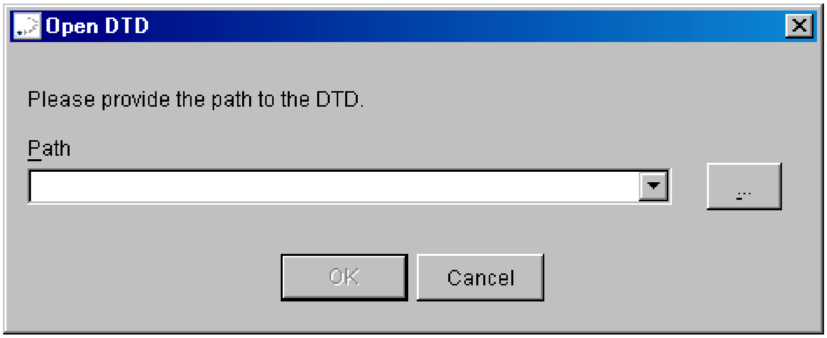
4 When you are satisfied with your JDBC filter settings, click **OK**.

Opening an XML DTD

You can open an XML DTD in a separate window in the Builder for reference. For example, you may want to compare a DTD definition with the structure of an XML document.

To open a DTD:

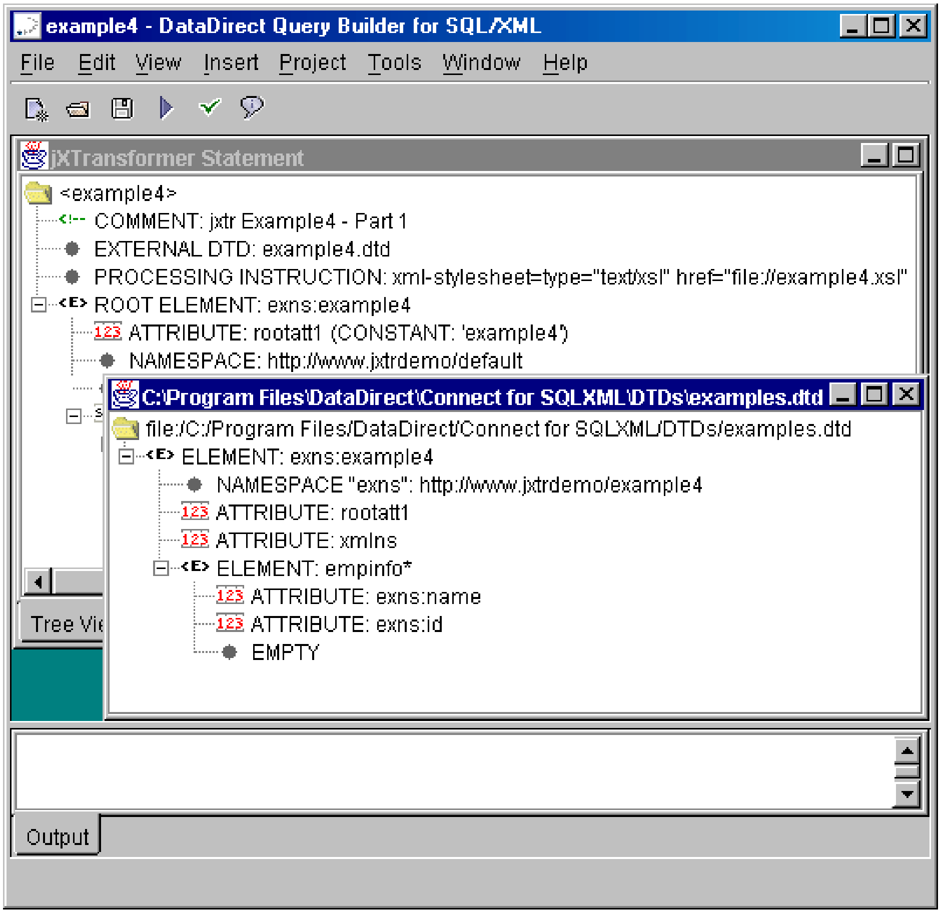
1 Select **File / Open DTD**. The Open DTD dialog box appears.



Please provide the path to the DTD.

2 In the Path field, select a path to the DTD you want to open from the drop-down list, type the path, or click the **Browse (...)** button to browse and select a DTD.

3 Click **OK**. A window opens with the path and name of the DTD displayed in the top-level node. The following figure shows a jXTransformer query with a DTD opened.


example4 - DataDirect Query Builder for SQL/XML
File
Edit
View
Insert
Project
Tools
Window
Help
jXTransformer Statement
Tree Vi
Output
C:\Program Files\DataDirect\Connect for SQLXML\DTDs\examples.dtd

```
<example4>
  <!-- COMMENT: jxtr Example4 - Part 1
  EXTERNAL DTD: example4.dtd
  PROCESSING INSTRUCTION: xml-stylesheet=type="text/xsl" href="file://example4.xsl"
  <E> ROOT ELEMENT: exns:example4
      123 ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
      NAMESPACE: http://www.jxtrdemo/default
```

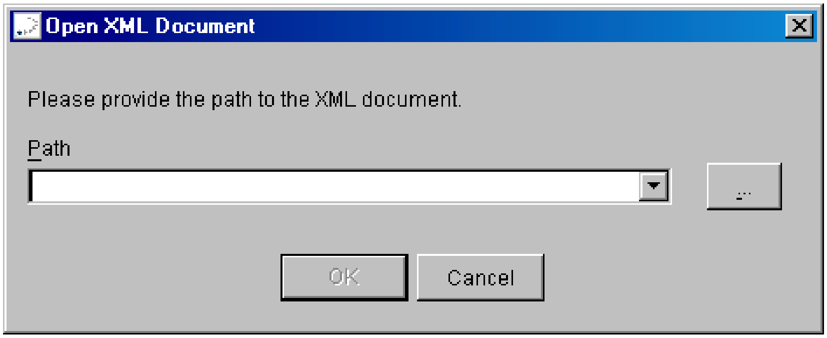
```
file:/C:/Program Files/DataDirect/Connect for SQLXML/DTDs/examples.dtd
  <E> ELEMENT: exns:example4
      NAMESPACE "exns": http://www.jxtrdemo/example4
      123 ATTRIBUTE: rootatt1
      123 ATTRIBUTE: xmlns
      <E> ELEMENT: empinfo*
          123 ATTRIBUTE: exns:name
          123 ATTRIBUTE: exns:id
          EMPTY
```

Opening an XML Document

You can open an XML document in a separate window in the Builder for reference. The Builder allows you to view the XML document in Tree view or Text view.

To open an XML document:

1 Select **File / Open XML Document**. The Open XML Document dialog box appears.



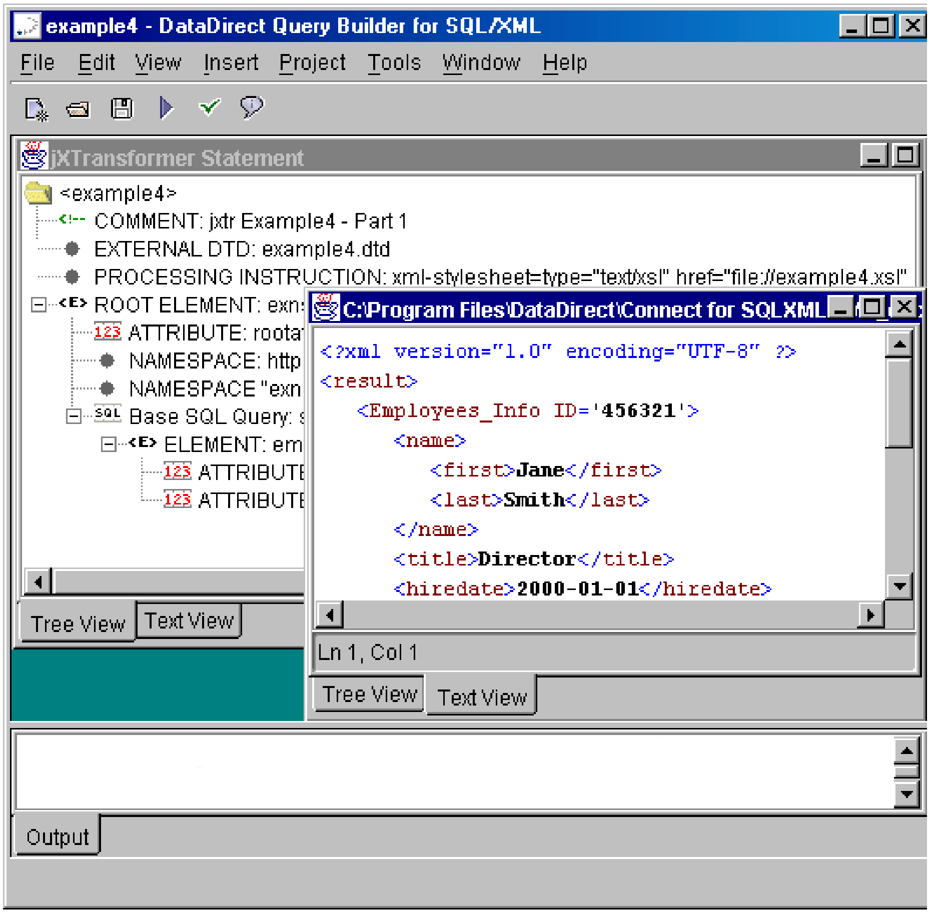
Open XML Document

Please provide the path to the XML document.

Path

2 In the Path field, select a path to the XML document you want to open from the drop-down list, type the path, or click the **Browse (...)** button to browse and select an XML document.

3 Click **OK**. The XML document opens in a separate window with the path and name of the XML document in the title bar. The following figure shows a jXTransformer query with an XML document opened in the separate window.



```
<example4>
  <!-- COMMENT: jxtr Example4 - Part 1
  ● EXTERNAL DTD: example4.dtd
  ● PROCESSING INSTRUCTION: xml-stylesheet=type="text/xsl" href="file://example4.xsl"
  <E> ROOT ELEMENT: exn
    123 ATTRIBUTE: roota
    ● NAMESPACE: http
    ● NAMESPACE "exn
    SQL Base SQL Query: s
      <E> ELEMENT: em
        123 ATTRIBUTE
        123 ATTRIBUTE
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<result>
   <Employees_Info ID='456321'>
      <name>
         <first>Jane</first>
         <last>Smith</last>
      </name>
      <title>Director</title>
      <hiredate>2000-01-01</hiredate>
```

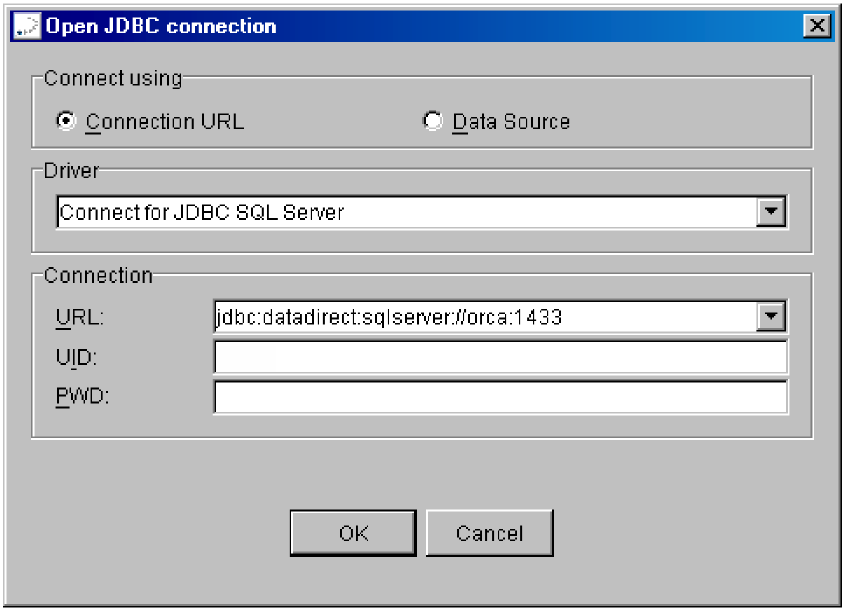
Ln 1, Col 1

Generating DTDs and XML Schemas

This section describes how to generate DTDs and XML schemas from a jXTransformer query using the Builder.

Generating a DTD

1 With a jXTransformer query open, select **Project / Create DTD from query**. If you are not connected to the database, you first must make a JDBC connection. The Open JDBC Connection dialog box appears.



Connect using

● Connection URL ○ Data Source

Driver: Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

PWD:

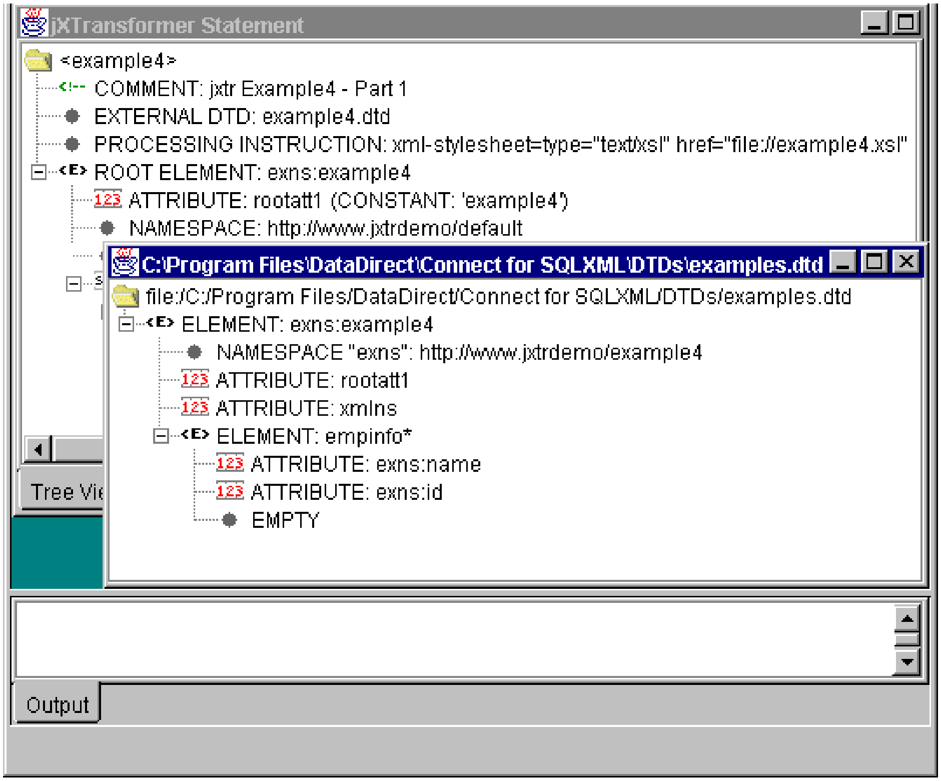
The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

Then, continue with the next step.

2 Use the Create DTD dialog box to navigate to an existing directory or create a new directory for the DTD, and specify the filename of the DTD you want to create. Then, click **Save**.

The DTD is created in the specified directory and appears in a separate window.


jXTransformer Statement
Tree Vie
Output
C:\Program Files\DataDirect\Connect for SQLXML\DTDs\examples.dtd

```
<example4>
   <!-- COMMENT: jxtr Example4 - Part 1
   ● EXTERNAL DTD: example4.dtd
   ● PROCESSING INSTRUCTION: xml-stylesheet=type="text/xsl" href="file://example4.xsl"
⊟ <E> ROOT ELEMENT: exns:example4
      123 ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
      ● NAMESPACE: http://www.jxtrdemo/default
```

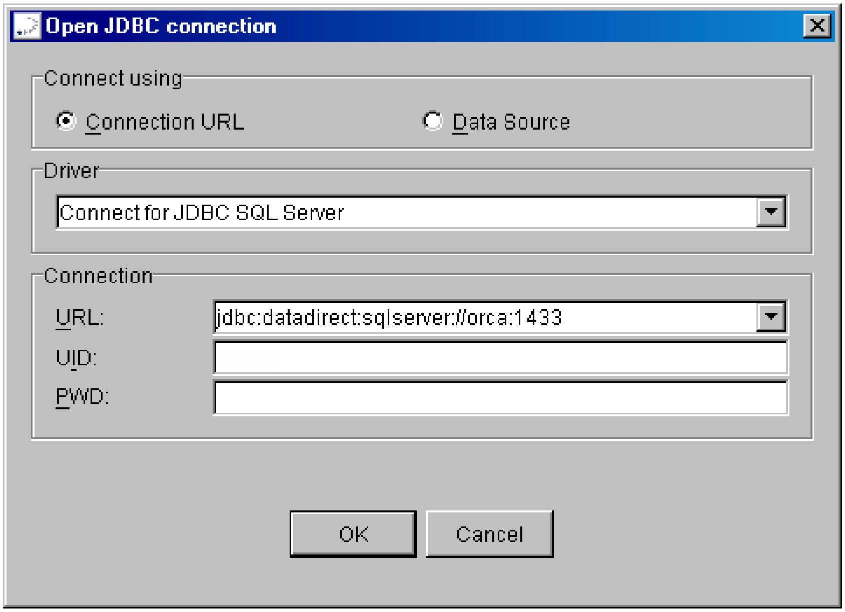
```
file:/C:/Program Files/DataDirect/Connect for SQLXML/DTDs/examples.dtd
⊟ <E> ELEMENT: exns:example4
      ● NAMESPACE "exns": http://www.jxtrdemo/example4
      123 ATTRIBUTE: rootatt1
      123 ATTRIBUTE: xmlns
   ⊟ <E> ELEMENT: empinfo*
         123 ATTRIBUTE: exns:name
         123 ATTRIBUTE: exns:id
         ● EMPTY
```

Generating an XML Schema

To generate an XML schema using the Builder, you must specify every namespace URI defined in your query. A separate XML schema will be created for each specified namespace.

To generate an XML schema:

1 With a jXTransformer query open, select **Project / Create XML Schema from query**. If you are not connected to the database, you first must make a JDBC connection. The Open JDBC Connection dialog box appears.



Driver: Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

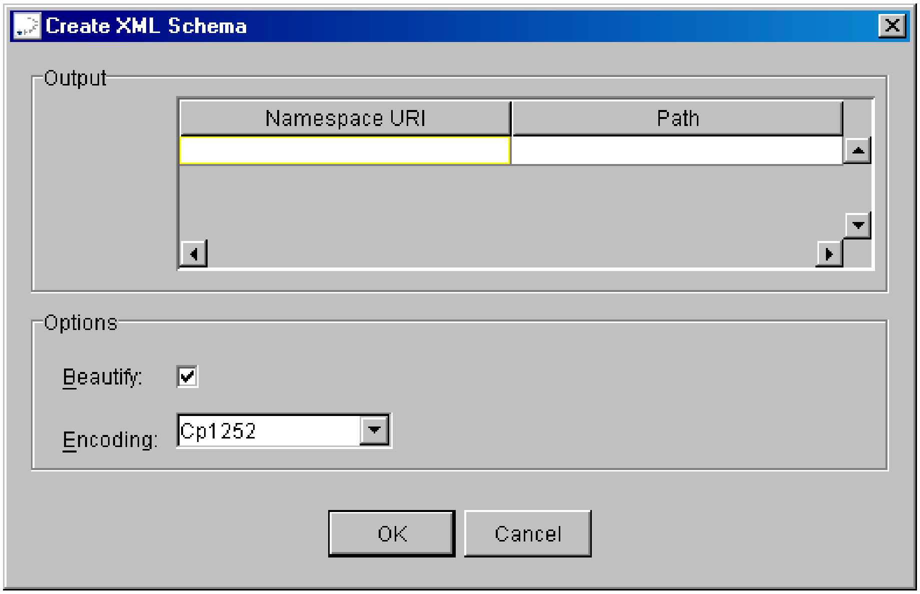
PWD:

The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

Then, continue with the next step.

2 The Create XML Schema dialog box appears.



In the Output group, map namespace URIs or XML constructs that are not qualified by a namespace to XML schema files by completing the following information:

Namespace URI: Type each namespace URI specified in your query as a separate entry. To generate an XML schema for XML constructs that are not qualified by a namespace, create a blank entry in this field, but make sure that you complete the Path field for that entry.

Path: Type the corresponding full path and name of the file to which you want each XML schema saved, or click the **Browse (...)** button to navigate to a file.

3 In the Options group, complete the following information:

Beautify: This option is turned on by default. The Beautify option formats the content of the XML schema with standard indents and line breaks. Uncheck the **Beautify** checkbox to turn off this option. The content of the XML schema will be formatted without indents and line breaks.

Encoding: Select the type of encoding to use for the XML schema. The default depends on your platform.

4 Click **OK**. A separate XML schema is generated for each namespace entry and, if specified, any XML constructs that are not qualified by a namespace.

Executing jXTransformer Write Statements

You can use the Builder to test jXTransformer write statements by executing the jXTransformer write statement in the Text view of the jXTransformer Statement window.

NOTE: You cannot use the Tree view of the jXTransformer Statement window to view or execute jXTransformer write statements.

So that the Builder can locate the XML document referenced in the jXTransformer write statement, the path to the XML document must be specified in the write statement as a URL or the XML document must be in the same directory as the Builder tool. For example, let us look at the following jXTransformer Insert statement fragment:

```
insert xml_document('Insert.xml', 1)
  into Employees (EmpId, FirstName, LastName, Title,
StartDate, HourlyRate, Resume)
 ...
```

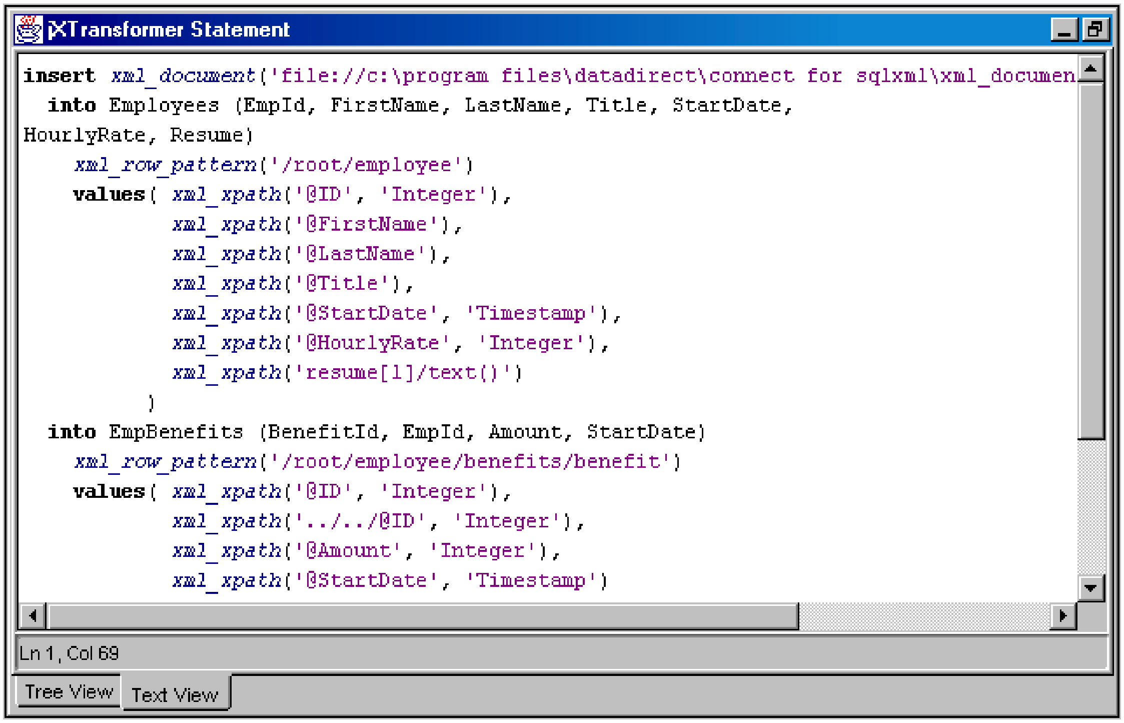
In the previous example, the Builder would be unable to find Insert.xml unless that document existed in the Builder directory. The following example shows the same jXTransformer Insert statement fragment with a path to the XML document specified in a URL format:

```
insert xml_document('file://C:\Program Files\DataDirect\
Connect for SQLXML\XML_documents\Insert.xml', 1)
  into Employees (EmpId, FirstName, LastName, Title,
StartDate, HourlyRate, Resume)
 ...
```

To execute a jXTransformer write statement:

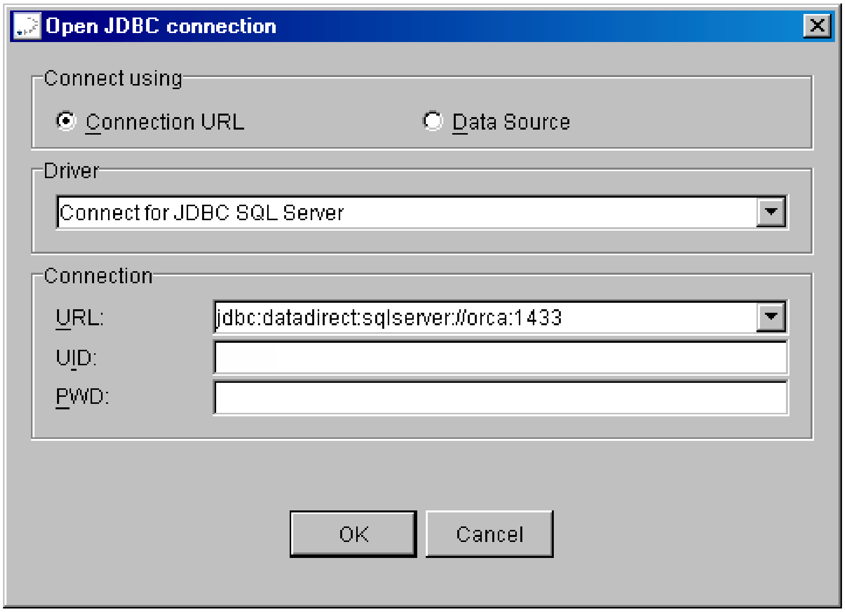
1 Create a Builder project for a jXTransformer write statement or open an existing Builder project for a jXTransformer write statement. See "Creating a Builder Project" on page 77 and "Opening a Builder Project" on page 79 for instructions.

2 If you are creating a Builder project for a jXTransformer write statement, you can copy or import the jXTransformer write statement into the Text view of the jXTransformer Statement window. See "Importing a Query or Statement" on page 115 for instructions. For example, the following figure shows an Insert statement in the jXTransformer statement window.



```
insert xml_document('file://c:\program files\datadirect\connect for sqlxml\xml_documen
  into Employees (EmpId, FirstName, LastName, Title, StartDate,
HourlyRate, Resume)
    xml_row_pattern('/root/employee')
    values( xml_xpath('@ID', 'Integer'),
            xml_xpath('@FirstName'),
            xml_xpath('@LastName'),
            xml_xpath('@Title'),
            xml_xpath('@StartDate', 'Timestamp'),
            xml_xpath('@HourlyRate', 'Integer'),
            xml_xpath('resume[1]/text()')
          )
  into EmpBenefits (BenefitId, EmpId, Amount, StartDate)
    xml_row_pattern('/root/employee/benefits/benefit')
    values( xml_xpath('@ID', 'Integer'),
            xml_xpath('../../@ID', 'Integer'),
            xml_xpath('@Amount', 'Integer'),
            xml_xpath('@StartDate', 'Timestamp')
```

3 Select **Project / Execute Statement** to execute the jXTransformer write statement. If you are not connected to the database, you must make a JDBC connection. The Open JDBC Connection dialog box appears.



Open JDBC connection

Connect using

● Connection URL ○ Data Source

Driver

Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

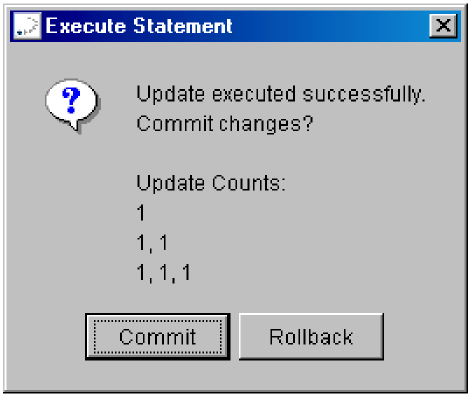
UID:

PWD:

The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

Once you are connected to the database, the Execute Statement dialog box appears confirming if the execution of the statement was successful. It also lists any update counts and prompts you to commit or roll back any database changes resulting from the write operation.



Update executed successfully.
Commit changes?

Update Counts:
1
1, 1
1, 1, 1

In the preceding example, an array of update counts is returned. Each line of the array represents one database table that is updated. Each line returns one value for each of the nodes returned from the row pattern expression for that table. If more than one row is updated, a comma-separated list is returned. For example, 1,1,1 shows that three rows were updated in the database table.

4 Commit or roll back database changes:

- To commit database changes, click the **Commit** button.
- To roll back database changes, click the **Rollback** button.

A message appears confirming the operation you performed.

4 Syntax of SQL/XML Queries

This chapter describes the syntax of Connect *for* SQL/XML queries. The following conventions are used to document the query syntax:

- **Bold** type indicates keywords and tokens that must be entered as part of the query.
- *Italic* type indicates variables.
- Square brackets [ ] surround optional items.
- A vertical rule | indicates an OR separator to delineate items.
- Curly brackets { } surround items that can repeat zero or more times.

The following SQL/XML query syntax shows the high-level components of a Connect *for* SQL/XML query.

```
sqlxml_query ::= select
SQL_value_expression | XML_value_expression
{, SQL_value_expression | XML_value_expression}
rest_of_sql_select
```

The following sections define each of the high-level components of a Connect *for* SQL/XML query:

High-level Component	See...
SQL_value_expression	"SQL_value_expression" on page 142
XML_value_expression	"XML_value_expression" on page 142
rest_of_sql_select	"rest_of_sql99_select" on page 151

SQL_value_expression

SQL_value_expression is all level 1 alternatives for a value expression, including the optional AS clause, as defined in the document *ISO/IEC 9075-2*.

XML_value_expression

The syntax for *XML_value_expression* is:

```
XML_value_expression ::= value_expression_primary |
XML_value_function
```

where:

value_expression_primary is a primitive value expression, such as a column reference, a variable reference, a parameter reference, and a literal. The complete definition can be found in the document *ISO/IEC 9075-2*. Also, *value_expression_primary* can be the SQL/XML operator XMLAGG. This operator is described in the following section.

XML_value_function is one of the following SQL/XML operators: XMLCONCAT, XMLELEMENT, or XMLFOREST. These operators are described next.

XMLAGG

XMLAGG produces a forest of XML elements from a collection of XML elements. XMLAGG accepts a single XML value expression as its argument, and produces a forest of elements by collecting the XML values that are returned from multiple rows and concatenating the values to make one value. XMLAGG concatenates the values returned from one column of multiple rows, unlike XMLCONCAT, which concatenates the values returned from multiple columns in the same row.

If a returned value is NULL, it is ignored in the result. If all the returned values are NULL, the result is NULL.

Subqueries in SQL/XML are allowed to return only one row; therefore, to return more than one row of values in a SQL/XML subquery, you must use XMLAGG.

Syntax

XMLAGG ::= **XMLAGG** (*XML_value_expression* [ORDER BY *sort_list*])

where:

See “XML_value_expression” on page 142 for a definition of *XML_value_expression*.

sort_list is all alternatives for a sort list as defined in the document *ISO/IEC 9075-2.*

Example 1

The following example produces a list of all employees in the company by producing an element for each employee, ordering the list by the last name, and concatenating the result. The result set will have one column and one row.

```
SELECT
 XMLAGG (XMLELEMENT (Name "emp", e.LastName)
   ORDER BY e.LastName) AS "Employee List"
FROM Employees e
```

Result Set:

Employee List
<emp>Allen</emp><emp>Ayers</emp><emp>Cover</emp> ... <emp>Westbrook</emp><emp>Williams</emp>

Example 2 The following example demonstrates using XMLAGG in a subquery.

```
SELECT
  XMLELEMENT(NAME "emp", e.LastName,
    XMLELEMENT(Name "projects",
      (SELECT
        XMLAGG(XMLELEMENT(NAME "project, p.ProjID))
      FROM Projects p
      WHERE EXISTS (
        SELECT * FROM Assignments a
          WHERE a.EmpId=e.EmpId and a.ProjId=p.ProjId))))
          AS "Employees"
FROM Employees e
```

Result Set:

Employees

`<emp>Ayers`
`  <projects>`
`    <project>3</project>`
`    <project>6</project>`
`</emp>`

`...`

`<emp>Marshall`
`  <projects>`
`    <project>1</project>`
`    <project>5</project>`
`</emp>`

`...`

Example 3 The following example uses a Group By clause to group all tasks assigned to each employee and employee project.

```
SELECT
  XMLELEMENT (NAME "Employee_Tasks_by_Project",
    XMLATTRIBUTES (a.EmpId, a.ProjId AS "Project"),
    XMLAGG (XMLELEMENT (NAME "Task", a.Task) ) )
  AS "Employee Task List"
FROM Assignments a GROUP BY a.EmpId, a.ProjId
```

Result Set:

Employee Task List

<Employee_Tasks_by_Project EmpId="1" Project="1"
  <Task>Analysis</Task>
  <Task>Documentation</Task>
  <Task>Planning</Task>
  <Task>Testing</Task>
</Employee_Tasks_by_Project>
<Employee_Tasks_by_Project EmpId="1" Project="5"
  <Task>Analysis</Task>
  <Task>Documentation</Task>
</Employee_Tasks_by_Project>
<Employee_Tasks_by_Project EmpId="2" Project="3"
  <Task>Analysis</Task>
  <Task>Development</Task>
</Employee_Tasks_by_Project>

XMLCONCAT

XMLCONCAT produces a forest of elements by concatenating a list of XML values. XMLCONCAT accepts a list of XML value expressions as its arguments, and produces a forest of elements by concatenating the XML values that are returned from the same row to make one value. XMLCONCAT performs a similar operation as XMLFOREST, except that XMLCONCAT expressions must evaluate to type XML.

If a returned value is NULL, it is ignored in the result. If all the returned values are NULL, the result is NULL.

Syntax

XMLCONCAT ::= **XMLCONCAT** (*XML_value_expression* {, *XML_value_expression*})

See "XML_value_expression" on page 142 for a definition of *XML_value_expression*.

Example 1

The following example produces an XML element for the first and last names, concatenates the result, and creates a one-column result set:

```
SELECT
  XMLCONCAT (
    XMLELEMENT (NAME "first", e.FirstName),
    XMLELEMENT (NAME "last", e.LastName)) AS "Result"
FROM Employees e
```

Result Set:

Result
<FirstName>Marc</FirstName><LastName>Marshall</LastName>
...
<FirstName>Richard</FirstName><LastName>Gamble</LastName>

Example 2 The following example produces an XML element for the first and last names, concatenates the result, and creates a two-column result set:

```
SELECT e.EmpId,
  XMLCONCAT (
    XMLELEMENT (NAME "first", e.FirstName),
    XMLELEMENT (NAME "last", e.LastName)) AS "Result"
FROM Employees e
```

Result Set:

EmpId	Result
1	<FirstName>Marc</FirstName><LastName>Marshall</LastName>
	...
20	<FirstName>Richard</FirstName><LastName>Gamble</LastName>

XMLELEMENT

XMLELEMENT is an operator that returns an XML element given an XML element name, an optional list of attributes, and an optional list of values as the content of the new element.

Syntax

```
XMLELEMENT ::= XMLELEMENT (NAME xml_element_name
[XMLATTRIBUTES (value_expression [AS attribute_name] [{,
value_expression [AS attribute_name]}] )
[{, SQL_value_expression | XML_value_expression}])
```

where:

xml_element_name is any valid XML element name. This value must be within double quotes.

XMLATTRIBUTES creates a list of XML attributes for the enclosing XML element.

attribute_name is any valid XML attribute name. This value must be within double quotes.

value_expression is all level 1 alternatives for a value expression as defined in the document *ISO/IEC 9075-2*.

See "SQL_value_expression" on page 142 for a definition of *SQL_value_expression*.

See "XML_value_expression" on page 142 for a definition of *XML_value_expression*.

Example 1

The following example creates the Employee element with a subelement named name. All values in the LastName column of the Employees table are returned.

```
SELECT
  XMLELEMENT(NAME "Employee",
    XMLELEMENT(NAME "name", e.LastName)) AS "Employee List"
FROM Employees e
```

Result Set:

Employee List
<Employee>   <name>Marshall></name> </Employee>
<Employee>   <name>Ayers</name> </Employee>
<Employee>   <name>Simpson</name> </Employee>
...

Example 2 The following example demonstrates defining an XML attribute within an XML element.

```
SELECT
      XMLELEMENT(NAME "Employee",
      XMLATTRIBUTES(e.EmpId, e.LastName as "name"))
FROM Employees e
```

Result Set:

Result
`<Employee EmpId="1" name="Marshall"/>`
`<Employee EmpId="2" name="Ayers"/>`
...
`<Employee EmpId="20" name="Gamble"/>`

XMLFOREST

XMLFOREST produces a forest of XML elements from a given list of arguments. XMLFOREST accepts a list of SQL value expressions as its arguments, and produces an XML element from each value returned. XMLFOREST uses the name of the column as the name of the XML element, unless otherwise specified in an optional AS clause, and uses the value of the SQL value expression as the content of the XML element.

If a value expression evaluates to NULL, then no element is created for that expression. This is unlike XMLELEMENT, where an empty element is created if a value expression evaluates to NULL.

Syntax

```
XMLFOREST ::= XMLFOREST
(value_expression [AS forest_element_name]
[{, value_expression [AS forest_element_name]}] )
```

where:

`value_expression` is all level 1 alternatives for a value expression as defined in the document *ISO/IEC 9075-2*.

`forest_element_name` is any valid SQL identifier as defined in the document *ISO/IEC 9075-2*.

Example

The following example produces four elements (EmpID, FirstName, LastName, and Start) from the value expressions e.EmpID, e.FirstName, e.LastName, and e.StartDate, concatenates the elements produced for each employee, and produces one row for each employee in the result set.

```
SELECT
  XMLFOREST (
    e.EmpId,
    e.FirstName,
    e.LastName,
    e.StartDate AS "Start") AS "EmployeeInformation"
FROM Employees e
```

Result Set:

EmployeeInformation
<EmpId>1</EmpId><FirstName>Marc</FirstName><LastName>Marshall</LastName><Start>1994-09-01</Start>
...
<EmpId>20</EmpId><FirstName>Richard</FirstName><LastName>Gamble</LastName><Start>1996-10-01</Start>

rest_of_sql99_select

This section of a SQL/XML query allows you to specify the database table from which the data is retrieved and the conditions by which the data is retrieved. The From clause defines the database tables and other optional SQL clauses, such as the Where clause, define the conditions.

Syntax

```
rest_of_sql99_select ::= from_clause [where_clause]
[having_clause] [group_by_clause]
```

These clauses are defined in the document *ISO/IEC 9075-2*.

Executing SQL/XML Queries

The SQL/XML JDBC driver processes the SQL/XML query and sends one or multiple SQL statements through to the database to retrieve the result set.

NOTE: As database vendors begin implementing SQL/XML extensions to SQL, the SQL/XML JDBC driver may not need to process the SQL/XML query before sending it to the database. Whether the driver processes the SQL/XML query in the future will depend on performance advantages.

See Chapter 7, “Using the SQL/XML JDBC Driver and JDBC API Extensions” on page 191 for information about this driver and using SQL/XML queries in your Java application.

5 Syntax of jXTransformer Queries

This chapter describes the syntax of jXTransformer queries. The following conventions are used to document the jXTransformer query syntax:

- **Bold** type indicates keywords and tokens that must be entered as part of the query.
- *Italic* type indicates variables.
- Square brackets [ ] surround optional items.
- A vertical rule | indicates an OR separator to delineate items.
- Curly brackets { } surround items that can repeat zero or more times.

The following jXTransformer syntax shows the high-level components of a jXTransformer query.

```
jxtr_query ::= [xml_document (xml_document_info,] select
xml_constructor {, xml_constructor} rest_of_sql99_select
{; query} [)]
```

The following sections define each of the high-level components of a jXTransformer query:

High-level Component	See...
xml_document	"xml_document" on page 154
xml_document_info	"xml_document_info" on page 154
xml_constructor	"xml_constructor" on page 158
rest_of_sql99_select	"rest_of_sql99_select" on page 169
query	"Query" on page 170

xml_document

xml_document indicates that a complete XML document will be created by the jXTransformer query, instead of an XML document fragment. A complete XML document contains an XML root element. If you do not specify this constructor in your jXTransformer query, a document fragment will be created. The xml_document constructor is followed by xml_document_info, which is described next.

xml_document_info

The xml_document_info section of a jXTransformer query allows you to perform the following tasks:

- Define an XML root element, which creates a complete XML document as opposed to an XML document fragment. Within the XML root element, you can define one or multiple XML attributes, one or multiple XML namespaces, or one or more of either.
- Enter one or multiple comments in the XML document header. This is an optional component of the xml_document_info section.
- Specify one or multiple sets of processing instructions. This is an optional component of the xml_document_info section.
- Reference one private or public external DTD. This is an optional component of the xml_document_info section.

Syntax

```
xml_document_info ::= [xml_comment('comment') |
xml_pi('target', 'instruction') |
xml_external_dtd(['public_id',] 'system_uri')
{, xml_comment('comment') |
, xml_pi('target', 'instruction') |
, xml_external_dtd( ['public_id',] 'system_uri') } ]

xml_element ('root_element_name' [,
   xml_attribute ('attribute_name','attribute_value') |
   xml_namespace (['prefix',] 'uri')]
   {, xml_attribute ('attribute_name','attribute_value') |
    , xml_namespace (['prefix',] 'uri') } ] )
```

where:

`xml_comment` adds an XML comment to the document header of the resulting XML document.

> *`comment`* is any valid string constant value as defined in the SQL specification, except for the optional introducer character_set_specification.

`xml_pi` adds an XML processing instruction to the XML document. Processing instructions allow you to pass one or multiple processing instructions to applications so that the application can use the resulting XML without requiring that extra steps be performed.

> *`target`* is any valid processing instruction name. It identifies the processing instruction to the application. Applications can only process the targets they recognize. An example of a target is xml-stylesheet. The target must be within single quotes.
>
> *`instruction`* is any valid XML processing instruction, for example, `'type="text/xsl"'`. The instruction must be within single quotes.

`xml_external_dtd` creates a reference to an external private or public DTD.

`public_id`, if specified, creates a reference to a public external DTD. This parameter must contain the public identifier for the DTD that is being referenced.

`system_uri` is a System URI that identifies the DTD being referenced. If this parameter is specified, in addition to the public_id parameter, the DTD being referenced is a public external DTD; otherwise, the DTD being referenced is a private external DTD.

`xml_element`, in this instance, is an XML root element. A root element is required for xml_document.

`xml_root_element_name` is any valid XML root element name. This value must be within single quotes and is the only required parameter for xml_element as the root element.

`xml_attribute` creates an XML attribute for the XML root element.

`xml_attribute_name` is any valid XML attribute name. This value must be within single quotes.

`xml_attribute_value` is any constant value for the XML attribute. This value must be within single quotes.

`xml_namespace` creates an XML namespace definition for the enclosing XML element.

`prefix` is the namespace prefix that will be used to qualify elements or attributes with the namespace URI as specified in the uri parameter. This value must be within single quotes and is optional. If you do not specify a prefix, the default namespace definition will be created.

`uri` is the URI that identifies the namespace being defined. This value must be within single quotes and is required.

Example The following example creates a complete XML document that includes document-level jXTransformer constructs such as processing instructions, comments, and a root element with optional namespace and attribute declarations.

```
xml_document(
   xml_comment('Example XML result'),
   xml_pi('xml-stylesheet', 'type="text/xsl"
      href="file://myxsl.xsl"'),
   xml_element('exns:example',
      xml_attribute('rootatt1', 'example'),
      xml_namespace('http://www.jxtrdemo/default'),
      xml_namespace('exns', 'http://www.jxtrdemo/example'),
   SELECT
      xml_element('empinfo',
         xml_attribute('exns:id', e.EmpId),
         xml_attribute('exns:name', e.LastName))
FROM Employees e WHERE e.EmpId < 6 ) )
```

XML Result:

```
<!--Example XML result-->
<?xml-stylesheet type="text/xsl" href="file://myxsl.xsl" ?>
<exns:example
   rootatt1='example'
   xmlns:"http://www.jxtrdemo/default"
   xmlns:exns="http://www.jxtrdemo/example">
   <empinfo exns:id='1' exns:name='Marshall' />
   <empinfo exns:id='2' exns:name='Ayers' />
   <empinfo exns:id='3' exns:name='Simpson' />
   <empinfo exns:id='4' exns:name='O&apos;Donnel' />
   <empinfo exns:id='5' exns:name='Jenkins' />
</exns:example>
```

xml_constructor

The xml_constructor section of a jXTransformer query allows you to perform the following tasks:

- Create XML elements (except for a root element), attributes, CDATA sections, and namespace definitions.
- Retrieve data that you do not want included in the resulting XML document. See "xml_hide" on page 167 for a complete explanation.

Syntax

```
xml_constructor ::= xml_element | xml_attribute |
xml_cdata | xml_namespace | xml_hide | select_expression
```

xml_element

The most simple form of xml_element is used to set the value of an XML element to the value of a database column. For example:

```
xml_element ('name', e.LastName)
```

More advanced forms of xml_element can be used to:

- Define XML attributes, XML CDATA, or XML namespaces within the XML element. For example:

  ```
  xml_element('Employees_Info',
       xml_attribute('ID', e.EmpID),...
  ```

- Define XML subelements within an XML element. For example:

  ```
  xml_element('Employees_Info',
       xml_attribute('ID', e.EmpID),
       xml_element('name',
            xml_element('first', e.FirstName),
            xml_element('last', e.LastName) ) )
  ```

- Nest one or more queries within the XML element.

Syntax

```
xml_element ::= xml_element[_key] ('xml_element_name',
xml_constructor | constant | (query) | select_expression
{, xml_constructor |, constant |, (query) |
, select_expression})
```

where:

`xml_element` creates an XML element.

`xml_element_key` creates an XML element similar to xml_element, but, in addition, xml_element_key indicates that the column selected from the database is the key or part of a multi-value key that uniquely identifies each row retrieved by the query. Specifying xml_element_key allows the underlying jXTransformer query to optimize processing.

`xml_element_name` is any valid XML element name. This value must be within single quotes.

`xml_constructor` is any of the following constructors described in this section: xml_element, xml_attribute, xml_cdata, xml_namespace, or xml_hide.

`constant` is a string constant value as defined in the SQL specification.

`query` is any valid jXTransformer query without *`xml_document_info`*. Queries can be nested. Nested queries must be surrounded by parentheses ( ). When you use nested queries, you must use an explicit key, specified through one, or a combination of, xml_attribute_key, xml_element_key, and xml_hide_key; otherwise, Connect *for* SQL/XML creates an implicit key by making all the columns specified in the parent jXTransformer query part of the key, in which case, the query results could be incorrect.

`select_expression` is a valid SQL Select list expression, except for the optional AS clause, for example, a simple table column name such as e.lastname. In addition, it can contain DataDirect-supported JDBC scalar functions. See "Example 2" on page 161 for an example of using JDBC scalar functions in a jXTransformer query.

Example 1 The following example creates the Employees_Info XML element with a subelement of name in an XML document. The name subelement has two subelements named first and last. Only data for employees whose hourly rate is 125 or higher is returned.

```
SELECT
   xml_element('Employees_Info',
      xml_element('name',
         xml_element('first', e.FirstName),
         xml_element('last', e.LastName) ) )
FROM Employees e WHERE e.HourlyRate >= 125
```

XML Result:

```
<Employees_Info>
   <name>
      <first>Betty</first>
      <last>Jenkins</last>
   </name>
</Employees_Info>
<Employees_Info>
   <name>
      <first>Mike</first>
      <last>Johnson</last>
   </name>
</Employees_Info>
<Employees_Info>
   <name>
      <first>Paul</first>
      <last>Steward</last>
   </name>
</Employees_Info>
<Employees_Info>
   <name>
      <first>Robert</first>
      <last>Healy</last>
   </name>
<Employees_Info>
```

Example 2 The following example demonstrates using a nested query. The explicit key definition is specified by xml_attribute_key in the parent query. Notice that JDBC scalar functions are used in this example.

```
SELECT
  xml_element('employee',
    xml_attribute_key('ID', e.EmpId),
    xml_attribute('name',
   {fn concat({fn concat(e.FirstName, ' ')},e.LastName)}),
    (SELECT
      xml_element('project',
        xml_attribute('name',p.Name),
        xml_attribute('task',a.Task))
    FROM Projects p, Assignments a
    WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
FROM Employees e WHERE e.EmpId < 3
```

XML Result:

```
<employee ID='1' name='Marc Marshall'>
   <project name='Medusa' task='Analysis'></project>
   <project name='Medusa' task='Documentation'></project>
   <project name='Medusa' task='Planning'></project>
   <project name='Medusa' task='Testing'></project>
   <project name='Phoenix' task='Analysis'></project>
   <project name='Phoenix' task='Documentation'></project>
</employee>
<employee ID='2' name='Brian Ayers'>
   <project name='Hydra' task='Analysis'></project>
   <project name='Hydra' task='Documentation'></project>
   <project name='Python' task='Analysis'></project>
   <project name='Python' task='Development'></project>
</employee>
```

xml_attribute

Syntax

```
xml_attribute ::= xml_attribute ('xml_attribute_name',
select_expression | constant | sql99_select) |
xml_attribute_key ('xml_attribute_name', select_expression)
```

where:

`xml_attribute` creates an XML attribute for the enclosing XML element.

`xml_attribute_key` creates an XML attribute similar to xml_attribute, but, in addition, xml_attribute_key indicates that the column selected from the database is the key or part of a multi-value key that uniquely identifies each row retrieved by the query. Specifying xml_attribute_key allows the underlying jXTransformer query to optimize processing.

`xml_attribute_name` is any valid XML attribute name. This value must be within single quotes and is required.

`constant` is a string constant value as defined in the SQL specification.

`select_expression` is a valid SQL Select list expression, except for the optional AS clause, for example, a simple table column name such as e.lastname. The Select expression can select only one column from the database. The Select expression value, as retrieved from the database, provides the value for the named attribute. In addition, *`select_expression`* can contain DataDirect-supported JDBC scalar functions. See "Example 2" on page 161 for an example of using JDBC scalar functions in a jXTransformer query.

`sql99_select` is any valid SQL query (with the exceptions listed in "Rules and Exceptions for jXTransformer Query Syntax" on page 172). The SQL query can contain only one Select expression. When you specify a SQL query, a space-separated concatenation of the complete result set is created. Typically, this is used to construct IDREFs values for an XML attribute. See "Query" on page 170 for an example of using a SQL query to provide the value for an XML attribute.

Example

The following example creates the Employees_Info XML element with an XML attribute named ID and a subelement of name in an XML document. This subelement has two subelements named first and last. In this example, xml_attribute_key is used to indicate that the table column that is selected is a key that uniquely identifies each row retrieved by the query, thereby, optimizing performance.

```
SELECT
xml_element('Employees_Info',
     xml_attribute_key('ID', e.EmpId),
     xml_element('name',
          xml_element('first', e.FirstName),
          xml_element('last', e.LastName) ) )
FROM Employees e WHERE e.EmpId in (12, 14)
```

XML Result:

```
<Employees_Info ID='12'>
     <name>
          <first>Paul</first>
          <last>Steward</last>
     </name>
</Employees_Info>
<Employees_Info ID='14'>
     <name>
          <first>John</first>
          <last>Jenkins</last>
     </name>
</Employees_Info>
```

xml_cdata

Syntax `xml_cdata ::=` **`xml_cdata`** `(`*`select_expression`* `|` *`constant`*`)`

where:

`xml_cdata` creates an XML CDATA section in the resulting XML document that contains the value from the specified column or the constant.

> *`select_expression`* is a valid SQL Select expression, except for the optional AS clause, for example, a simple table column name such as e.lastname. In addition, *`select_expression`* can contain DataDirect-supported JDBC scalar functions. See "Example 2" on page 161 for an example of using JDBC scalar functions in a jXTransformer query.
>
> *`constant`* is a string constant value as defined in the SQL specification.

Example The following example creates a CDATA section beneath the Employees_Info element. The content for the CDATA section is retrieved from the Resume column in the Employees table. A CDATA section is used because the Resume column contains some markup characters that must not be parsed.

```
SELECT
xml_element('Employees_Info',
     xml_attribute_key('ID', e.EmpId),
     xml_element('name',
          xml_element('first', e.FirstName),
          xml_element('last', e.LastName) ),
     xml_element ('HireDate',
         xml_attribute ('start', e.StartDate),
         xml_attribute ('end', e.EndDate)),
     xml_cdata (e.Resume))
FROM Employees e WHERE e.EmpId in (12, 14)
```

XML Result:

```
<Employees_Info ID='12'>
     <name>
          <first>Paul</first>
          <last>Steward</last>
     </name>
     <HireDate start='1997-01-04 00:00:00.0' />
     <![CDATA[ <a href=
       'http://www.xesdemo/resume/12.htm'>P.Steward</a>]]>
</Employees_Info>
<Employees_Info ID='14'>
     <name>
          <first>John</first>
          <last>Jenkins</last>
     </name>
     <HireDate start='1990-04-01 00:00:00.0'
         end='1998-01-12 00:00:00.0' />
     <![CDATA[ <a href=
       'http://www.xesdemo/resume/14.htm'>J.Jenkins</a>]]>
</Employees_Info>
```

xml_namespace

Syntax xml_namespace ::= **xml_namespace** ([**'***prefix***',**] **'***uri***'**)

where:

xml_namespace creates an XML namespace definition for the enclosing XML element.

> *prefix* is the namespace prefix that will be used to qualify elements or attributes with the namespace URI as specified in the uri parameter. This value must be within single quotes and is optional. If you do not specify a prefix, the default namespace is defined.
>
> *uri* is the URI that identifies the namespace being defined. This value must be within single quotes and is required.

Example

The following example associates the namespace prefix emp with the namespace URI http://mycomp.com/employees for the Employees_Info element and its contents. Only the elements with the emp prefix use the http://mycomp.com/employees namespace; the other elements use the default namespace, which is http://mycomp.com/default.

```
SELECT
xml_element('emp:Employees_Info',
     xml_attribute_key('emp:ID', e.EmpId),
     xml_namespace ('http://mycomp.com/default'),
     xml_namespace ('emp', 'http://mycomp.com/employees'),
     xml_element('emp:name',
          xml_element('first', e.FirstName),
          xml_element('last', e.LastName) ) )
FROM Employees e WHERE e.EmpId in (12, 14)
```

XML Result:

```
<emp:Employees_Info emp:ID='12'
   xmlns="http://mycomp.com/default"
   xmlns:emp="http://mycomp.com/employees">
   <emp:name>
      <first>Paul</first>
      <last>Steward</last>
     </emp:name>
</emp.Employees_Info>
<emp:Employees_Info emp:ID='14'>
   xmlns="http://mycomp.com/default"
   xmlns:emp="http://mycomp.com/employees">
   <emp:name>
       <first>John</first>
       <last>Jenkins</last>
   </emp:name>
</emp.Employees_Info>
```

xml_hide

Syntax `xml_hide ::= **xml_hide**[**_key**] (*select_expression*)`

where:

`xml_hide` allows you to specify information to be retrieved from the database that you do not want to include in the resulting XML document.

`xml_hide_key` provides the same functionality as xml_hide, but in addition, xml_hide_key indicates that the column selected from the database is the key or part of a multi-value key that uniquely identifies each row retrieved by the query. Specifying xml_hide_key allows the underlying jXTransformer query to optimize processing.

> *`select_expression`* is a valid SQL Select list expression, except for the optional AS clause, for example, a simple table column name such as e.lastname.

Example The following example creates the Employees_Info XML element and a subelement of name. This subelement has two subelements named first and last. In this example, xml_hide_key is used to retrieve the data in the e.EmpID column that uniquely identifies each of the rows selected by the query, but is not included in the XML results.

```
SELECT
xml_element('Employees_Info',
     xml_hide_key(e.EmpId),
     xml_element('name',
          xml_element('first', e.FirstName),
          xml_element('last', e.LastName) ) )
FROM Employees e WHERE e.EmpId=12
```

XML Result:

```
<Employees_Info>
     <name>
          <first>Paul</first>
          <last>Steward</last>
     </name>
</Employees_Info>
```

select_expression

You can define a jXTransformer query that is similar to a SQL query. For example:

```
select t.a, t.b, t.c from tab t
```

This type of jXTransformer query lists in the XML document the values of each database column selected. For example:

```
row1-value-of-a row1-value-of-b row1-value-of-c
row2-value-of-a row2-value-of-b row2-value-of-c ...
```

rest_of_sql99_select

This section of a jXTransformer query allows you to specify the database table from which the data is retrieved and the conditions by which the data is retrieved. The From clause defines the database tables and other optional SQL clauses, such as the Where clause, define the conditions.

Syntax

```
rest_of_sql99_select ::= from_clause [where_clause]
[having_clause] [group_by_clause]
```

These clauses are defined in the SQL specification.

IMPORTANT: The jXTransformer syntax requires that you use unique table name aliases, for example:

```
FROM employees e
```

and

```
xml_element ('project', e.EmpId)
```

Query

This section of a jXTransformer query allows you to specify multiple top-level queries. Additional top-level queries cannot contain xml_document_info (see "xml_document_info" on page 154). Multiple top-level queries result in the concatenation of the results of all top-level queries.

Example

The following example uses two top-level queries and uses a SQL query to provide the value of an XML attribute. This example selects employees that have benefits assigned to them. For each of the selected employees, a list of benefit IDs is selected. The second top-level query selects benefit information, provided at least one employee has the benefit assigned to them. For each benefit selected, a list of employees that have the benefit assigned to them is selected.

```
SELECT
  xml_element('employees',
    xml_attribute_key('emp-id',
     {fn concat('e-',{fn convert(e1.EmpId,VARCHAR)})}),
    xml_attribute('emp-name',e1.LastName),
    xml_attribute('emp-benefits',
      SELECT
        {fn concat('b-',
           {fn convert(eb1.BenefitId,VARCHAR)})}
      FROM EmpBenefits eb1 WHERE eb1.EmpId=e1.EmpId))
FROM Employees e1 WHERE exists
  (SELECT * FROM EmpBenefits eb2 WHERE e1.EmpId=eb2.EmpId)
;
SELECT
  xml_element('benefits',
    xml_attribute_key('benefit-id',
      {fn concat('b-',
        {fn convert(b2.BenefitId,VARCHAR)})}),
    xml_attribute('benefit-description',b2.Description),
    xml_attribute('benefit-employees',
      SELECT
        {fn concat('e-',
           {fn convert(eb3.EmpId,VARCHAR)})}
      FROM EmpBenefits eb3
       WHERE eb3.BenefitId=b2.BenefitId))
FROM Benefits b2
WHERE exists
  (SELECT * FROM EmpBenefits eb4
  WHERE b2.BenefitId=eb4.BenefitId)
```

XML Result:

```
<employees emp-id="e-1" emp-name="Marshall"
  emp-benefits="b-1 b-3" />
<employees emp-id="e-12" emp-name="Steward"
  emp-benefits="b-3" />
<employees emp-id="e-2" emp-name="Allen"
  emp-benefits="b-1 b-4" />
<benefits benefit-id="b-1" benefit-description="Bonus"
  benefit-employees="e-1 e-2" />
<benefits benefit-id="b-3" benefit-description="Car"
  benefit-employees="e-1 e-12" />
<benefits benefit-id="b-4" benefit-description="Commission"
  benefit-employees="e-2" />
```

Rules and Exceptions for jXTransformer Query Syntax

The jXTransformer query syntax adheres to the following rules and exceptions:

- Does not support the relational set operators (UNION, INTERSECT).
- Must contain unique tables aliases for all tables used in the query.
- Allows the use of parameters (bind markers) where a constant value is allowed in the SQL sections of the query. Parameters are specified using a ? token, and you must provide values for these parameters using the jXTransformer API.
- Supports aggregate functions. For Informix and Sybase, jXTransformer queries that contain nested queries in which both parent and child queries contain aggregate functions cannot be processed using the sorted outer union algorithm.
- Supports the DISTINCT query modifier. If a parent query uses DISTINCT, all child queries must also use DISTINCT; otherwise, unexpected results may be returned.

Executing jXTransformer Queries

A jXTransformer query is executed using a JXTRQuery object, which is an object defined in the jXTransformer API. See Chapter 8, "Using the jXTransformer API" on page 207 for more information about the jXTransformer API.

6 Syntax of jXTransformer Write Statements

This chapter describes the syntax of jXTransformer Write statements. jXTransformer write statements allow you to insert, update, and delete data, as explained in the following list:

- jXTransformer Insert statements insert data from an XML document into a relational database. New rows are created with the new data.
- jXTransformer Update statements update data in a relational database with data from an XML document.
- jXTransformer Delete statements delete data in a relational database. The rows that are deleted are specified by a Where clause in the jXTransformer Delete statement.

The following conventions are used to document the jXTransformer syntax:

- **Bold** type indicates keywords and tokens that must be entered as part of the write statement.
- *Italic* type indicates variables.
- Square brackets [ ] surround optional items.
- A vertical rule **|** indicates an OR separator to delineate items.
- Curly brackets { } surround items that can repeat zero or more times.

Insert Statement

jXTransformer Insert statements insert rows into a relational database based on column values extracted from an XML document. When you define a jXTransformer Insert statement, you specify:

1 The XML document from which to retrieve values to insert into the database table. For example:

```
insert xml_document('emp.xml')
```

2 The database table and columns in which to insert the values. For example:

```
into Employees (EmpId, FirstName, LastName, Title,
StartDate, HourlyRate, Resume)
```

3 The nodes in the XML document from which values will be extracted. These locations are specified using XPath expressions. For example:

```
xml_row_pattern('/insert/employee')
```

4 The XML element or attribute of the nodes specified in Step 3 from which to extract the values to insert in the database table. For example:

```
 values(xml_xpath('@ID', 'Integer'),
        xml_xpath('@FirstName'),
        xml_xpath('@LastName'),
        xml_xpath('@Title'),
        xml_xpath('@StartDate', 'Timestamp'),
        xml_xpath('@HourlyRate', 'Integer'),
        xml_xpath('resume[1]/text()' )
        )
```

Syntax

The syntax of a jXTransformer Insert statement is as follows:

Syntax

```
jxtr_insert ::= insert xml_document
(('reference_to_xml' | ?)[, (ignore_whitespace | ?)] )
[xml_namespace (['prefix',]  'uri')
{,xml_namespace (['prefix',]  'uri')}]
{into table_name [(column_list)]
  xml_row_pattern(('row_pattern_expression' | ? ))
  jxtr_query_expression
}
```

where:

insert xml_document defines the XML document from which values are extracted and inserted into the database table. This construct is required.

> *reference_to_xml* is a reference to the XML document from which the values for the Insert statement are being extracted. The value is the location of the XML document in URL format, for example, 'file://employee.xml'. This value must be surrounded by single quotes.
>
> **?** is a parameter marker. You must set the values for parameter markers in your Java application using the jXTransformer API.
>
> *ignore_whitespace* is 0 or 1. This parameter is optional. If set to 1 (the default), any leading or trailing whitespace that is part of the value of a node is deleted. If set to 0, the whitespace is not deleted. Whitespace is newline, carriage return, spaces, and tabs.

xml_namespace defines a namespace (prefix/URI mapping) for all XPath expressions used in the Insert statement. This construct is optional.

> *prefix* is the namespace prefix that is used to qualify elements or attributes with the namespace URI as specified in the uri parameter. This value must be within single quotes and is optional. If you do not specify a prefix, the default namespace for the XPath expression is defined.
>
> *uri* is the URI that identifies the namespace for the XPath expression to use. This value must be within single quotes and is required when you are defining a namespace for the XPath expression to use.

into defines the database table and columns in which to insert values. This construct is required.

> *table_name* is the name and path of a database table. Refer to the SQL99 specification for more information.
>
> *column_list* is an optional list of database table column names, separated by commas. It specifies the name and order of the columns that will store the values specified in *jxtr_query_expression*. If you omit *column_list*, *jxtr_query_expression* must provide values for all columns defined in the database table and they must be in the same order that the columns are defined in the table. Refer to the SQL99 specification for more information.

xml_row_pattern identifies the nodes in the XML input document from which values are extracted and inserted into the database table. This construct is required.

> *row_pattern_expression* is an absolute XPath expression that returns a node set. The value must be surrounded by single quotes, for example, '//employee'. For each node in the returned node set, one row is inserted into the database table. Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.

jxtr_query_expression is any valid SQL99 query expression with the difference that xml_xpath constructs can be used in the SQL99 query expression where the SQL99 syntax allows expressions. For example:

```
values (
  xml_xpath('@ID', 'Integer'),
  xml_xpath('@FirstName')
)
```

This construct is required.

The syntax for an xml_xpath construct is:

```
xml_xpath(('xpath_expression' | ?)
    [ [, ('java_sql_datatype' | ?) [, (scale | ?)] ]
    [, (mixed_content_index | ?)] ] )
```

where:

xml_xpath identifies the XML element or attribute in the XML input document from which the value is extracted and inserted into the database table. This construct is optional.

> *xpath_expression* is any valid XPath expression that is evaluated relative to each of the nodes returned from the xml_row_pattern construct. The value must be surrounded by single quotes, for example, '@ID'. These expressions, when specified, define the column values being inserted into the database. Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.
>
> *java_sql_datatype* is one of the field string names or int values from java.sql.Types. When using field string names, the value must be surrounded by single quotes, for example, 'INTEGER'. If you do not specify a value, CHAR is used.
>
> *scale* is an integer that specifies the number of digits after the decimal point; it is only valid for DECIMAL and NUMERIC java.sql.Types. For all other types, this value is ignored.
>
> *mixed_content_index* is an integer that specifies the ordinal position of one value in a set of returned values to use for

the Insert statement. The default is 0, which means to concatenate all the returned values and use that concatenated value for the Insert statement. Typically, you need only specify a value for this argument when the element referred to in the XPath expression has mixed content and you want to insert only one value in the database table.

Example The following example inserts new rows into three tables:

- Employees table: EmpID, FirstName, LastName, Title, StartDate, HourlyRate, and Resume columns
- EmpBenefits table: BenefitId, EmpId, Amount, and StartDate columns
- Assignments table: ProjId, EmpId, and Task columns

The values for the columns are extracted from the XML document emp.xml, shown next:

```
<?xml version="1.0" encoding="UTF-8"?>
<insert>
  <employee ID="21" FirstName="Anne" LastName="Dodsworth"
    Title="Miss" StartDate="2001-10-24" HourlyRate="115">
    <resume><![CDATA[
      <a href='http://www.xesdemo/resume/21.htm'>
       A. Dodsworth</a>]]></resume>
    <benefits>
      <benefit ID="1" Amount="1"
      <benefit ID="2" Amount="175" StartDate="2001-11-01"/>
    </benefits>
    <projects>
      <project ID="8">
        <task>Analysis</task>
        <task>Development</task>
      </project>
      <project ID="9">
        <task>Analysis</task>
      </project>
    </projects>
  </employee>
</insert>
```

Insert statement:

```
insert xml_document('emp.xml', 1)
  into Employees (EmpId, FirstName, LastName, Title,
    StartDate, HourlyRate, Resume)
    xml_row_pattern('/insert/employee')
    values(xml_xpath('@ID', 'Integer'),
           xml_xpath('@FirstName'),
           xml_xpath('@LastName'),
           xml_xpath('@Title'),
           xml_xpath('@StartDate', 'Timestamp'),
           xml_xpath('@HourlyRate', 'Integer'),
           xml_xpath('resume[1]/text()' )
          )
  into EmpBenefits (BenefitId, EmpId, Amount, StartDate)
    xml_row_pattern('/insert/employee/benefits/benefit')
    values(xml_xpath('@ID', 'Integer'),
           xml_xpath('../../@ID', 'Integer'),
           xml_xpath('@Amount', 'Integer'),
           xml_xpath('@StartDate', 'Timestamp')
          )
  into Assignments (ProjId, EmpId, Task)
  xml_row_pattern('/insert/employee/projects/project/task')
    values(xml_xpath('../@ID', 'Integer'),
           xml_xpath('../../../@ID', 'Integer'),
           xml_xpath('text()')
          )
```

Results:

One new row inserted into the Employees table:

EmpId	LastName	FirstName	Title	StartDate	EndDate	HourlyRate	Resume
21	Dodsworth	Anne	Miss	2001-10-24		115	<a href= 'http://www.xesdemo/ resume21.htm'> A.Dodsworth</a>

Two new rows inserted into the EmpBenefits table:

EmpId	BenefitId	StartDate	EndDate	Amount
21	1	2001-10-24		1
21	2	2001-22-01		175

Three new rows inserted into the Assignments table:

EmpId	ProjId	Task	StartDate	EndDate	TimeUsed	EstimatedDuration
21	8	Analysis				
21	8	Development				
21	9	Analysis				

Update Statement

jXTransformer Update statements update column values in a relational database with new column values extracted from an XML document. When you define a JXTransformer Update statement, you specify:

1 The XML document from which to retrieve new values to use to update the database table. For example:

```
update xml_document('emp.xml')
```

2 The database table to update. For example:

```
Employees
```

3 The nodes in the XML document from which values will be extracted. These locations are specified using XPath expressions. For example:

```
xml_row_pattern('/update/employee')
```

4 The database column to update and the XML element or attribute of the nodes specified in Step 3 from which to extract the new value for the column. For example:

```
set HourlyRate = xml_xpath('@HourlyRate','Integer')
```

5 The database columns to use to identify which rows to update and the XML elements or attributes of the node from which to retrieve the value that identifies the rows to update. For example:

```
WHERE EmpId = xml_xpath('@ID','Integer')
```

NOTE: It is possible to have multiple parts in the Where clause. For example:

```
WHERE EmpId = xml_xpath('@ID','Integer') and
StartDate < xml_xpath('@StartDate', 'Timestamp')
```

Syntax

The syntax of a jXTransformer Update statement is as follows:

Syntax

```
jxtr_update ::= update xml_document
(('reference_to_xml' | ?)[, (ignore_whitespace | ?)] )
[xml_namespace (['prefix',]  'uri')
{,xml_namespace (['prefix',]  'uri')}]
{table_name xml_row_pattern(
  ('row_pattern_expression' | ? ))
  {set jxtr_set_clause_list}
  where jxtr_search_condition
}
```

where:

`update xml_document` defines the XML document from which values are extracted and updated in the database table. This construct is required.

`reference_to_xml` is a reference to the XML document from which the values for the Update statement are being extracted. The value is the location of the XML document in URL format, for example, 'file://employee.xml'. This value must be surrounded by single quotes.

`?` is a parameter marker. You must set the values for parameter markers in your Java application using the jXTransformer API.

`ignore_whitespace` is 0 or 1 and is optional. If set to 1 (the default), any leading or trailing whitespace that is part of the value of a node is deleted. If set to 0, the whitespace is not deleted. Whitespace is newline, carriage return, spaces, and tabs.

`xml_namespace` defines a namespace (prefix/URI mapping) for all XPath expressions used in the Update statement. This construct is optional.

`prefix` is the namespace prefix that is used to qualify elements or attributes with the namespace URI as specified in the uri parameter. This value must be within single quotes and is optional. If you do not specify a prefix, the default namespace for the XPath expression is defined.

`uri` is the URI that identifies the namespace for the XPath expression to use. This value must be within single quotes and is required when you are defining a namespace for the XPath expression to use.

`table_name` is the name of a database table. This construct is required. Refer to the SQL99 specification for more information.

`xml_row_pattern` defines the XML nodes from which values will be used to update the database table. This construct is required.

`row_pattern_expression` is an absolute XPath expression that returns a node set. The value must be surrounded by single quotes, for example, '//employee'. Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.

`set` defines the database columns to update and the XML document nodes from which to retrieve the new value. This construct is required.

`jxtr_set_clause_list` is any valid SQL99 Set clause list with the addition that xml_xpath constructs can be used in the

SQL99 Set clause list where the SQL99 syntax allows expressions. For example:

```
set HourlyRate = xml_xpath('@HourlyRate','Integer')
```

The syntax for an xml_xpath construct is:

```
xml_xpath(('xpath_expression' | ?)
   [ [, ('java_sql_datatype' | ?) [, (scale | ?)] ]
   [, (mixed_content_index | ?)] ] )
```

where:

`xml_xpath` defines the XML element or attribute from which to retrieve the value to be used to update the database table. This construct is optional.

`xpath_expression` is any valid XPath expression. The value must be surrounded by single quotes, for example, '@ID'. These expressions, when specified, define either the new values to use to update the columns (*`jxtr_set_clause_list`*) or the column values being updated in the database (*`jxtr_search_condition`*). Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.

`java_sql_datatype` is one of the field string names or int values from java.sql.Types. When using field string names, the value must be surrounded by single quotes, for example, 'INTEGER'. If you do not specify a value, CHAR is used.

`scale` is an integer that specifies the number of digits after the decimal point; it is only valid for DECIMAL and NUMERIC java.sql.Types. For all other types, this value is ignored.

`mixed_content_index` is an integer that specifies the ordinal position of one value in a set of returned values to use for the Update statement. The default is 0, which means to concatenate all of the returned values and use that value for the Update statement. Typically, you need

only specify a value for this argument when the element referred to in the XPath expression has mixed content and you want to update only one value in the database table.

`where` defines the rows in the database table to update. This construct is required.

`jxtr_search_condition` is any valid SQL99 search condition with the addition that xml_xpath constructs can be used in the SQL99 search condition where the SQL99 syntax allows expressions. For example:

```
WHERE EmpId = xml_xpath('@ID','Integer')
```

Example

The following example updates one row in one database table, Employees.

The XML document emp.xml, shown next, specifies the column value that the Update statement uses to identify which row in the relational database table to update and the new value to use for the column to be updated. The row with an EmpId of 21 is the row to be updated, and the HourlyRate column is the column to be updated to a value of 120.

```
<?xml version="1.0" encoding="UTF-8"?>
<update>
  <employee ID="21" HourlyRate="120">
  </employee>
</update>
```

Update statement:

```
Update xml_document('emp.xml')
  Employees xml_row_pattern('/update/employee')
    set HourlyRate = xml_xpath('@HourlyRate','Integer')
    WHERE EmpId = xml_xpath('@ID','Integer')
```

Delete Statement

jXTransformer Delete statements delete rows from a relational database table based on column values extracted from an XML document. When you define a jXTransformer Delete statement, you specify:

1. The XML document from which to retrieve values that identify the rows in the database table to delete. For example:

   ```
   delete xml_document('emp.xml')
   ```

2. The database table from which to delete rows. For example:

   ```
   FROM Employees
   ```

3. The nodes of the XML document from which the values that identify which rows to delete are extracted. These locations are specified using XPath expressions. For example:

   ```
   xml_row_pattern('/update/employee')
   ```

4. The database columns to use to identify the rows to delete and the XML element or attribute of the nodes specified in Step 3 from which to extract the value that identifies the rows to delete. For example:

   ```
   WHERE EmpId = xml_xpath('@ID','Integer')
   ```

Syntax

The syntax of a jXTransformer Delete statement is as follows:

```
jxtr_delete ::= delete xml_document
(('reference_to_xml' | ?)[, (ignore_whitespace | ?)] )
[xml_namespace (['prefix',]  'uri')
{,xml_namespace (['prefix',]  'uri')}]
{from table_name
  xml_row_pattern(('row_pattern_expression' | ? ))
    where jxtr_search_condition
}
```

where:

delete xml_document defines the XML document from which values are extracted to identify which rows to delete from the database table. This construct is required.

> *reference_to_xml* is a reference to the XML document from which the values for the Delete statement are being extracted. The value is the location of the XML document in URL format, for example, 'file://employee.xml'. This value must be surrounded by single quotes.
>
> **?** is a parameter marker. You must set the values for parameter markers in your Java application using the jXTransformer API.
>
> *ignore_whitespace* is 0 or 1. This parameter is optional. If set to 1 (the default), any leading or trailing whitespace that is part of the value of a node is deleted. If set to 0, the whitespace is not deleted. Whitespace is newline, carriage return, spaces, and tabs.

xml_namespace defines a namespace (prefix/URI mapping) for all XPath expressions used in the Delete statement.

> *prefix* is the namespace prefix that will be used to qualify elements or attributes with the namespace URI as specified in the uri parameter. This value must be within single quotes

and is optional. If you do not specify a prefix, the default namespace for the XPath expression is defined.

uri is the URI that identifies the namespace for the XPath expression to use. This value must be within single quotes and is required when you are defining a namespace for the XPath expression to use.

from defines the database table on which the delete operation will take place. This construct is required.

table_name is a simple database table name or full pathname. Refer to the SQL99 specification for more information.

xml_row_pattern defines the XML nodes from which values are used to identify the row to delete in the database table.

row_pattern_expression is an absolute XPath expression that returns a node set. The value must be surrounded by single quotes, for example, '//employee'. Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.

where defines the rows in the database table to delete. This construct is required.

jxtr_search_condition is any valid SQL99 search condition with the addition that xml_xpath constructs can be used in the SQL99 search condition where the SQL99 syntax allows expressions. For example:

```
WHERE EmpId = xml_xpath('@ID','Integer')
```

The syntax for the xml_xpath construct is:

```
xml_xpath(('xpath_expression' | ?)
   [ [, ('java_sql_datatype' | ?) [, (scale | ?)] ]
   [, (mixed_content_index | ?)] ] )
```

where:

xml_xpath defines the XML element or attribute from which to retrieve the value to be used to identify the rows to delete from the database table. This construct is optional.

xpath_expression is any valid XPath expression. The value must be surrounded by single quotes, for example, '@ID'. These expressions, when specified, define the column values of the rows being deleted from the database. Refer to the XPath specification at http://www.w3.org/TR/xpath for more information.

java_sql_datatype is one of the field string names or int values from java.sql.Types. When using field string names, the value must be surrounded by single quotes, for example, 'INTEGER'. If you do not specify a value, CHAR is used.

scale is an integer that specifies the number of digits after the decimal point; it is only valid for DECIMAL and NUMERIC java.sql.Types. For all other types, this value is ignored.

mixed_content_index is an integer that specifies the ordinal position of one value in a set of returned values to use for the Delete statement. The default is 0, which means to concatenate all the returned values and use that value for the Delete statement. Typically, you need only specify a value for this argument when the element referred to in the XPath expression has mixed content and you want to use only one of the mixed content parts in the Delete statement's Where clause.

Example The following example deletes rows from three different tables—Assignments, EmpBenefits, and Employees. The rows that are deleted contain the value 21 in the EmpId column of these tables. The example uses a parameter marker for the reference to the XML document that is used in this delete transaction. The value for this marker must be set in the Java application that executes the jXTransformer Delete statement. Also, this example uses an xml_namespace constructor to define a prefix and URI mapping for all XPath expressions used in the Delete statement.

The XML document delete.xml, shown next, specifies the column value that the Delete statement uses to identify which rows in the relational database table to delete.

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:emp="http://www.jxtrdemo/delete">
  <emp:employee ID="21"/>
</root>
```

Delete statement:

```
delete xml_document(?)
  xml_namespace('emp','http://www.jxtrdemo/delete')
  FROM Assignments
    xml_row_pattern('/root/emp:employee')
    WHERE EmpId = xml_xpath('@ID','Integer')
  FROM EmpBenefits
    xml_row_pattern('/root/emp:employee')
    WHERE EmpId = xml_xpath('@ID','Integer')
  FROM Employees
    xml_row_pattern('/root/emp:employee')
    WHERE EmpId = xml_xpath('@ID','Integer')
```

Executing jXTransformer Write Statements

jXTransformer Insert, Update, and Delete statements are executed using a JXTRUpdate object, which is an object defined in the jXTransformer API. See Chapter 8, “Using the jXTransformer API” on page 207 for more information about using the jXTransformer API.

7 Using the SQL/XML JDBC Driver and JDBC API Extensions

This chapter describes the SQL/XML JDBC driver and the classes it uses to process SQL/XML queries. It also provides information about connecting to the database and using SQL/XML queries in Java applications.

The SQL/XML JDBC driver translates SQL/XML statements into database-specific statements that do not contain any XML operators. The driver uses either a DataDirect Connect *for* JDBC driver or the DataDirect SequeLink *for* JDBC driver to communicate with the database.

Because the result of a SQL/XML operator is an XML-typed column and because the current JDBC specification does not support an XML data type, the SQL/XML JDBC driver introduces a new type and associated Java class to represent and access XML-Typed columns. This Java class is com.ddtek.jdbc.jxtr.XMLType.

Driver and Data Source Classes

The driver class for the SQL/XML JDBC driver is:

com.ddtek.jdbc.jxtr.JXTRDriver

The data source class for the SQL/XML JDBC driver is:

com.ddtek.jdbc.jxtr.JXTRDataSource

SQL/XML JDBC Driver Data Source

Because a SQL/XML JDBC driver data source embeds a Connect for JDBC data source or a SequeLink for JDBC data source, constructing a SQL/XML data source (com.ddtek.jdbc.jxtr.JXTRDataSource) requires you to pass a pre-constructed data source to the SQL/XML data source constructor.

Your Connect *for* SQL/XML installation contains examples that show how to create and use Connect *for* SQL/XML data sources. These examples are installed in the examples/src/examples/datasource directory in your Connect *for* SQL/XML installation directory. You can use these examples as templates for creating a SQL/XML data source that meets your needs.

Connection URLs

The syntax of URLs for the SQL/XML JDBC driver is:

jdbc:datadirect:jxtr:*db*://*hostname*:*port*[;*conn_properties*]

where:

db is one of the following values: db2, informix, oracle, sqlserver, sybase, or sequelink. The values db2, informix, oracle, sqlserver, and sybase indicate that the SQL/XML JDBC driver uses an underlying Connect *for* JDBC driver for a database connection. The value sequelink indicates that the SQL/XML JDBC driver uses the SequeLink JDBC driver for a database connection.

hostname is the TCP/IP address or TCP/IP host name of the server to which you are connecting.

port is the number of the TCP/IP port.

conn_properties is a semicolon-separated list of connection properties for the SQL/XML JDBC driver and the DataDirect Technologies JDBC driver you are using for your connection. See "Connection Properties" on page 193 for information about connection properties supported for the SQL/XML JDBC driver. Refer to the Connect *for* JDBC or SequeLink documentation for information about connection properties supported by each DataDirect Technologies JDBC driver.

For example:

```
jdbc:datadirect:jxtr:db2://server1:50000;DatabaseName=SAMPLE;
PackageName=JDBCPKG;binaryEncoding=hex
```

or

```
jdbc:datadirect:jxtr:sequelink://189.23.5.132:19996;
databaseName=stores7;timeStampEncoding=iso8601
```

Connection Properties

The SQL/XML JDBC driver embeds either one Connect *for* JDBC driver or the SequeLink JDBC driver. All connection properties supported by these drivers also are supported by the SQL/XML JDBC driver. Refer to the Connect *for* JDBC or SequeLink documentation for information about supported connection properties.

Table 7-1 lists connection properties that are specific to the SQL/XML JDBC driver, and describes each property. The properties have the form:

property=value

NOTE: All connection property names are case-sensitive. For example, binaryencoding is different than binaryEncoding.

Table 7-1. SQL/XML JDBC Connection Properties

Property	Description
binaryEncoding OPTIONAL	binaryEncoding={base64 | hex}. Specifies the type of binary-to-string conversion to use when retrieving binary information from the database. The default value is base64.
nullReplacementValue OPTIONAL	Sets the value to replace NULL values that are retrieved from the database (for example, nullReplacementValue=no value available). If no value is specified, NULL values are not replaced by another value.
timestampEncoding OPTIONAL	timestampEncoding={odbc | iso8601}. Specifies the type of timestamp-to-string conversion to be used when representing timestamp values in an XML document. The default value is odbc, which uses the standard ODBC encoding as specified in the ODBC specification. Timestamps are converted to a string with the following format mask: YYYY-MM-DD HH:MI:SS[.ffffff]. The value, iso8601, uses an ISO standard for the timestamp-to-string conversion. The format mask used is YYYY-MM-DDTHH:MI:SS[.ffffff].

Using Connect *for* SQL/XML Hints

Connect *for* SQL/XML supports some hints (options or optimizations) that are not supported by the JDBC API. Hints are set by adding comments to your SQL/XML query.

Table 7-2 lists Connect *for* SQL/XML hints that are supported in SQL/XML queries.

Table 7-2. Connect for SQL/XML Hints

Hint	Description
binary_encoding	Specifies the type of binary-to-string conversion to use when retrieving binary information from the database. Valid values are hex and base64 (default).
key_expr	Specifies a select expression that uniquely identifies each of the rows retrieved from the database. Multi-part keys are supported by specifying more than one key_expr value pair in the hint (for example, key_expr=c.CustId; key_expr=c.Name). Because a SQL/XML query can contain nested queries, you can specify multiple sets of keys by prefixing each of the queries with a set of key_expr hints. See "Using key_expr Hints" on page 40 for more information about the key_expr hint.
null_replacement	Sets the value to replace NULL values that are retrieved from the database. If no value is specified, NULL values are not replaced by another value.
rewrite_algorithm	Specifies the way in which the SQL/XML JDBC driver translates the SQL/XML query into one or more statements that the underlying database supports. Valid values are nested_loop and sorted_outer_join. The default is sorted_outer_join. See "Using rewrite_algorithm Hints" on page 41 for more information about rewrite algorithms.

***Table 7-2. Connect for SQL/XML Hints** (cont.)*

Hint	Description
timestamp_encoding	Specifies the type of timestamp-to-string conversion to be used when retrieving timestamp data from the database. Valid values are iso8601 and odbc. The default is odbc.

The following example shows how to specify SQL/XML hints in a SQL/XML query.

```
/*{jxtr-hints timestamp_encoding=iso8601;null_replacement=not applicable; */
SELECT
  XMLELEMENT(name empinfo,
     XMLATTRIBUTES(e.EmpId as "id", e.FirstName as "first",
     e.LastName as "last", e.StartDate as "start", e.EndDate as "end"))
FROM Employees e
```

For more examples of hints, see the examples shipped with the product in the examples/src/examples/sqlxml directory in your Connect *for* SQL/XML installation directory.

com.ddtek.jdbc.jxtr Java Package

In a JDBC programming environment, you use the com.ddtek.jdbc.jxtr Java package included with Connect *for* SQL/XML to perform the following tasks:

- Process the result set returned from a SQL/XML query
- Construct a SQL/XML data source
- Execute jXTransformer statements when using a SQL/XML JDBC driver connection

The com.ddtek.jdbc.jxtr package contains:

- Two classes: XMLType and JXTRDataSource
- One interface: JXTRStatementFactory

See Chapter 8, "Using the jXTransformer API" on page 207 for information about the jXTransformer API and the com.ddtek.jxtr Java package.

XMLType Class

For SQL/XML queries that return XML data in the result set, your Java application must use the getObject method on the XML data and cast the retrieved object to com.ddtek.jdbc.jxtr.XMLType. The XMLType class has different methods to instantiate the actual XML document or document fragment. These methods are:

- generateSAX
- getClob
- getDom
- getJDOM
- getString
- writeXML

All of these methods, except generateSAX and writeXML, require the complete instantiation of the XML document or document fragment on the client side. Depending on the size of the generated XML, this can cause memory usage or performance issues.

JXTRDataSource Class

Because a SQL/XML JDBC driver data source embeds either a Connect *for* JDBC data source or a SequeLink *for* JDBC data source, constructing a SQL/XML data source requires you to pass a pre-constructed data source to the SQL/XML data source constructor. You use the methods in the JXTRDataSource class to accomplish this task. You also can set any of the connection properties defined in Table 7-1, "SQL/XML JDBC Connection Properties" on page 194 using this class.

JXTRStatementFactory Interface

The methods of this interface let you process jXTransformer statements when you are using a SQL/XML JDBC driver connection in a Java application.

For more information about the com.ddtek.jdbc.jxtr package, see the Javadoc shipped with Connect *for* SQL/XML.

Connecting to the Database

Once Connect *for* SQL/XML is installed and your application is using the SQL/XML JDBC driver, you can connect from your application to your database in either of the following ways:

- Using a connection URL through the JDBC Driver Manager as described in this section.
- Using a JNDI data source. See "SQL/XML JDBC Driver Data Source" on page 192 for more information about connecting using JNDI data sources.

You can connect through the JDBC Driver Manager with the DriverManager.getConnection method. This method uses a string containing a URL.

The following list provides a summary of the steps required to connect to the database using a connection URL. After this list, each step is described in more detail.

1. Set your CLASSPATH to include the DataDirect Technologies JDBC driver you are using for the connection and the jxtr.jar file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate the jar files on your computer.
2. Register the driver.
3. Pass the driver's connection URL.
4. Test the connection.

1. Setting the Classpath

The jar files that must be defined in your CLASSPATH variable depend on whether you are using a Connect *for* JDBC driver or the SequeLink *for* JDBC driver. If the files are not defined on your

CLASSPATH, you will receive the a `class not found` error when trying to load the driver.

DataDirect Connect for JDBC Drivers

Set your system CLASSPATH to include the following entries, where *`driver.jar`* is the driver jar file (for example, sqlserver.jar) and *`install_dir`* is the path to your Connect *for* SQL/XML installation directory:

```
install_dir/lib/base.jar
install_dir/lib/util.jar
install_dir/lib/driver.jar
install_dir/lib/jxtr.jar
```

Windows Example

```
CLASSPATH=
.;c:\CfSQLXML\lib\base.jar;c:\CfSQLXML\lib\util.jar;
c:\CfSQLXML\lib\sqlserver.jar;c:\CfSQLXML\lib\jxtr.jar
```

UNIX Example

```
CLASSPATH=.;/home/user1/CfSQLXML/lib/base.jar;/home/user1/
CfSQLXML/lib/util.jar;/home/user1/CfSQLXML/lib/
sqlserver.jar;/home/user1/CfSQLXML/lib/jxtr.jar
```

DataDirect SequeLink JDBC Driver

Set your system CLASSPATH to include the following entries where *`install_dir`* is the path to your Connect *for* SQL/XML installation directory:

```
install_dir/lib/sljc.jar
install_dir/lib/jxtr.jar
```

Windows Example

```
CLASSPATH=.;c:\CfSQLXML\lib\sljc.jar;c:\CfSQLXML\lib\
jxtr.jar
```

UNIX Example

```
CLASSPATH=
.;/home/user1/CfSQLXML/lib/sljc.jar;/home/user1/CfSQLXML/
lib/jxtr.jar
```

2. Registering the Driver

Registering the SQL/XML JDBC driver tells the JDBC Driver Manager which driver to load. One way to register the driver is to explicitly load the driver class using the standard Class.forName() method and call DriverManager.getConnection().

The class of the SQL/XML JDBC driver is:

com.ddtek.jdbc.jxtr.JXTRDriver

For example:

```
Class.forName("com.ddtek.jdbc.jxtr.JXTRDriver");
```

3. Passing the Connection URL

After registering the SQL/XML JDBC driver, you can pass your database connection information using a connection URL. For example, to specify a connection URL for the SQL/XML JDBC driver that uses the Connect *for* JDBC SQL Server driver:

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:jxtr:sqlserver://server1:1433;User=test;
Password=secret;binaryEncoding=hex");
```

See "Connection URLs" on page 192 for more information about the connection URL used for the SQL/XML JDBC driver.

4. Testing the Connection

To test your connection to the database, you can use the DataDirect Query Builder *for* SQL/XML, a tool provided with Connect *for* SQL/XML for creating and modifying Connect *for* SQL/XML queries. You can create your own Connect *for* SQL/XML query for testing or use one of the example Connect *for* SQL/XML queries in the examples/src/examples directory in the Connect *for* SQL/XML installation directory. For instructions on using the DataDirect Query Builder *for* SQL/XML to connect to the database, see "Connecting to the Database" on page 116.

Using SQL/XML Queries in Java Applications

NOTE: To compile and run Java applications that use SQL/XML queries, you must add the appropriate jar files to your classpath. For information about the jar files you need to add, refer to the *DataDirect Connect for SQL/XML Installation Guide*.

Typically, a Java application containing a SQL/XML query performs the following tasks:

1. Connects to the database using the SQL/XML JDBC driver
2. Prepares the SQL/XML query in a String object
3. Executes the SQL/XML query
4. Retrieves data from the result set
5. Creates an XML document or document fragment (optional)

The following example shows a SQL/XML query coded in a Java application. For more examples of SQL/XML queries that demonstrate specific SQL/XML features and functions in a Java application, refer to the example files in the examples/src/example/sqlxml directory in the Connect *for* SQL/XML installation directory.

```
/*
 *
---------------------------------------------------------------------------
 * Copyright(c) 2003 DataDirect Technologies. All rights reserved.
 *
 * This product includes Xerces, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2000 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes Xalan, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2000 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes JDOM, developed by the JDOM Project
 * (http://jdom.org). Copyright (C) 2001 Brett McLaughlin & Jason Hunter.
 * All rights reserved.
 *
 * Description:
 *
 * SQL/XML Example4:
 * Demonstrates the following hints for SQL/XML queries
 * (a) null replacement hint.
 * (b) timestamp encoding hint.
 *
 * The example can be invoked either through the Example class or by invoking
 * the main in this class.
 *
 * See the accompanying readme file for more information on the
 * provided examples.
 *
 *
---------------------------------------------------------------------------
 */
package examples.sqlxml;

import java.io.OutputStreamWriter;
import java.sql.*;
import com.ddtek.jdbc.jxtr.*;
import examples.*;

public class SQLXMLExample4 extends Example
{

  /**
   * Executes the 'SQL/XML Example4' example
   */
  public void execute() throws Exception
  {
    // Output example description
    System.out.println( "=================================" );
    System.out.println( "= SQL/XML Example4 demonstrates =" );
    System.out.println( "=================================" );
    System.out.println( "  the following hints for SQL/XML queries." );
    System.out.println( "  (a) null replacement hint." );
    System.out.println( "  (b) timestamp encoding hint." );

    // Load properties from the resource
    loadProperties();

    // Create the SQL/XML JDBC connection
    connectWithConnectForSQLXML();

    // Build SQL/XML query
    StringBuffer sqlXml = new StringBuffer();
    sqlXml.append ( "--{jxtr-hints null_replacement=NA;key_expr=
e.EmpId;timestamp_encoding=odbc " );
    sqlXml.append ( lineSeparator );
    sqlXml.append ( "select " );
    sqlXml.append ( "  xmlelement (NAME \"employee\", " );
    sqlXml.append ( "    xmlattributes(e.EmpId as \"id\"), " );
    sqlXml.append ( "    xmlelement(NAME \"names\", " );
    sqlXml.append ( "      xmlelement(NAME \"first\", e.FirstName), " );
    sqlXml.append ( "      xmlelement(NAME \"last\", e.LastName)), " );
    sqlXml.append ( "    xmlelement(NAME \"hire-dates\", " );
    sqlXml.append ( "      xmlattributes(e.StartDate as \"start\",e.EndDate
as \"end\") ");
    sqlXml.append ( "  )) as SQLXMLCOL1 " );
    sqlXml.append ( "from Employees e where e.EmpId = 12 " );
    // Output result header
    System.out.println( "---------------------------------------" );
    System.out.println( "Query result (via getString on XMLType)" );
    System.out.println( "---------------------------------------" );

    // Execute the SQL/XML query
    resultSet = stmt.executeQuery(sqlXml.toString());

    // Retrieve data from the resultset
    while(resultSet.next())
    {
      XMLType xmlType = (XMLType)resultSet.getObject(1);
      System.out.println(xmlType.getString());
    }
    resultSet.close();

    // Close SQL/XML JDBC connection
    disconnect();
  }

  /**
   * Main method.
   */
  public static void main ( String[] args ) throws Exception
  {
    Example thisDemo = new SQLXMLExample4();
    thisDemo.execute();
  }
}
```

8 Using the jXTransformer API

In a JDBC programming environment, you use the jXTransformer API to perform the following tasks:

- Execute jXTransformer queries to return results as XML documents or document fragments, or result sets of XML values. 8This API can return an XML document or XML document fragment in any of the supported formats: DOM level 2, JDOM, character stream, or SAX2.
- Execute jXTransformer write statements (Insert, Update, or Delete) to write data to a relational database from an XML document and return a n update count of the rows inserted, updated, or deleted.

This chapter describes the jXTransformer API and the classes it uses to process jXTransformer queries and write statements. It also provides information about connecting to the database and about using jXTransformer queries and write statements in a Java application.

See Chapter 7, "Using the SQL/XML JDBC Driver and JDBC API Extensions" on page 191 for information about the JDBC API extension classes and methods used to execute SQL/XML queries, about connecting to the database, and about using SQL/XML queries in a Java application.

jXTransformer API Classes

Table 8-1 lists and defines the jXTransformer API classes in the com.ddtek.jxtr package that can be used to create and execute jXTransformer queries and jXTransformer write statements.

Table 8-1. jXTransformer API Classes

Class	Description
com.ddtek.jxtr.JXTRColInfo	Implements methods that allow you to work with XML that is returned from a jXTransformer query as a result set.
com.ddtek.jxtr.JXTRException	Implements a Java exception object that is thrown if an unexpected occurrence is encountered during the processing of a jXTransformer method.
com.ddtek.jxtr.JXTRQuery	Implements the methods required to: ■ Transform the results from a jXTransformer query into any of the supported formats: DOM level 2, JDOM, character stream, or SAX2 ■ Transform the XML results from a jXTransformer query into a result set ■ Generate a DTD or schema based on the jXTransformer query that describes the XML structure ■ Add document-level processing instructions and comments ■ Add a reference to a public or private external DTD or an externally stored XML schema See "JXTRQuery and JXTRResultSetWrapper Classes" on page 211 for more information.

***Table 8-1. jXTransformer API Classes** (cont.)*

Class	Description
com.ddtek.jxtr.JXTRResultSetWrapper	Implements the methods required to execute a SQL query and generate an element-centric or attribute-centric XML document. This class wraps XML elements or attributes around the row and column data that is retrieved from the database. Use this class when you do **not** want to define the hierarchical structure of an XML document. See "JXTRQuery and JXTRResultSetWrapper Classes" on page 211 for more information.
com.ddtek.jxtr.JXTRSaxInputSource	Implements the methods that extend the SAX2 InputSource interface and is typically used with a JXTRSaxReader object.
com.ddtek.jxtr.JXTRSaxReader	Implements the methods required to implement a SAX2 XMLReader interface and is used with a JXTRResultSetWrapper or JXTRQuery object.
com.ddtek.jxtr.JXTRSingleTableUpdate	Implements the methods required to execute a SQL Insert, Update, or Delete statement based on one or multiple sets of parameter marker values retrieved from an input XML document. See "JXTRUpdate and JXTRSingleTableUpdate Classes" on page 213 for more information.
com.ddtek.jxtr.JXTRSingleTableUpdateException	Implements a Java exception object that is thrown if an unexpected occurrence is encountered during the processing of a SQL Insert, Update, or Delete statement.
com.ddtek.jxtr.JXTRStatement	Implements the methods required to determine if a jXTransformer statement is a jXTransformer query statement or a jXTransformer Insert, Update, or Delete statement.

Table 8-1. jXTransformer API Classes (cont.)

Class	Description
com.ddtek.jxtr.JXTRUpdate	Implements the methods required to: ■ Execute a jXTransformer Insert, Update, or Delete statement ■ Specify the format of the XML document from which the data will be retrieved ■ Specify values for parameter markers See "JXTRUpdate and JXTRSingleTableUpdate Classes" on page 213 for more information.
com.ddtek.jxtr.JXTRUpdateException	Implements a Java exception object that is thrown if an unexpected occurrence is encountered during the processing of a jXTransformer Insert, Update, or Delete statement.
com.ddtek.jxtr.JXTRWarning	Extends JXTRException and implements a Java throwable object that is created if a warning condition is encountered during the processing of a jXTransformer method. The object that is thrown can reference other exceptions.

JXTRQuery and JXTRResultSetWrapper Classes

Table 8-2 lists some common tasks that you can perform using the methods of the JXTRQuery and JXTRResultSetWrapper classes. These two classes are the most frequently used jXTransformer API classes when you are working with jXTransformer queries.

Table 8-2. Tasks Performed by the JXTRQuery and JXTRResultSetWrapper Classes

Task	Method
Add document-level comments	addDocumentComment
Add document-level processing instructions	addDocumentPI
Add root attributes	addRootAttribute
Add a namespace definition to a root element	addRootNamespace
Execute a query and return the XML as a DOM level 2 document object	executeDOM
Execute a query and create the XML under the specified DOM level 2 node	executeDOM
Execute a query and return the XML as a JDOM document	executeJDOM
Execute a query and create the XML under the specified JDOM element	executeJDOM
Execute a query and create the XML as a result set. Some of the columns of this result set can contain XML data type values.	executeQuery
Execute a query and invoke the SAX2 callbacks as registered with the specified XML reader	executeSAX

***Table 8-2. Tasks Performed by the JXTRQuery and JXTRResultSetWrapper Classes** (cont.)*

Task	Method
Execute a query and write the XML as a character stream to the specified writer, which uses either UTF-8 encoding or encoding you specify	executeWriter
Generate a DTD describing the structure of the query result and write the DTD on the specified Writer object	generateDTD
Generate one or multiple XML schemas describing the structure of the query result and optionally writes the schemas on the specified Writer object	generateXMLSchema
Set values for parameters	setBigDecimal setBoolean setByte setBytes setDatabaseExtension setDate setDouble setFloat setInt setLong setNull setObject setShort setString setTime setTimestamp
Set configuration options	setBinaryEncoding setNullReplacementValue setTimestampEncoding
Set the root tag name for the resulting XML	setRootTag
Set a private or public external DTD definition	setExternalDTD

Table 8-2. Tasks Performed by the JXTRQuery and JXTRResultSetWrapper Classes (cont.)

Task	Method
Set how order by expressions are interpreted	setOrderByOrdinalBehavior
Set whether an empty element is created when a NULL value is retrieved from the database	setEmptyCreateBehavior

For more information about each jXTransformer API class in the com.ddtek.jxtr package, see the Javadoc shipped with Connect *for* SQL/XML.

JXTRUpdate and JXTRSingleTableUpdate Classes

Table 8-3 lists some common tasks that you can perform using the methods of the JXTRUpdate and JXTRSingleTableUpdate classes. These classes are the most frequently used jXTransformer API classes when you are working with jXTransformer write statements.

JXTRUpdate Class

Table 8-3. Tasks Performed by the JXTRUpdate Class

Task	Method
Execute a jXTransformer Insert, Update, or Delete statement and return an update count	executeUpdate

Table 8-3. Tasks Performed by the JXTRUpdate Class *(cont.)*

Task	Method
Read the XML input from a DOM level 2 object	setDOM
Read the XML input from a JDOM object	setJDOM
Read the XML input from a character stream	setReader
Read the XML input from SAX2 object	setSAX
Set values for parameters	setBigDecimal setBoolean setByte setBytes setDate setDouble setFloat setInt setLong setNull setObject setShort setString setTime setTimestamp
Set configuration options	setBatchSize setBinaryEncoding setNullReplacementValue setTimestampEncoding

For more information about each jXTransformer API class in the com.ddtek.jxtr package, see the Javadoc shipped with Connect *for* SQL/XML.

JXTRSingleTableUpdate Class

The JXTRSingleTableUpdate class performs the tasks described in Table 8-4.

Table 8-4. Tasks of the JXTRSingleTableUpdate Class

Task	Method
Execute the SQL Insert, Update, or Delete statement and return an update count	executeUpdate
Provide the XML input in one of the supported formats: URL, DOM Level 2 document or node, JDOM document or node, character stream, or SAX2	setXMLDocument
Set the namespace definitions used in the XPath expressions	setXMLNamespaces
Set the XPath expression that will be used as the XML row pattern to identify the data to be extracted from the XML document	setXMLRowPattern
Set the XPath expressions used to extract the values from the XML document	setXMLXPath
Set values for parameters	setBigDecimal setBoolean setByte setBytes setDate setDouble setFloat setInt setLong setNull setObject setShort setString setTime setTimestamp

Table 8-4. Tasks of the JXTRSingleTableUpdate Class (cont.)

Task	Method
Set configuration options	setBatchSize setBinaryEncoding setNullReplacementValue setTimestampEncoding

For more information about each jXTransformer API class in the com.ddtek.jxtr package, see the Javadoc shipped with Connect *for* SQL/XML.

Connecting to the Database

Once Connect *for* SQL/XML is installed and your application is using the jXTransformer API, you can connect from your application to your database in either of the following ways:

- Using a connection URL through the JDBC Driver Manager as described in this section.
- Using a JNDI data source. For information about using JNDI data sources to connect, refer to either the *DataDirect Connect for JDBC User's Guide and Reference* or the *DataDirect SequeLink Developer's Reference*, depending on the DataDirect Technologies JDBC driver you are using.

You can connect through the JDBC Driver Manager with the DriverManager.getConnection method. This method takes one parameter, a string that contains a URL.

The following list provides a summary of the steps you take to connect to the database. After this list, each step is described in more detail.

1. Set your CLASSPATH to include the DataDirect Technologies JDBC driver you are using for the connection and the jxtr.jar file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate the jar files on your computer.
2. Register the driver.
3. Pass the driver's connection URL.
4. Test the connection.

1. Setting the Classpath

The jar files that must be defined in your CLASSPATH variable depend on whether you are using a Connect *for* JDBC driver or the SequeLink *for* JDBC driver. If the files are not defined on your CLASSPATH, you will receive the `class not found` error when trying to load the driver.

DataDirect Connect for JDBC Drivers

Set your system CLASSPATH to include the following entries, where *`driver.jar`* is the driver jar file (for example, sqlserver.jar) and *`install_dir`* is the path to your Connect *for* SQL/XML installation directory:

```
install_dir/lib/base.jar
install_dir/lib/util.jar
install_dir/lib/jxtr.jar
install_dir/lib/driver.jar
```

Windows Example

```
CLASSPATH=
.;c:\CfSQLXML\lib\base.jar;c:\CfSQLXML\lib\util.jar;
c:\CfSQLXML\lib\jxtr.jar;c:\CfSQLXML\lib\sqlserver.jar
```

UNIX Example

```
CLASSPATH=.;/home/user1/CfSQLXML/lib/base.jar;/home/user1/
CfSQLXML/lib/util.jar;/home/user1/CfSQLXML/lib/jxtr.jar;/ho
me/user1/CfSQLXML/lib/sqlserver.jar
```

DataDirect SequeLink JDBC Driver

Set your system CLASSPATH to include the following entries where *install_dir* is the path to your Connect *for* SQL/XML installation directory:

```
install_dir/lib/sljc.jar
install_dir/lib/jxtr.jar
```

Windows Example

```
CLASSPATH=
.;c:\CfSQLXML\lib\sljc.jar;c:\CfSQLXML\lib\jxtr.jar
```

UNIX Example

```
CLASSPATH=
.;/home/user1/CfSQLXML/lib/sljc.jar;/home/user1/CfSQLXML/
lib/jxtr.jar
```

2. Registering the Driver

Registering the driver tells the JDBC Driver Manager which driver to load. One way to register a DataDirect Technologies JDBC driver is to explicitly load the driver class using the standard Class.forName() method and call DriverManager.getConnection().

DataDirect Connect for JDBC Drivers

See the following list for the class names of each Connect *for* JDBC driver:

- com.ddtek.jdbc.db2.DB2Driver
- com.ddtek.jdbc.informix.InformixDriver
- com.ddtek.jdbc.oracle.OracleDriver
- com.ddtek.jdbc.sqlserver.SQLServerDriver
- com.ddtek.jdbc.sybase.SybaseDriver

For example:

```
Class.forName("com.ddtek.jdbc.sqlserver.SQLServerDriver");
```

DataDirect SequeLink JDBC Driver

The class name of the SequeLink JDBC Driver is com.ddtek.jdbc.sequelink.SequeLinkDriver.

For example:

```
Class.forName(“com.ddtek.jdbc.sequelink.SequeLinkDriver”);
```

For information about alternative methods of registering the SequeLink JDBC Driver, refer to the *DataDirect SequeLink Developer’s Reference*.

3. Passing the Connection URL

After registering the driver, you can pass your database connection information in the form of a connection URL.

DataDirect Connect for JDBC Drivers

See the following URL formats for each Connect *for* JDBC driver. Use them as templates to create your own connection URLs, substituting the appropriate values specific to your database.

DB2 UDB[1]

```
jdbc:datadirect:db2://server_name:50000;DatabaseName=your_database;
PackageName=your_packagename
```

DB2 OS/390 and iSeries[1]

```
jdbc:datadirect:db2://server_name:50000;Location=db2_location;
CollectionId=your_collectionname;PackageName=your_packagename
```

Informix

```
jdbc:datadirect:informix://server_name:2003;InformixServer=your_server;
DatabaseName=your_database
```

Oracle

```
jdbc:datadirect:oracle://server_name:1521
```

SQL Server [2]

```
jdbc:datadirect:sqlserver://server_name:1433
```

Sybase

```
jdbc:datadirect:sybase://server_name:5000
```

[1] Refer to the DB2 driver chapter of the *DataDirect Connect for JDBC User's Guide and Reference* before configuring your initial connection.

[2] For instructions on connecting to named instances, refer to the SQL Server driver chapter of the *DataDirect Connect for JDBC User's Guide and Reference*.

For example, to specify a connection URL for SQL Server that includes the user ID and password:

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:sqlserver://server1:1433;User=test;Password=secret");
```

NOTES:

- The *server_name* is an IP address or a host name, assuming that your network resolves host names to IP addresses. You can test this by using the ping command to access the host name and verifying that you receive a reply with the correct IP address.
- The numeric value after the server name is the port number on which the database is listening. The values listed here are sample defaults. You should determine the port number that your database is using and substitute that value.

Refer to the *DataDirect Connect for JDBC User's Guide and Reference* for a list of connection properties for each driver.

DataDirect SequeLink for JDBC Driver

The connection URL format is:

```
jdbc:sequelink://hostname:port[;key=value]...
```

NOTES:

- *hostname* is the TCP/IP address or TCP/IP host name of the server to which you are connecting.
- *port* is the TCP/IP port on which the SequeLink server is listening. A default installation of SequeLink Server uses the port 19996.
- *key=value* specifies connection properties. Refer to the *DataDirect SequeLink Developer's Reference* for a list of valid connection properties for the SequeLink JDBC driver.

The following examples show some typical SequeLink JDBC driver connection URLs:

```
jdbc:sequelink://sequelinkhost:19996;
```

```
jdbc:sequelink://189.23.5.25:19996;user=john;
password=whatever
```

```
jdbc:sequelink://189.23.5.132:19996;databaseName=stores7
```

```
jdbc:sequelink://189.23.5.68:19996;databaseName=pubs;
HUser=john;HPassword=whatever
```

```
jdbc:sequelink://sequelinkhost:4006;databaseName=pubs;
DBUser=john;DBPassword=whatever
```

4. Testing the Connection

To test your connection to the database, you can use the DataDirect Query Builder *for* SQL/XML, a tool provided with Connect *for* SQL/XML for creating and modifying Connect *for* SQL/XML queries. You can create your own Connect *for* SQL/XML query for testing or use one of the example Connect *for* SQL/XML queries in the examples/src/examples/jxtrapi directory in the Connect *for* SQL/XML installation directory. See "Connecting to the Database" on page 116 for instructions on using the DataDirect Query Builder *for* SQL/XML to connect to the database.

Using jXTransformer Queries and Write Statements in a Java Application

NOTE: To compile and run Java applications that use jXTransformer queries and write statements, you must add the appropriate jar files to your classpath. For information about the jar files you need to add, refer to the *DataDirect Connect for SQL/XML Installation Guide*.

Typically, a Java application containing a jXTransformer query or write statement performs the following tasks:

1. Connects to the database using the JDBC API.
2. Prepares the jXTransformer query or write statement in a String object.
3. Creates a JXTRQuery or JXTRUpdate object, passing in the JDBC connection and the jXTransformer query or write statement.
4. Sets options that are available to the jXTransformer query or write statement.
5. Optionally, sets parameters for the query or write statement.
6. Uses one of the execute methods to execute the query or write statement. In the case of a jXTransformer query, the method used to execute the jXTransformer query determines the format of the resulting XML (DOM, JDOM, SAX2 event stream, or Writer) or result set. In the case of a jXTransformer write statement, the method also returns an update count of rows inserted, updated, or deleted.

The following examples show a jXTransformer query and a jXTransformer write statement in a Java application.

Example A: jXTransformer Query

The following example shows a jXTransformer query in a Java application. For more examples of jXTransformer queries that demonstrate specific features and functions in a Java application, refer to the examples_readme.txt file in the examples directory in your Connect *for* SQL/XML installation directory.

```
/*
 *---------------------------------------------------------------------------
 * Copyright(c) 2003 DataDirect Technologies. All rights reserved.
 *
 * This product includes Xerces, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2000 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes Xalan, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2000 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes JDOM, developed by the JDOM Project
 * (http://jdom.org). Copyright (C) 2001 Brett McLaughlin & Jason Hunter.
 * All rights reserved.
 *
 * Description:
 *
 * JXTRExample1:
 * Demonstrates
 * (a) Basic XML document fragment construction features.
 * (b) Writing the XML result to a writer created from System.out.
 *
 * The example can be invoked either through the Example class or by invoking
 * the main in this class.
 *
 * See the accompanying readme file for more information on the
 * provided examples.
 *
 * --------------------------------------------------------------------------
 */
```

```
package examples.jxtrapi;

import java.io.OutputStreamWriter;
import java.sql.*;

import com.ddtek.jxtr.*;
import examples.*;

public class JXTRExample1 extends Example
{

  /**
   * Executes the 'JXTRExample1' example
   */
  public void execute() throws Exception
  {
    // Output example description
    System.out.println( "==============================" );
    System.out.println( "= JXTR Example1 demonstrates =" );
    System.out.println( "==============================" );
    System.out.println( "  (a) Basic XML document fragment construction." );
    System.out.println( "  (b) Writing the XML to a writer created from 
System.out." );

    // Load properties from the resource
    loadProperties();

    // Create JDBC connection
    connectWithStandardJDBC();

    // Build JXTR query
    StringBuffer jxtrQ = new StringBuffer();
    jxtrQ.append ( "select " );
    jxtrQ.append ( "  xml_element ('employee', " );
    jxtrQ.append ( "    xml_attribute('ID', e.EmpId), " );
    jxtrQ.append ( "    xml_element('names', " );
    jxtrQ.append ( "      xml_element('first', e.FirstName), " );
    jxtrQ.append ( "      xml_element('last', e.LastName)), " );
    jxtrQ.append ( "    xml_element('hire-dates', " );
    jxtrQ.append ( "      xml_attribute('start', e.StartDate), " );
    jxtrQ.append ( "      xml_attribute('end', e.EndDate)), " );
    jxtrQ.append ( "    xml_cdata(e.Resume)) " );
    jxtrQ.append ( "from Employees e where e.EmpId in (12, 14) " );

    // Construct new JXTRQuery object
    JXTRQuery jxtrQuery = new JXTRQuery ( conn, new String ( jxtrQ ) );

    // Output result header
    System.out.println( "-------------------------------------" );
    System.out.println( "Query result (without implicit root)" );
    System.out.println( "-------------------------------------" );

    // Execute
    OutputStreamWriter systemOutWriter = new OutputStreamWriter ( System.out
);
    boolean generateImplicitRoot=false;
    jxtrQuery.executeWriter ( systemOutWriter, generateImplicitRoot,
systemOutWriter.getEncoding (), 2 );

    // Output result header
    System.out.println( "----------------------------------" );
    System.out.println( "Query result (with implicit root)" );
    System.out.println( "----------------------------------" );

    // Execute
    generateImplicitRoot=true;
    jxtrQuery.executeWriter ( systemOutWriter, generateImplicitRoot,
systemOutWriter.getEncoding (), 2 );

    // Close JDBC connection
    disconnect();

  }

  /**
   * Main method.
   */
  public static void main ( String[] args ) throws Exception
  {
    Example thisDemo = new JXTRExample1();
    thisDemo.execute();
  }
```

```
}
```

Example B: jXTransformer Write Statement

The following example shows a jXTransformer write statement in a Java application. For more examples of jXTransformer write statements that demonstrate specific features and functions in a Java application, refer to the examples_readme.txt file in the examples directory in your Connect *for* SQL/XML installation directory.

```
/*
 *----------------------------------------------------------------------
 * Copyright(c) 2003 DataDirect Technologies. All rights reserved.
 *
 * This product includes Xerces, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2003 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes Xalan, developed by the Apache Software
 * Foundation (http://www.apache.org). Copyright (C) 1999-2003 The Apache
 * Software Foundation.  All rights reserved.
 *
 * This product includes JDOM, developed by the JDOM Project
 * (http://jdom.org). Copyright (C) 2003 Brett McLaughlin & Jason Hunter.
 * All rights reserved.
 *
 * Description:
 *
 * JXTRExample9:
 * Demonstrates
 * (a) Basic insert from an XML document into multiple database tables.
 * (b) Writing the number of inserted rows to System.out.
 *
 * The example can be invoked either through the Example class or by invoking
 * the main in this class.
 *
 * See the accompanying readme file for more information on the
 * provided examples.
 *
 *-----------------------------------------------------------------------------
 */

package examples.jxtrapi;

import java.io.OutputStreamWriter;
import java.sql.*;

import com.ddtek.jxtr.*;
import examples.*;

public class JXTRExample9 extends Example
{

    /**
     * Executes the 'JXTRExample9' example
     */
    public void execute() throws Exception
    {
        // Output example description
        System.out.println( "===============================" );
        System.out.println( "= JXTR Example9 demonstrates =" );
        System.out.println( "===============================" );
        System.out.println( "  (a) Basic insert from an XML document into
multiple database tables." );
        System.out.println( "  (b) Writing the number of inserted rows to
System.out." );

        // Load properties from the resource
        loadProperties();

        // Create JDBC connection
        connectWithStandardJDBC();
        // Turn off autocommit
        conn.setAutoCommit ( false );

        // Set table names
        String[] table = new String[] {"Employees", "EmpBenefits", 
"Assignments"};

        // Build JXTR query
        StringBuffer jxtrU = new StringBuffer();
        jxtrU.append ( "insert xml_document('data/Insert.xml', 1) " );
        jxtrU.append ( "  into " + table[0] + "(EmpId, FirstName, LastName, 
Title, StartDate, HourlyRate, Resume) " );
        jxtrU.append ( "    xml_row_pattern('/root/employee') " );
        jxtrU.append ( "    values(xml_xpath('@ID', 'Integer'), " );
        jxtrU.append ( "      xml_xpath('@FirstName'), " );
        jxtrU.append ( "      xml_xpath('@LastName'), " );
        jxtrU.append ( "      xml_xpath('@Title'), " );
        if ( dateOrTimestamp == java.sql.Types.DATE )
            jxtrU.append ( "      xml_xpath('@StartDate', 'Date'), " );
        else
            jxtrU.append ( "     xml_xpath('@StartDate', 'Timestamp'), " );
        jxtrU.append ( "      xml_xpath('@HourlyRate', 'Integer'), " );
        jxtrU.append ( "      xml_xpath('resume[1]/text()') " );
        jxtrU.append ( "      ) " );
        jxtrU.append ( "  into " + table[1] + "(BenefitId, EmpId, Amount, 
StartDate) " );
        jxtrU.append ( " 
xml_row_pattern('/root/employee/benefits/benefit') " );
        jxtrU.append ( "    values(xml_xpath('@ID', 'Integer'), " );
        jxtrU.append ( "     xml_xpath('../../@ID', 'Integer'), " );
        jxtrU.append ( "     xml_xpath('@Amount', 'Integer'), " );
        if ( dateOrTimestamp == java.sql.Types.DATE )
            jxtrU.append ( "      xml_xpath('@StartDate', 'Date') " );
        else
            jxtrU.append ( "     xml_xpath('@StartDate', 'Timestamp') " );
        jxtrU.append ( "      ) " );
        jxtrU.append ( "  into " + table[2] + "(ProjId, EmpId, Task) " );
        jxtrU.append ( " 
xml_row_pattern('/root/employee/projects/project/task') " );
        jxtrU.append ( "    values(xml_xpath('../@ID', 'Integer'), " );
        jxtrU.append ( "      xml_xpath('../../../@ID', 'Integer'), " );
        jxtrU.append ( "      xml_xpath('text()') " );
        jxtrU.append ( "    )   " );

        // Construct new JXTRUpdate object
        JXTRUpdate jxtrUpdate = new JXTRUpdate ( conn, new String ( jxtrU )
);

        // XML document contains ISO8601 timestamps.
        jxtrUpdate.setTimestampEncoding(JXTRUpdate.TIMESTAMP_AS_ISO8601);

        // Execute
        int[][] insertCount = null;
        try
        {
            insertCount = jxtrUpdate.executeUpdate();
        }
        catch ( Exception ex )
        {
            System.out.println ( "!!! Insert failed !!!" );
            if ( ex instanceof JXTRUpdateException )
            {
                // Get failed count.
                insertCount = ((JXTRUpdateException)ex).getUpdateCount();
            }
            // Rollback changes
            conn.rollback();
            // Throw again.
            throw ex;
        }
        finally
        {
            // Write change count to System.out
            if ( insertCount == null )
            {
                System.out.println( "---------------------------" );
                System.out.println( "No insert count available." );
                System.out.println( "---------------------------" );
            }
            else
            {
                // Output count header
                System.out.println( "------------" );
                System.out.println( "Insert count" );
                System.out.println( "------------" );
                // Write total change count per table to System.out.
                for ( int i = 0; i < insertCount.length; i++ )
                {
                    int count = 0;
                    for ( int j = 0; j < insertCount[i].length; j++ )
                    {
                        count += insertCount[i][j];
                    }
                    System.out.println ( count + " row(s) were inserted into 
table " + table[i] );
                }
            }
        }

        // Commit changes
        conn.commit();

        // Close JDBC connection
        disconnect();

    }

    /**
     * Main method.
     */
    public static void main ( String[] args ) throws Exception
    {
        Example thisDemo = new JXTRExample9();
        thisDemo.execute();
    }


}
```

9 Tutorial: Using SQL/XML Queries

This chapter contains a step-by-step tutorial that shows you how to create a SQL/XML query using the DataDirect Query Builder for SQL/XML (the Builder). It also shows you how to embed that query in a Java application and use Connect *for* SQL/XML hints to provide processing options that are not supported through the standard SQL/XML query syntax.

This tutorial steps you through the process of constructing the following SQL/XML query.

```
SELECT
   XMLELEMENT (NAME "employee",
      XMLATTRIBUTES(e.EmpId as "id"),
      XMLELEMENT(NAME "names",
         XMLELEMENT(NAME "first", e.FirstName),
         XMLELEMENT(NAME "last", e.LastName)),
      XMLELEMENT(NAME "hire-dates",
         XMLATTRIBUTES(e.StartDate as  "start",
         e.EndDate as "end")
    )) as SQLXMLCOL1
FROM Employees e WHERE e.EmpId = 12
```

NOTE: The preceding SQL/XML query example can be found in the SQLXMLExample4.java file in the examples/src/examples/sqlxml directory in your DataDirect Connect *for* SQL/XML installation directory. For information on populating your database with test data and using this example, refer to the examples_readme.txt file in the examples directory.

Our SQL/XML query example generates a result set that looks like this:

SQLXMLCOL1

<employee id='12'>
  <names>
    <first>Paul</first>
    <last>Steward</last>
  </names>
  <hire-dates start='1997-01-04 00:00:00.0'
  end='NA'></hire-dates>
</employee>

This tutorial uses this SQL/XML query example to demonstrate how to accomplish the following tasks:

- Create result sets that can contain XML.
- Use the following Connect *for* SQL/XML hints to specify processing options that are not available in the SQL/XML syntax:
    - null_replacement is a hint that replaces NULL values retrieved from the database with a specified value. In our example, we replace any NULL value that is encountered with the letters "NA."
    - timestamp_encoding is a hint that specifies the type of timestamp-to-string conversion to be used, either iso8601 or odbc, when timestamp information is retrieved from the database.
    - key_expr is a hint that specifies a key or part of a multi-value key. Keys uniquely identify rows in the database selected by the base SQL query. In our example, the key is specified in the Builder and is translated into a key_expr hint.

  See "Using Connect for SQL/XML Hints" on page 195 for information about using Connect *for* SQL/XML hints.

Complete the tasks described in the next sections to create the SQL/XML query in our example. These tasks include:

- "Creating the Builder Project" on page 235
- "Constructing the SQL/XML Query in the Builder" on page 237
- "Using a SQL/XML Query in a Java Application" on page 249

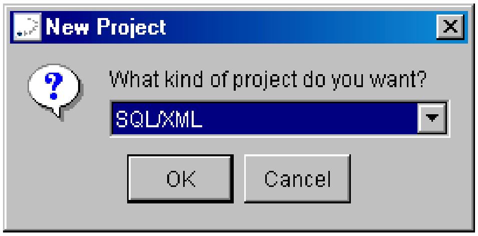
Creating the Builder Project

1 Start the Builder. How you start the Builder depends on your platform:

- **On Windows**: Run the builder.bat file located in the Connect *for* SQL/XML installation directory.
- **On UNIX**: Run the builder.sh shell script located in the Connect *for* SQL/XML installation directory.

During a normal installation, the builder.bat file and builder.sh script are automatically customized to include the path to your JDK. If the installer was unable to detect a required JDK or you want to change the JDK to be used, refer to the *DataDirect Connect for SQL/XML Installation Guide* for instructions on configuring the startup file for the Builder.

2 Open a new Builder project for the SQL/XML query by selecting **File** / **New Project**. The New Project dialog box appears.



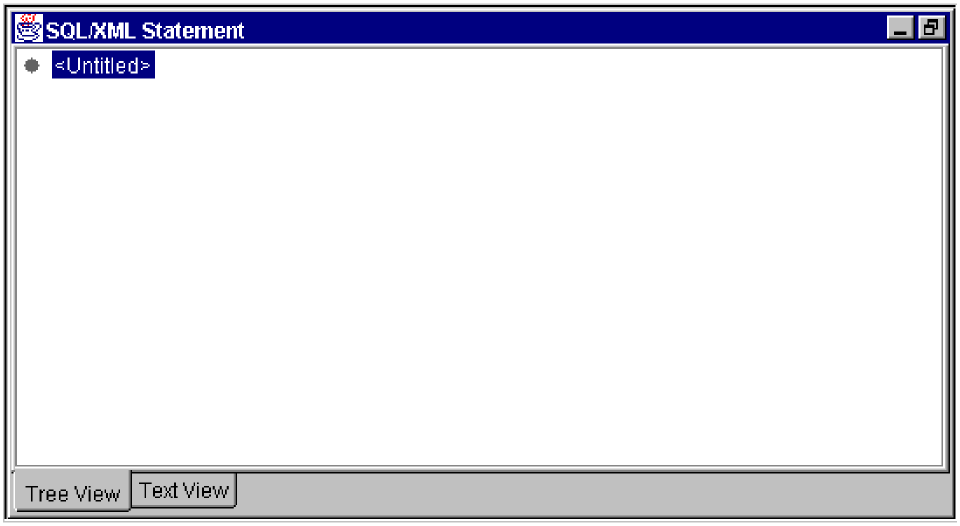
New Project

What kind of project do you want?

SQL/XML

3 From the drop-down list, select **SQL/XML**.

4 Click **OK**. An untitled project node appears in the Tree view of the SQL/XML Statement window.



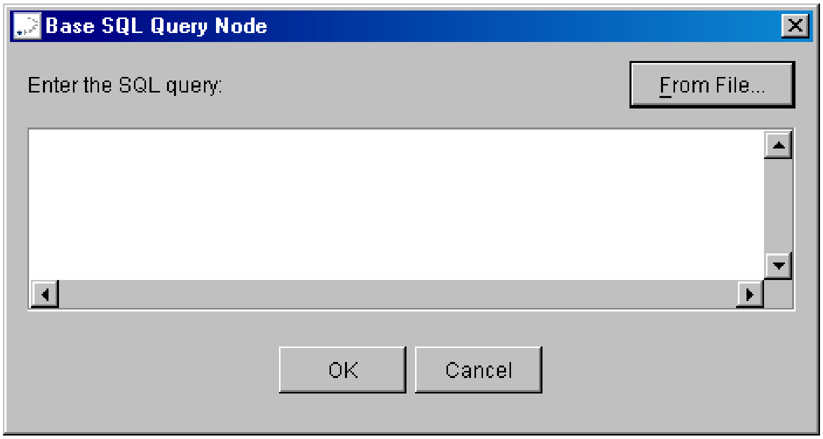
SQL/XML Statement

<Untitled>

5 Save the Builder project by selecting **File** / **Save Project As** and specifying `example4.cfb` as the filename of the project. Notice that the project node is renamed to example4. As you work with this example, you can save your changes to the project file at any time by selecting **File** / **Save Project**.

Constructing the SQL/XML Query in the Builder

1 First, create a Base SQL Query for the SQL/XML query. The Base SQL Query is required. It helps define the data to be retrieved and facilitates the process of constructing the query. To create a Base SQL Query, right-click the project node, and select **Insert / Base SQL Query Node**. The Base SQL Query Node dialog box appears.



TIP: When specifying table names, you must use unique table aliases (for example, Employees e).

2 Type the following Base SQL Query for our SQL/XML query:

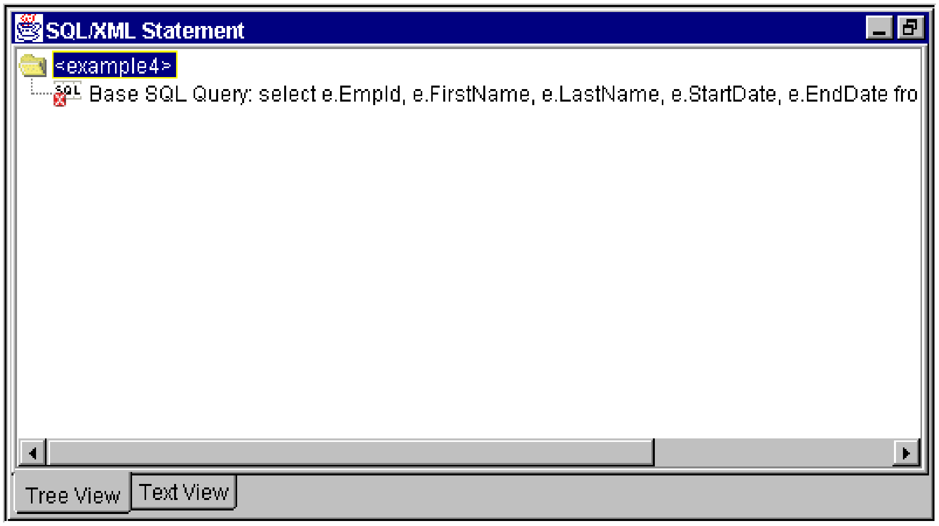
```
SELECT
   e.EmpId,
   e.FirstName,
   e.LastName,
   e.StartDate,
   e.EndDate
FROM Employees e WHERE e.EmpId = 12
```

The Base SQL Query accomplishes the following tasks:

- Selects columns in the query containing data to retrieve
- Specifies the tables from which to retrieve data
- Optionally, specifies filters on the Select statement

Notice that the Where clause in our example specifies a filter on the Select statement of the SQL query.

3 Click **OK**. The Builder checks the Base SQL Query for syntax. The Base SQL Query node appears in the project tree.



Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro

Now that you have defined which data to retrieve from the database for the SQL/XML query, you can define the XML structure, including the hierarchical relationships of the data.

Examine the SQL/XML query in our example. You need to create an XML element that contains employee information for an employee based on their employee ID, including first and last name, as well as the start and end hire dates for that employee.

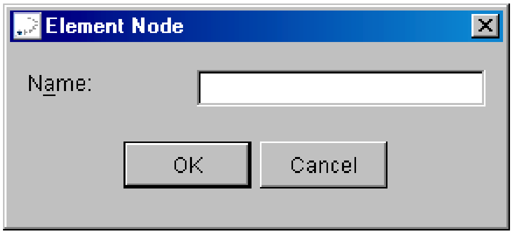
```
SELECT
   XMLELEMENT (NAME "employee",
      XMLATTRIBUTES(e.EmpId as "id"),
      XMLELEMENT(NAME "names",
         XMLELEMENT(NAME "first", e.FirstName),
         XMLELEMENT(NAME "last", e.LastName)),
      XMLELEMENT(NAME "hire-dates",
         XMLATTRIBUTES(e.StartDate as  "start",
         e.EndDate as "end")
```

```
    )) as SQLXMLCOL1
FROM Employees e WHERE e.EmpId = 12
```

Notice that the element named employee in the SQL/XML query is a parent to the following attributes and elements:

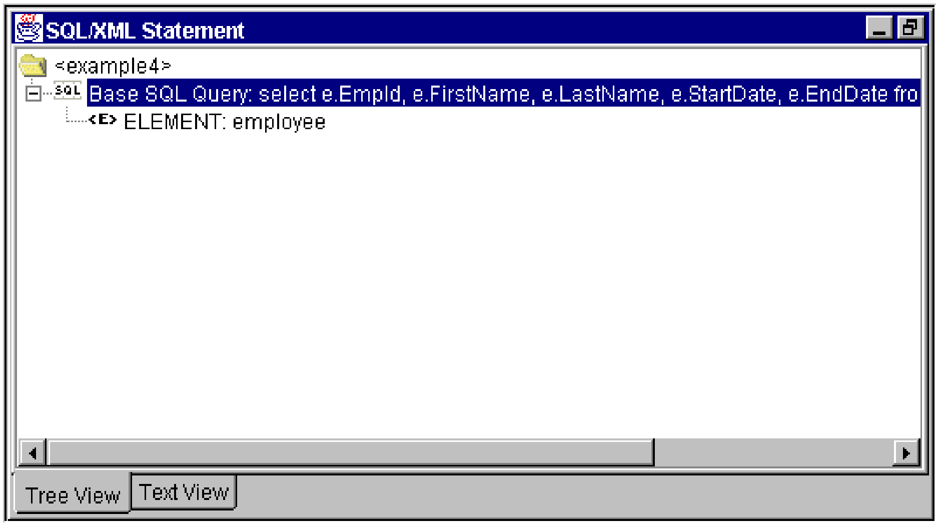
- An attribute named id that retrieves the employee ID (a key) from the database column e.EmpId in the database table Employees with an alias of e.
- An element named names that contains two subelements that retrieve the first and last names of the employee from the database columns e.Firstname and e.LastName in the database table Employees with an alias of e.
- An element named hire-dates that contains two attributes that retrieve the start hire date and end hire date in the database table Employees with an alias of e.

4 Right-click the Base SQL Query node, and select **Insert / Element / Empty**. The Element Node dialog box appears.



5 In the Name field, type `employee`.

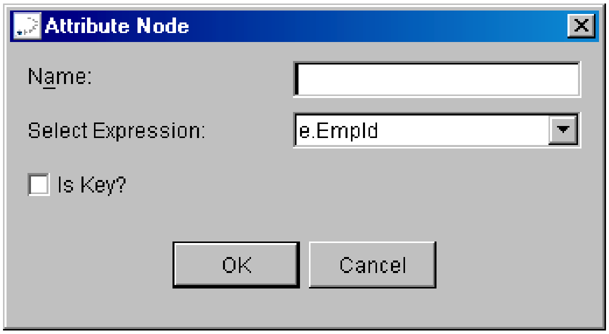
6 Click **OK**. The ELEMENT node appears in the project tree.



SQL/XML Statement
<example4>
Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
ELEMENT: employee
Tree View
Text View

TIP: Because keys uniquely identify rows in the database, we recommend that you use keys when possible to facilitate the performance of data retrieval.

7 Now, add an attribute key named id that will retrieve the employee ID from the database column e.EmpId. Because this attribute key will be a child of the element named employee, right-click the ELEMENT node named employee, and select **Insert / Attribute node / As Select Expression**. The Attribute Node dialog box appears.



Name:

Select Expression: e.EmpId

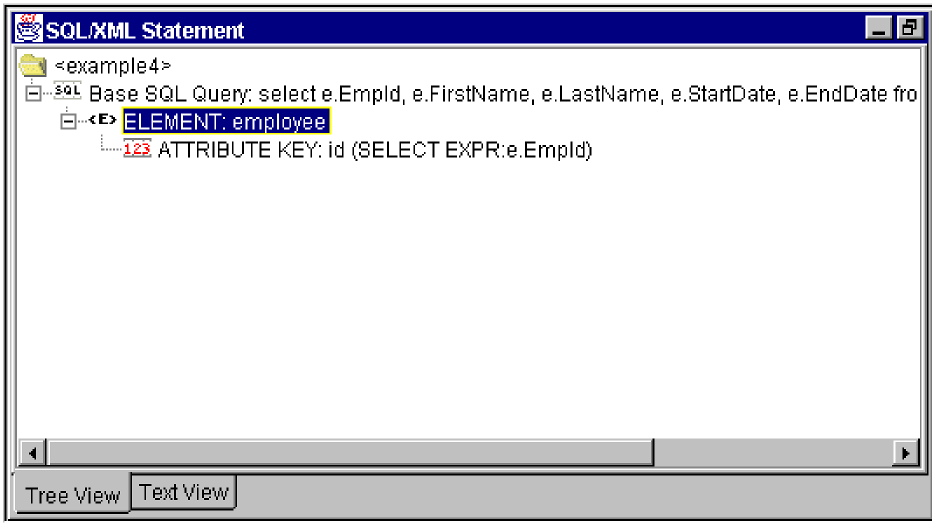
□ Is Key?

8 In the Name field, type `id`.

9 The database column e.EmpId is already specified in the Select Expression drop-down list, so there is no need to select it.

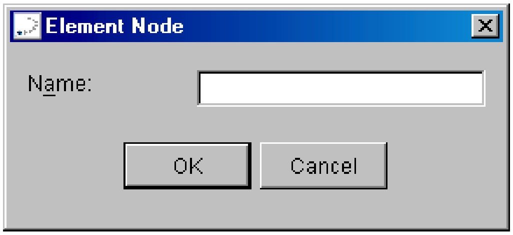
10 Select the **Is Key?** checkbox.

11 Click **OK**. The ATTRIBUTE KEY node appears in the project tree.


SQL/XML Statement

<example4>
Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
ELEMENT: employee
ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)

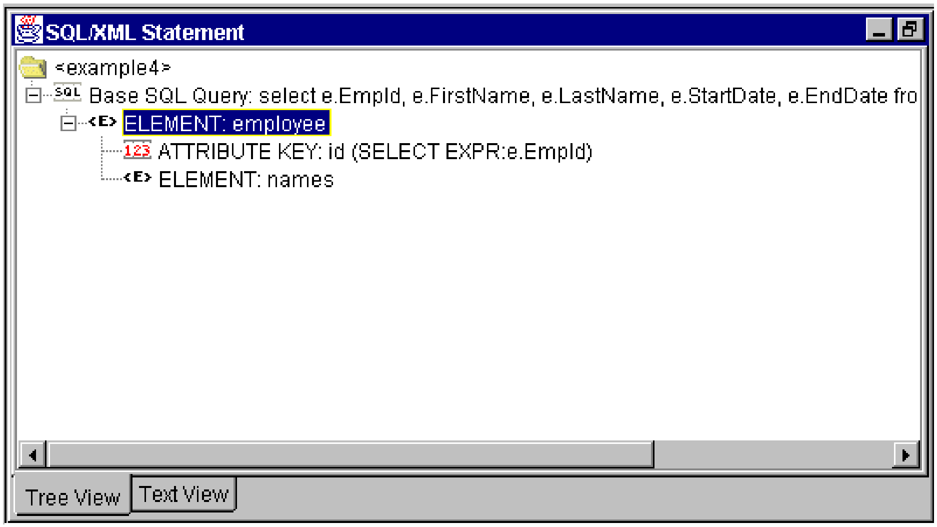
12 Add an element named names that will contain the elements named first and last. Because this element is a child to the element named employee, right-click the ELEMENT node named employees, and select **Insert** / **Element Node** / **Empty**. The Element Node dialog box appears.


Element Node

Name:

13 In the Name field, type `names`.

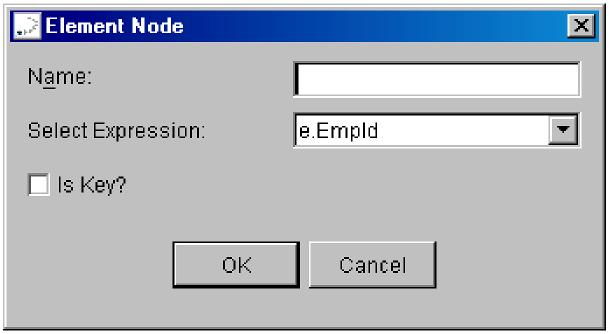
14 Click **OK**. The ELEMENT node appears in the project tree.



SQL/XML Statement

<example4>
Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
ELEMENT: employee
ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
ELEMENT: names

15 Add an element named first that will retrieve the first name of the employee from the database column e.FirstName. Because this element is a child to the element named names, right-click the ELEMENT node named names and select **Insert / Element Node / as Select Expression**. The Element Node dialog box appears.



Element Node

Name:

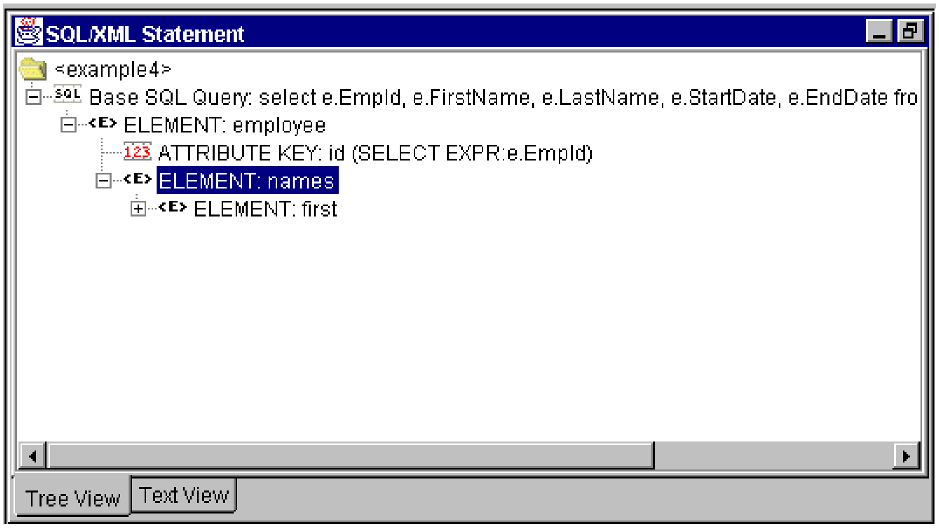
Select Expression: e.EmpId

☐ Is Key?

16 In the Name field, type `first`.

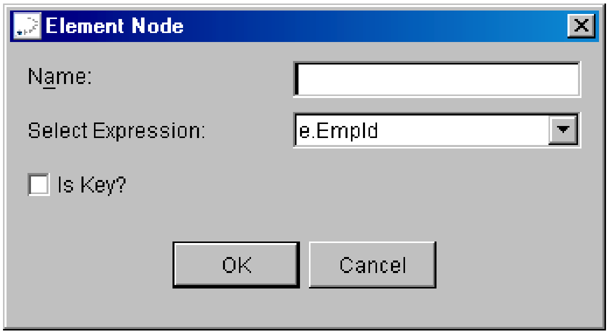
17 From the Select Expression drop-down list, select **e.FirstName**.

18 Click **OK**. The ELEMENT node appears in the project tree.



```
<example4>
  Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
    ELEMENT: employee
      ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
      ELEMENT: names
        ELEMENT: first
```

19 Add an element named last that will retrieve the last name of the employee from the database column e.LastName. Because this element is a child to the element named names, right-click the ELEMENT node named names and select **Insert / Element Node / as Select Expression**. The Element Node dialog box appears.



Name:

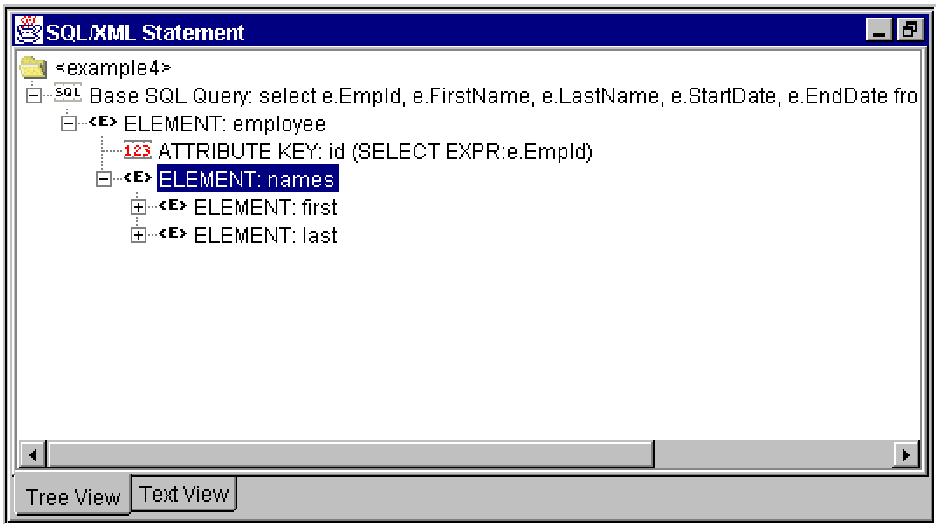
Select Expression: e.EmpId

□ Is Key?

20 In the Name field, type `last`.

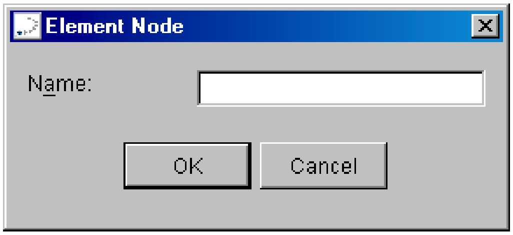
21 From the Select Expression drop-down list, select **e.LastName**.

22 Click **OK**. The ELEMENT node appears in the project tree.



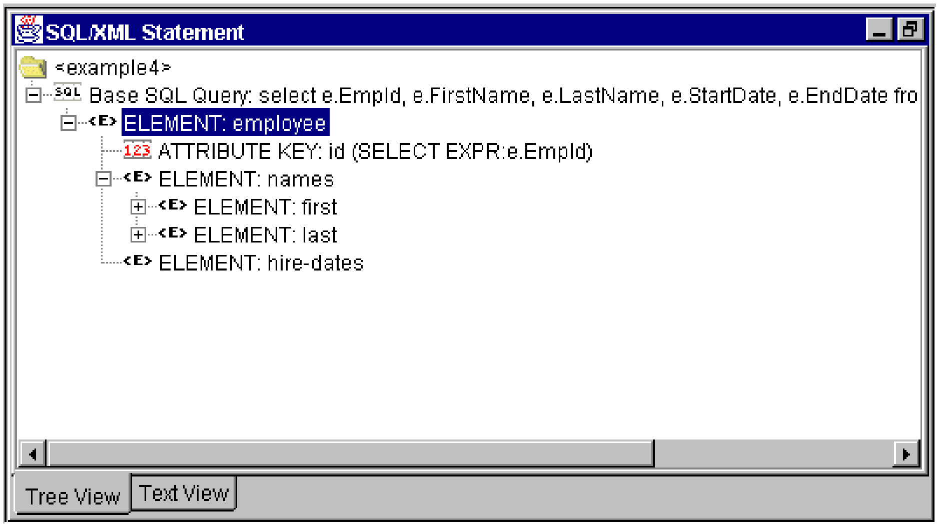
```
<example4>
  Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
    ELEMENT: employee
      ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
      ELEMENT: names
        ELEMENT: first
        ELEMENT: last
```

23 Add an element named hire-dates that will contain the attributes named start and end. Because this element is a child to the element named employee, right-click the ELEMENT node named employees, and select **Insert** / **Element Node** / **Empty**. The Element Node dialog box appears.



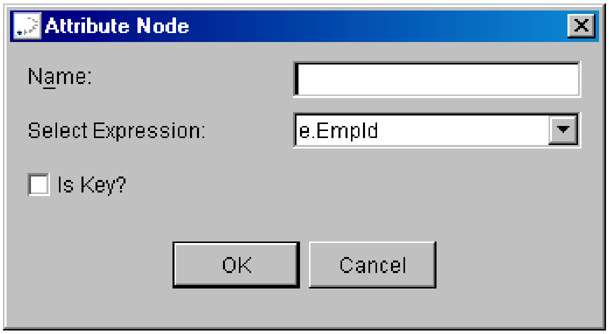
24 In the Name field, type `hire-dates`.

25 Click **OK**. The ELEMENT node appears in the project tree.



```
<example4>
⊟ Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
  ⊟ <E> ELEMENT: employee
      123 ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
    ⊟ <E> ELEMENT: names
      ⊞ <E> ELEMENT: first
      ⊞ <E> ELEMENT: last
      <E> ELEMENT: hire-dates
```

26 Add an attribute named start that will retrieve the start hire date of the employee from the database column e.StartDate. Because this attribute is a child to the element named hire-dates, right-click the ELEMENT node named hire-dates, and select **Insert / Attribute Node / as Select Expression**. The Attribute Node dialog box appears.



Name:

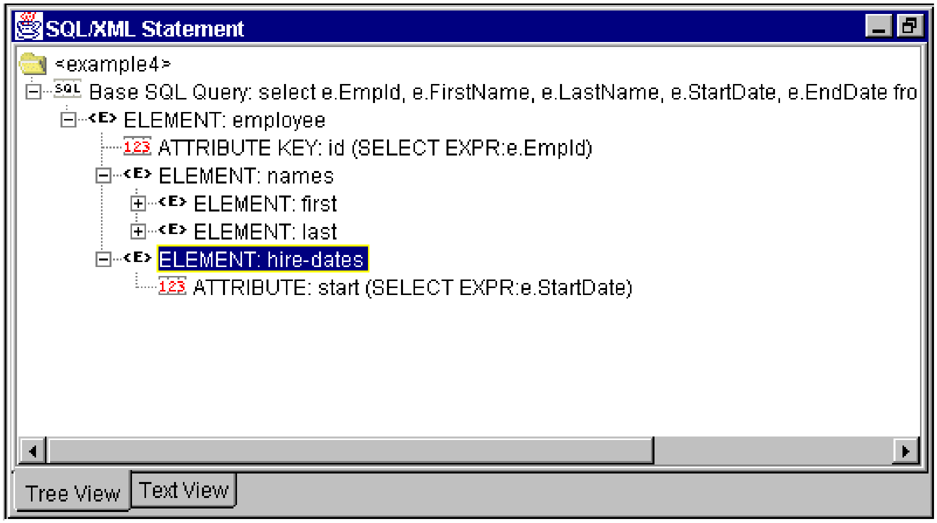
Select Expression: e.EmpId

□ Is Key?

27 In the Name field, type `start`.

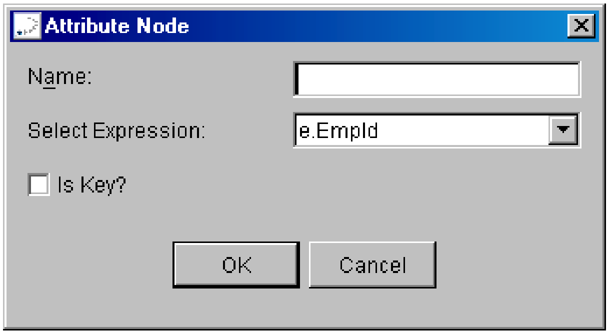
28 From the Select Expression drop-down list, select **e.StartDate**.

29 Click **OK**. The ATTRIBUTE node appears in the project tree.



```
<example4>
 Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
   ELEMENT: employee
     ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
     ELEMENT: names
       ELEMENT: first
       ELEMENT: last
     ELEMENT: hire-dates
       ATTRIBUTE: start (SELECT EXPR:e.StartDate)
```

30 Add an attribute named end that will retrieve the end hire date of the employee from the database column e.EndDate. Because this attribute is a child to the element named hire-dates, right-click the ELEMENT node named hire-dates, and select **Insert / Attribute Node / as Select Expression**. The Attribute Node dialog box appears.



Name:

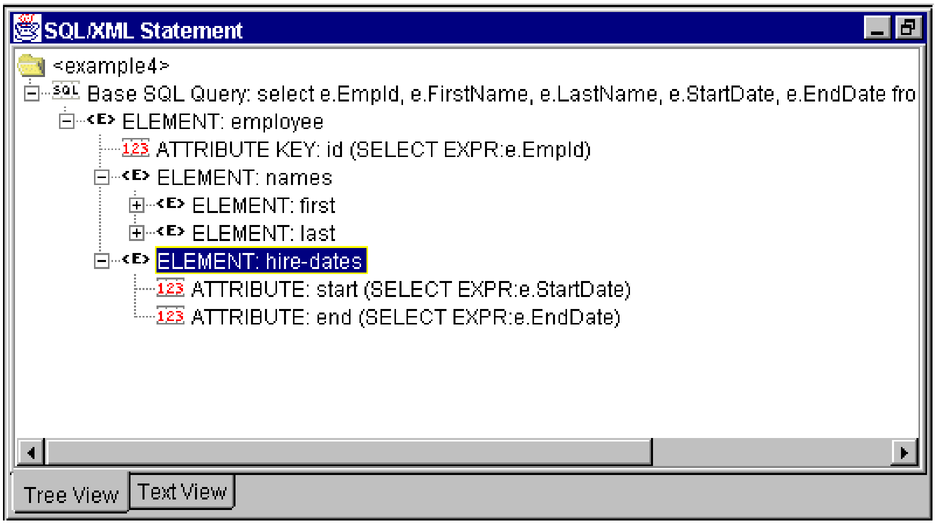
Select Expression: e.EmpId

□ Is Key?

31 In the Name field, type `end`.

32 From the Select Expression drop-down list, select **e.EndDate**.

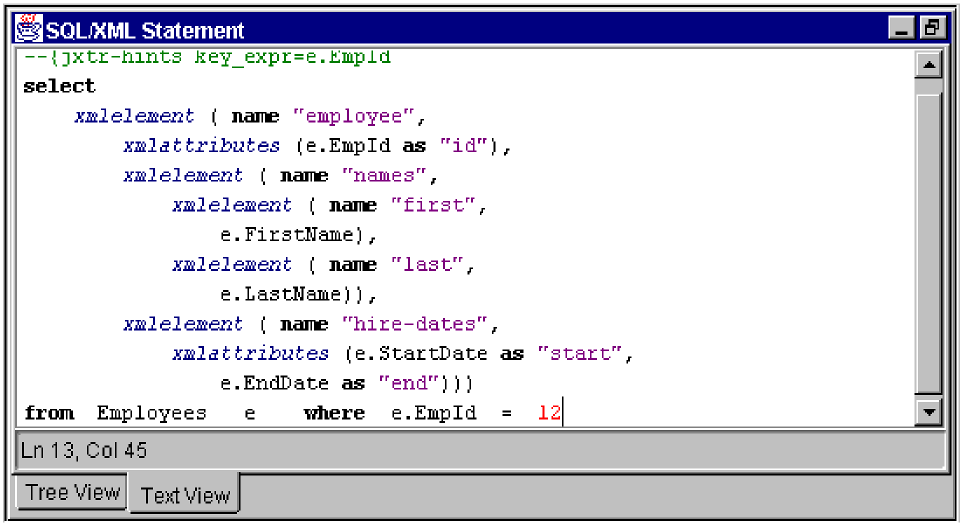
33 Click **OK**. The ATTRIBUTE node appears in the project tree.



```
<example4>
Base SQL Query: select e.EmpId, e.FirstName, e.LastName, e.StartDate, e.EndDate fro
  ELEMENT: employee
    ATTRIBUTE KEY: id (SELECT EXPR:e.EmpId)
    ELEMENT: names
      ELEMENT: first
      ELEMENT: last
    ELEMENT: hire-dates
      ATTRIBUTE: start (SELECT EXPR:e.StartDate)
      ATTRIBUTE: end (SELECT EXPR:e.EndDate)
```

You have completed the part of the query that specifies which data to retrieve and defines the XML structure of the data in the result set.

If you want to see what the SQL/XML query looks like using the SQL/XML syntax, switch to Text view in the Builder by selecting the Text View tab. If the SQL/XML query you created is correct syntactically, the Builder allows you to switch from Tree view to Text view or the reverse.



```
--{jxtr-hints key_expr=e.EmpId
select
    xmlelement ( name "employee",
        xmlattributes (e.EmpId as "id"),
        xmlelement ( name "names",
            xmlelement ( name "first",
                e.FirstName),
            xmlelement ( name "last",
                e.LastName)),
        xmlelement ( name "hire-dates",
            xmlattributes (e.StartDate as "start",
                e.EndDate as "end")))
from  Employees   e    where  e.EmpId  =  12
```

Notice that the key designation we made in our example when we created the ATTRIBUTE node named id has been translated to the Connect *for* SQL/XML hint:

```
--{jxtr-hints key_expr=e.EmpId
```

See "Using a SQL/XML Query in a Java Application" on page 249 to learn how to specify other Connect *for* SQL/XML hints and how to code a SQL/XML query in a Java application.

Using the SQL/XML Query in a Java Application

You can cut and paste the SQL/XML query from the Text view of the Builder into a Java application.

The code example in this section can be found in the SQLXMLExample4.java file in the examples/src/examples/sqlxml directory in your Connect *for* SQL/XML installation directory.

The following code example performs the following tasks:

- Imports the necessary Java classes.
- Creates a connection to the database with the Connect *for* SQL/XML JDBC driver.
- Embeds the SQL/XML query you created in the preceding sections. Notice that each line of the query begins with:

  ```
  sqlxml.append ( "
  ```

  and ends with:

  ```
  ");
  ```

  Also, notice that quotes in a Java string constant must be preceded by the Java escape character \, for example:

  ```
  sqlXml.append ( "   xmlelement(NAME \"hire-dates\", ");
  sqlXml.append ( "     xmlattributes(e.StartDate as
  \"start\",e.EndDate as \"end\") ");
  ```

TIP: Connect *for* SQL/XML hints can also be typed in the Text view of the Builder.

- Provides Connect *for* SQL/XML hints that specify processing options that are not available in the SQL/XML syntax. For example:

  ```
  sqlXml.append ( "--{jxtr-hints null_replacement=
  NA;key_expr=e.EmpId;timestamp_encoding=odbc " );
  ```

The preceding code specifies the following hints:

- null_replacement replaces NULL values retrieved from the database with the letters "NA".
- timestamp_encoding specifies that the odbc timestamp-to-string conversion be used.
- key_expr specifies that the database column e.EmpId be used to uniquely identify the row in the database.

- Sends a description of the query result to System.out.
- Executes the SQL/XML query.
- Sends the result set to System.out.
- Closes the connection to the database with the Connect *for* SQL/XML JDBC driver and declares the main method.

```
package examples.sqlxml;

import java.io.OutputStreamWriter;
import java.sql.*;

import com.ddtek.jdbc.jxtr.*;
import examples.*;

public class SQLXMLExample4 extends Example
{

    /**
     * Executes the 'SQL/XML Example4' example
     */
    public void execute() throws Exception
    {
        // Output example description
        System.out.println( "=================================" );
        System.out.println( "= SQL/XML Example4 demonstrates =" );
        System.out.println( "=================================" );
        System.out.println( "  (a) Specifying null replacement hint." );
        System.out.println( "  (b) Specifying timestamp encoding hint." );
        // Load properties from the resource
        loadProperties();

        // Create the Connect for SQL/XML JDBC connection
        connectWithConnectForSQLXML();

        // Build SQL/XML query
        StringBuffer sqlXml = new StringBuffer();
        sqlXml.append ( "--{jxtr-hints null_replacement=NA;key_expr=
e.EmpId;timestamp_encoding=odbc " );
        sqlXml.append ( lineSeparator );
        sqlXml.append ( "select " );
        sqlXml.append ( "  xmlelement (NAME \"employee\", " );
        sqlXml.append ( "    xmlattributes(e.EmpId as \"id\"), " );
        sqlXml.append ( "    xmlelement(NAME \"names\", " );
        sqlXml.append ( "      xmlelement(NAME \"first\", e.FirstName), " );
        sqlXml.append ( "      xmlelement(NAME \"last\", e.LastName)), " );
        sqlXml.append ( "    xmlelement(NAME \"hire-dates\", " );
        sqlXml.append ( "      xmlattributes(e.StartDate as
\"start\",e.EndDate as \"end\") ");
        sqlXml.append ( "  )) as SQLXMLCOL1 " );
        sqlXml.append ( "from Employees e where e.EmpId = 12 " );

        // Output result header
        System.out.println( "---------------------------------------" );
        System.out.println( "Query result (via getString on XMLType)" );
        System.out.println( "---------------------------------------" );

        // Execute the SQL/XML query
        resultSet = stmt.executeQuery(sqlXml.toString());

        // Retrieve data from the resultset
        while ( resultSet.next() )
        {
            XMLType xmlType = (XMLType)resultSet.getObject(1);
            System.out.println(xmlType.getString());
        }
        resultSet.close();

        // Close Connect for SQL/XML JDBC connection
        disconnect();
```

```
    }

    /**
     * Main method.
     */
    public static void main ( String[] args ) throws Exception
    {
        Example thisDemo = new SQLXMLExample4();
        thisDemo.execute();
    }


}
```

10 Tutorial: Using jXTransformer Queries

This chapter contains a step-by-step tutorial that shows you how to create a jXTransformer query using the DataDirect Query Builder *for* SQL/XML (the Builder). It also shows you how to embed the query in a Java using the jXTransformer API.

This tutorial steps you through the process of constructing the following jXTransformer query, which logically can be divided into two parts: the first part specifies document-level constructs and the second part specifies the data to be retrieved and defines the structure of the XML document.

Specifies document-level constructs

```
xml_document(
   xml_comment('jxtr Example4 - Part 1'),
   xml_external_dtd('example4.dtd'),
   xml_pi('xml-stylesheet', 'type="text/xsl"
   href="file://example4.xsl"'),
   xml_element('exns:example4',
      xml_attribute('rootatt1','example4'),
      xml_namespace('http://www.jxtrdemo/default'),
      xml_namespace('exns','http://www.jxtrdemo/example4'),
```

Specifies data to be retrieved and defines the structure of the XML document

```
      SELECT
         xml_element('empinfo',
            xml_attribute_key('exns:id',e.EmpId ),
            xml_attribute('exns:name',e.LastName),
            (SELECT
               xml_element('project',
                  xml_attribute('name',p.Name),
                  xml_attribute('task',a.Task))
            FROM Projects p, Assignments a
            WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId) )
   FROM Employees e WHERE e.EmpId between ? and ?) )
```

NOTE: The preceding jXTransformer query example can be found in the JXTRExample4.java file in the examples/src/examples/jxtrapi directory in your Connect *for* SQL/XML installation directory. For information on populating your database with test data and using this example, refer to the examples_readme.txt file in the examples directory.

Our jXTransformer query example generates an XML document that looks like this:

```
<!--jxtr Example4-->
<!DOCTYPE exns:example4 SYSTEM "example4.dtd">
<?xml-stylesheet type="test/xsl" href="file://example4.xsl"?>
<exns:example4
rootatt1='example4'
xmlns="http://www.jxtrdemo/default"
xmlns=exns="http://www.jxtrdemo/example4">
   <empinfo exns:id='1' name='Marshall'>
     <project name='Medusa' task='Analysis'></project>
     <project name='Medusa' task='Documentation'></project>
     <project name='Medusa' task='Planning'></project>
     <project name='Medusa' task='Testing'></project>
     <project name='Phoenix' task='Analysis'></project>
     <project name='Phoenix' task='Documentation'></project>
   </empinfo>
   <empinfo exns:id='2' exns:name='Ayers'>
     <project name='Hydra' task='Analysis'></project>
     <project name='Hydra' task='Documentation'></project>
     <project name='Python' task='Analysis'></project>
     <project name='Python' task='Development'></project>
   </empinfo>
   <empinfo exns:id='3' exns:name='Simpson'>
     <project name='Pegasus' task='Analysis'></project>
     <project name='Pegasus' task='Testing'></project>
   </empinfo>
   <empinfo exns:id='4' exns:name='O&apos;Donnel'>
     <project name='Centaur' task='Analysis'></project>
     <project name='Centaur' task='Documentation'></project>
     <project name='Centaur' task='Planning'></project>
   </empinfo>
```

```
<empinfo exns:id='5' exns:name='Jenkins'>
  <project name='Centaur' task='Analysis'></project>
  <project name='Centaur' task='Testing'></project>
</empinfo>
</exns:example4>
```

This tutorial uses this jXTransformer query example to demonstrate how to accomplish the following tasks:

- Create XML documents that contain document-level processing instructions, comments, namespaces, an external DTD reference, and a root element. First, the tutorial will show you how to create document-level constructs by specifying them within the query. Later, it will show how to specify them using the jXTransformer API. The advantage of setting document-level constructs through the jXTransformer API is that you can make changes to them without editing the query.
- Use keys to facilitate performance of data retrieval. Keys uniquely identify each row selected by the base SQL query.
- Use a nested query within a jXTransformer query. In our jXTransformer query example, the following part of the jXTransformer query example is a nested query:

  ```
  (SELECT
     xml_element('project',
        xml_attribute('name',p.Name),
        xml_attribute('task',a.Task))
  FROM Projects p, Assignments a
  WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
  ```

  For the purposes of this tutorial, we refer to the nested query as the *child query* and the query that contains the nested query as the *parent query*.
- Use parameter markers in the jXTransformer query and set the values for those markers using the jXTransformer API. Setting values for parameter markers can be done only through the jXTransformer API.

- Generate a DTD that describes the XML that results from the jXTransformer query using the jXTransformer API.

Complete the tasks described in the next sections to create the jXTransformer query in our example. These tasks include:

- "Creating the Builder Project" on page 256
- "Constructing the Parent Query in the Builder" on page 259
- "Constructing the Child Query in the Builder" on page 266
- "Specifying Document-Level Constructs in the jXTransformer Query" on page 273
- "Using a jXTransformer Query in a Java Application" on page 286

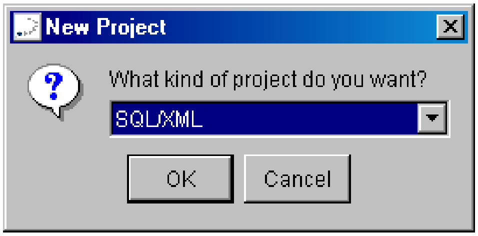
Creating the Builder Project

1 Start the Builder. How you start the Builder depends on your platform:

   - **On Windows**: Run the builder.bat file located in the Connect *for* SQL/XML installation directory.
   - **On UNIX**: Run the builder.sh shell script located in the Connect *for* SQL/XML installation directory.

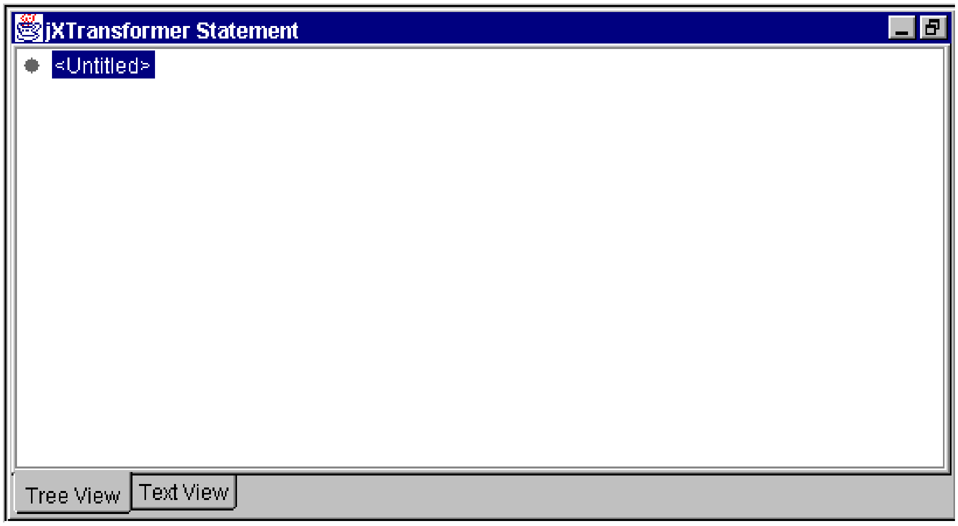
   During a normal installation, the builder.bat file and builder.sh script are automatically customized to include the path to your JDK. If the installer was unable to detect a required JDK or you want to change the JDK to be used, refer to the *DataDirect Connect for SQL/XML Installation Guide* for instructions on configuring the startup file for the Builder.

2 Open a new Builder project for the jXTransformer query by selecting **File / New Project**. The New Project dialog box appears.



3 From the drop-down list, select **jXTransformer**.

4 Click **OK**. An untitled project node appears in the Tree view of the jXTransformer Statement window.



5 Save the jXTransformer query project by selecting **File / Save Project As** and specifying `example4.jxb` as the filename of the project. Notice that the project node is renamed to example4. As you work with this example, you can save your changes to the project file at any time by selecting **File / Save Project**.

Our jXTransformer query example contains a nested, or child, query as shown in the following example.

Parent query

Child query

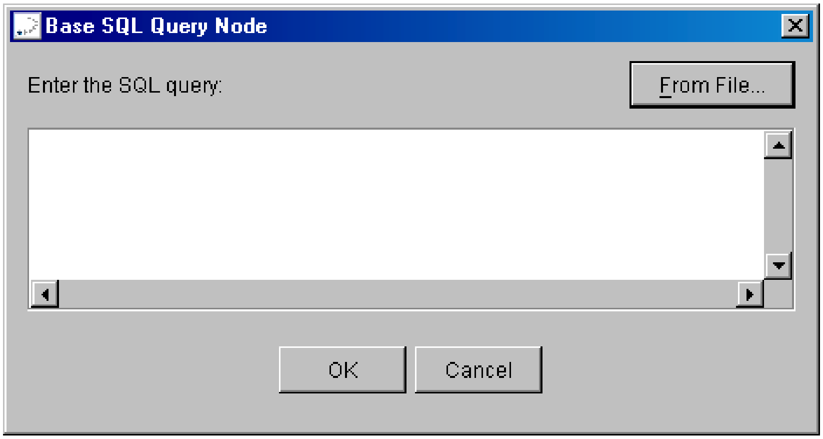
```
SELECT
   xml_element('empinfo',
      xml_attribute_key('exns:id',e.EmpId ),
      xml_attribute('exns:name',e.LastName),
      (SELECT
         xml_element('project',
            xml_attribute('name',p.Name),
            xml_attribute('task',a.Task))
      FROM Projects p, Assignments a
      WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
   FROM Employees e WHERE e.EmpId between ? and ?
```

As you build the query in the example, you will construct the parent query first. Then, you will construct the child query.

NOTE: Typically, you would construct the part of the query that represents the structure of the XML document first, and then, insert document-level constructs in the query. If you prefer, however, you can insert document-level constructs first (see "Specifying Document-Level Constructs in the jXTransformer Query" on page 273 for instructions). Or, you can set the document-level constructs in the jXTransformer API (see "Using a jXTransformer Query in a Java Application" on page 286 for instructions).

Constructing the Parent Query in the Builder

1. First, create a Base SQL Query for the parent query. The Base SQL Query is required and helps define the data to be retrieved. It also facilitates the process of constructing the query. To create a Base SQL Query, right-click the project node and select **Insert** / **Base SQL Query Node**. The Base SQL Query Node dialog box appears.



2. Type the following Base SQL Query for the parent query:

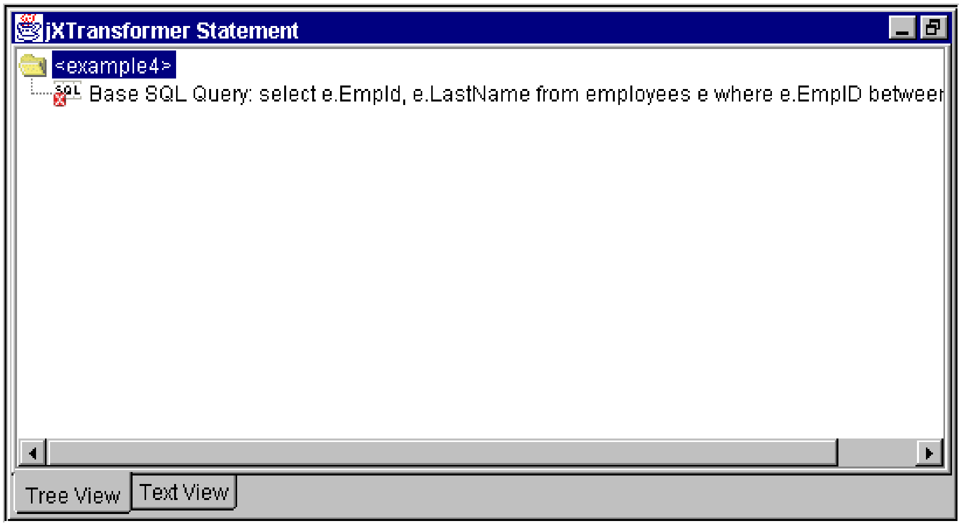
```
SELECT
   e.EmpId,
   e.LastName
FROM Employees e WHERE e.EmpId between  ?  and  ?
```

The Base SQL Query accomplishes the following tasks:

- Selects columns in the parent query containing data to retrieve
- Specifies tables from which to retrieve data
- Optionally, specifies filters on the Select statement

Notice that the Where clause in our example specifies a filter on the Select statement of the parent query and contains parameter markers. Parameter markers are placeholders for values, represented by question marks (?). Later, this tutorial will show you how to set the values of the parameter markers using the jXTransformer API. See "Using a jXTransformer Query in a Java Application" on page 286 for instructions.

3 Click **OK**. The Builder checks the Base SQL Query for syntax. The Base SQL Query node appears in the project tree.



jXTransformer Statement
<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
Tree View
Text View

Now that you have defined which data to retrieve from the database for the parent query, you can define the XML structure, including the hierarchical relationships of the data.

So, examine the parent query in our example. You need to create an XML element that contains related employee information, including employee ID, last name of the employee, and basic project information for that employee.

Parent query

Child query

```
SELECT
  xml_element('empinfo',
     xml_attribute_key('exns:id',e.EmpId ),
     xml_attribute('exns:name',e.LastName),
     (SELECT
        xml_element('project',
           xml_attribute('name',p.Name),
           xml_attribute('task',a.Task))
     FROM Projects p, Assignments a
     WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
  FROM Employees e WHERE e.EmpId between ? and ?
```

Notice that the element named empinfo in the parent query is a parent to the following attributes and nested query:

TIP: When specifying table names, you must use unique table aliases (for example, Employees e).

- An attribute named exns:id that retrieves the employee ID (a key) from the database column e.EmpId in the database table Employees with an alias of e.
- An attribute named exns:name that retrieves the last name of the employee from the database column e.LastName in the database table Employees with an alias of e.

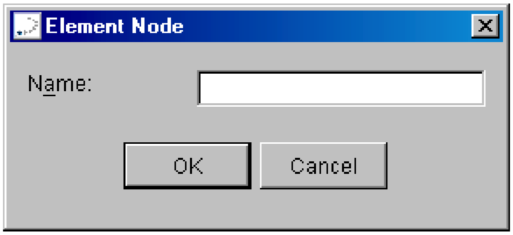
  NOTE: In our query example, the attribute names exns:id and exns:name include a namespace prefix (exns) which is preceded by a colon(:). The namespace in our example is specified using a jXTransformer document-level construct. See “Specifying Document-Level Constructs in the jXTransformer Query” on page 273 and “Using a jXTransformer Query in a Java Application” on page 286 for instructions.
- A nested query that retrieves basic project information for the employee, such as the name of the project and the task the employee worked on for that project, from

two different database tables, Projects with an alias of p and Assignments with an alias of a.

```
(SELECT
   xml_element('project',
      xml_attribute('name',p.Name),
      xml_attribute('task',a.Task))
FROM Projects p, Assignments a
WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
```

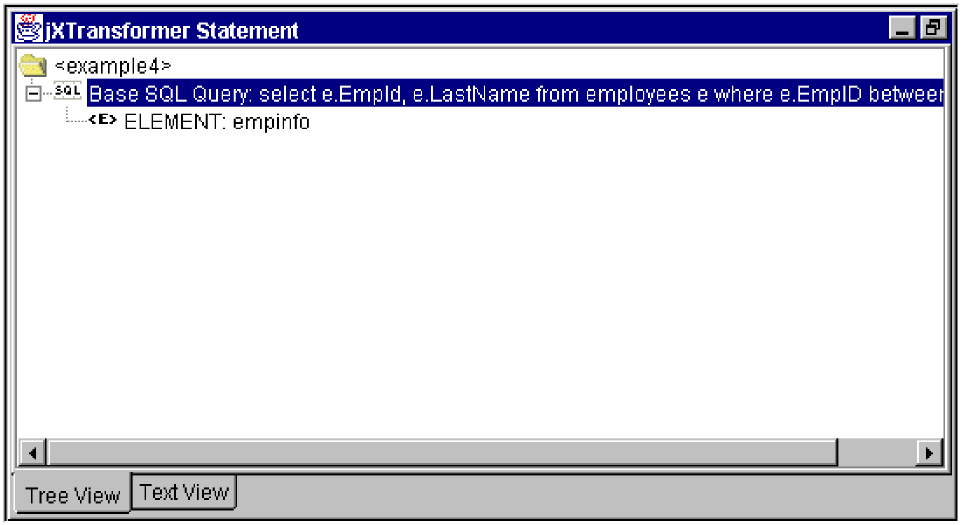
Notice that the Where clause in the nested query creates a link between the database tables, Projects and Assignments and Assignments and Employees.

4 Because the element only contains child constructs, you can create an ELEMENT node named empinfo with an empty value. Right-click the Base SQL Query node, and select **Insert / Element / Empty**. The Element Node dialog box appears.



5 In the Name field, type `empinfo`.

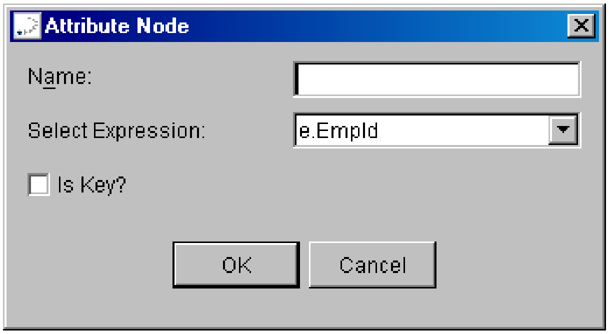
6 Click **OK**. The ELEMENT node appears in the project tree.



jXTransformer Statement
<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betwee
ELEMENT: empinfo
Tree View
Text View

TIP: Because keys uniquely identify rows in the database, we recommend that you use keys when possible to facilitate the performance of data retrieval.

7 Now, add an attribute key named exns:id that will retrieve the employee ID from the database column e.EmpId. Because this attribute key will be a child of the element named empinfo, right-click the ELEMENT node named empinfo. Select **Insert** / **Attribute node** / **As Select Expression**. The Attribute Node dialog box appears.



Name:

Select Expression: e.EmpId

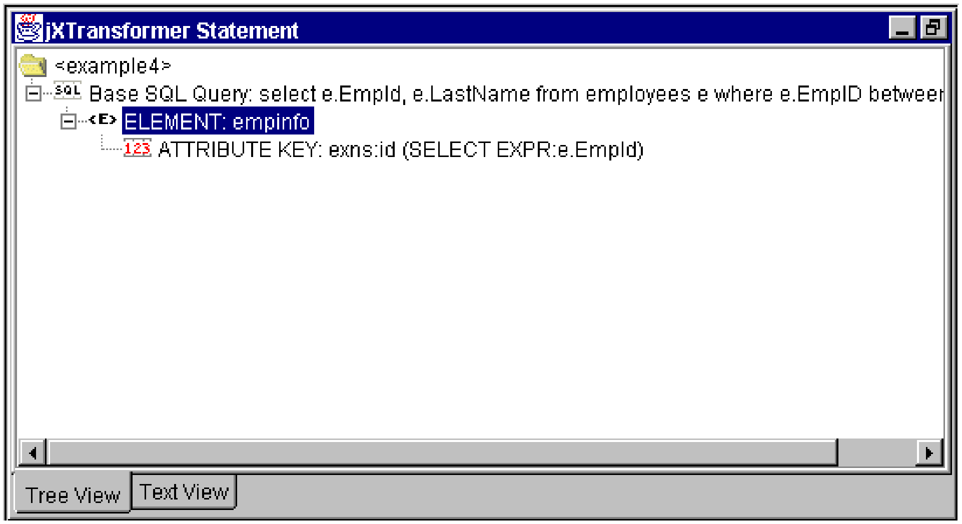
□ Is Key?

8 In the Name field, type `exns:id`.

9 The database column e.EmpId is already specified in the Select Expression drop-down list, so there is no need to select it.

10 Select the **Is Key?** checkbox.

11 Click **OK**. The ATTRIBUTE KEY node appears in the project tree.



jXTransformer Statement

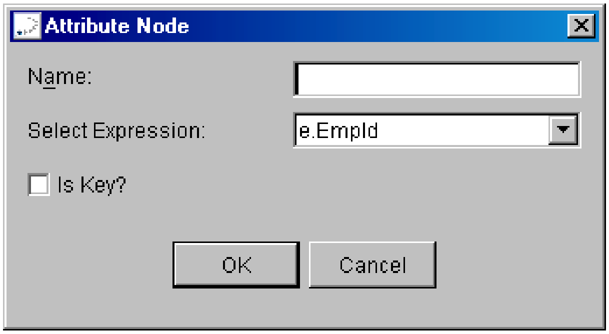
<example4>

Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer

ELEMENT: empinfo

ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)

12 Next, add an attribute named exns:name that will retrieve the last name of the employee from the database column e.LastName. Because this attribute will be a child of the element named empinfo, right-click the ELEMENT node named empinfo. Select **Insert / Attribute node / As Select Expression**. The Attribute Node dialog box appears.



Attribute Node

Name:

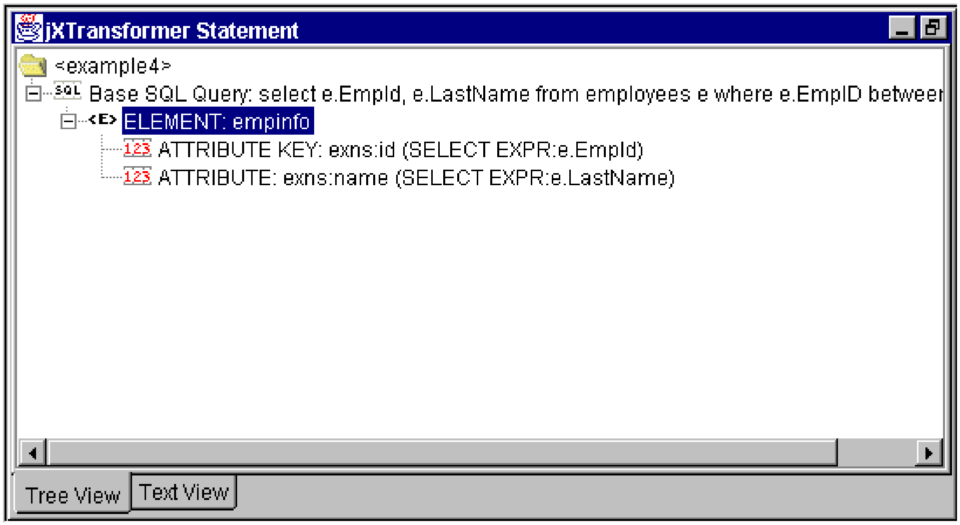
Select Expression: e.EmpId

□ Is Key?

13 In the Name field, type `exns:name`.

14 From the Select Expression drop-down list, select **e.LastName**.

15 Click **OK**. The ATTRIBUTE node appears in the project tree.



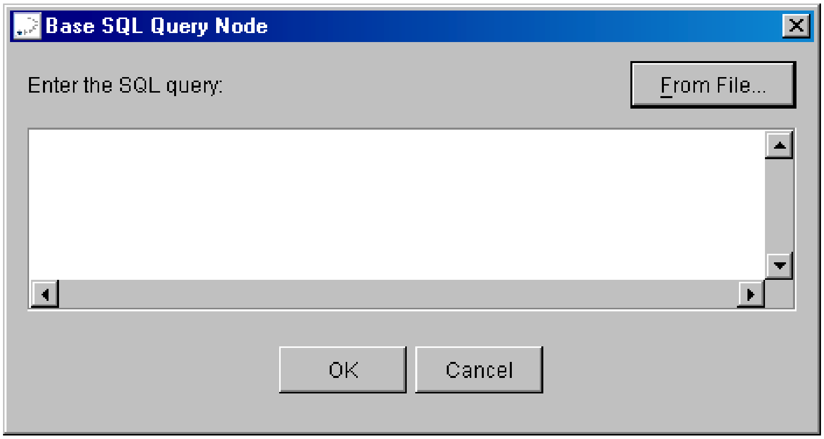
jXTransformer Statement

```
<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
  ELEMENT: empinfo
    ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
    ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
```

Now, you have constructed the parent query and can construct the nested child query as described in the next section.

Constructing the Child Query in the Builder

1 To construct the child query, create another Base SQL Query that is a child of the element named empinfo. To do this, right-click the ELEMENT node named empinfo. Select **Insert** / **Base SQL Query**. The Base SQL Query Node dialog box appears.



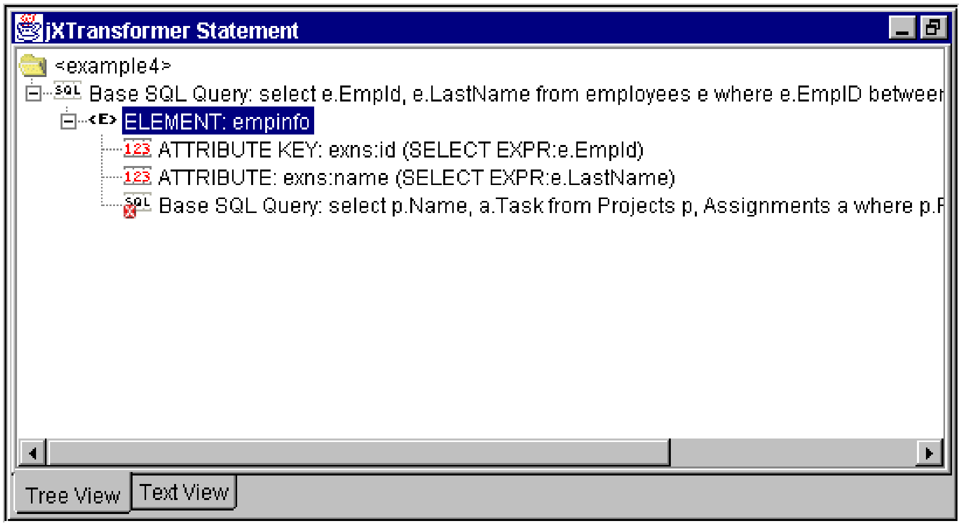
2 Type the following Base SQL Query for the child query:

```
SELECT
   p.Name,
   a.Task
FROM Projects p, Assignments a
WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId
```

Remember that the Base SQL Query accomplishes the following tasks:

- Selects columns in the parent query containing data to retrieve
- Specifies tables from which to retrieve data
- Optionally, specifies filters on the Select statement

3 Click **OK**. The Builder checks the Base SQL Query for syntax. The Base SQL Query node appears in the project tree.



<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
ELEMENT: empinfo
ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
Base SQL Query: select p.Name, a.Task from Projects p, Assignments a where p.F

Now that you have defined which data to retrieve from the database for the child query, you can now define the XML structure, including the hierarchical relationships of the data.

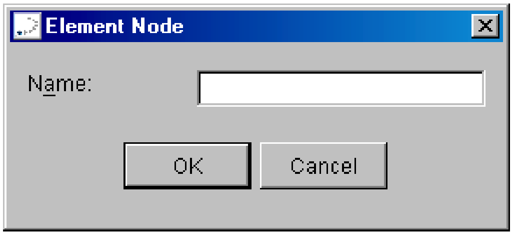
Examine the child query in our example as shown in the following code. You need to create an element that contains basic project information for the employee, including the name of the project and the task the employee worked on for that project.

```
(SELECT
   xml_element('project',
       xml_attribute('name',p.Name),
       xml_attribute('task',a.Task))
 FROM Projects p, Assignments a
 WHERE p.ProjId=a.ProjId and a.EmpId=e.EmpId))
```

Notice that the element named project is a parent to the following attributes:

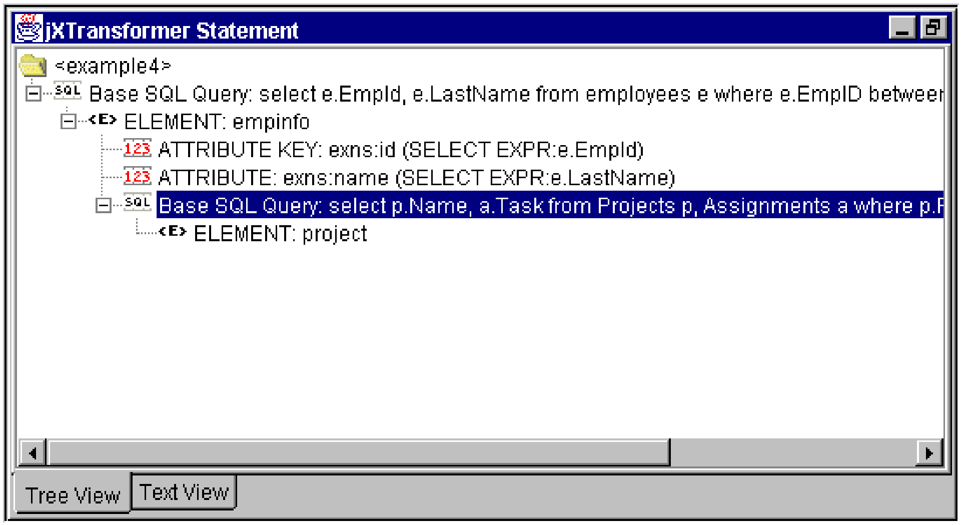
- An attribute named name that retrieves the name of the project the employee worked on from the database column p.Name in the database table Projects with an alias of p
- An attribute named task that retrieves the task the employee worked on from the database column a.Task in the database table Assignments with an alias of a

4 Because the element only contains child constructs, you can create an ELEMENT node named project with an empty value. To do this, right-click the Base SQL Query node, and select **Insert / Element Node / Empty**. The Element Node dialog box appears.



5 In the Name field, type `project`.

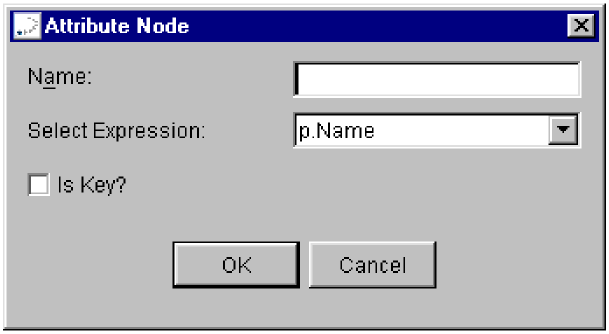
6 Click **OK**. The ELEMENT node appears in the project tree.



jXTransformer Statement

<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
ELEMENT: empinfo
ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
Base SQL Query: select p.Name, a.Task from Projects p, Assignments a where p.F
ELEMENT: project

7 Now, add an attribute named name that will retrieve the name of the project from the database column p.Name. Because this attribute will be a child of the element named project, right-click the ELEMENT node named project. Select **Insert / Attribute node / As Select Expression**. The Attribute Node dialog box appears.



Attribute Node

Name:

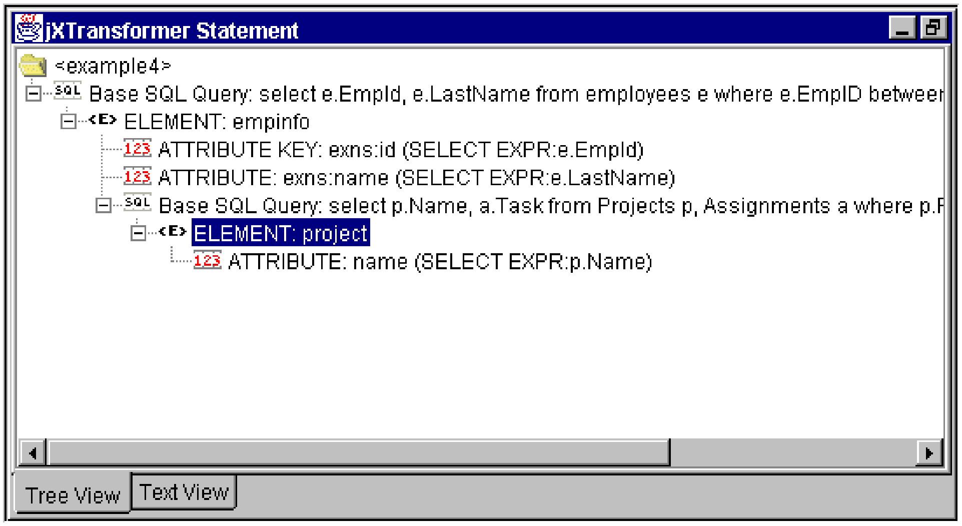
Select Expression: p.Name

□ Is Key?

8 In the Name field, type `name`.

9 The database column p.Name is already selected in the Select Expression drop-down list, so there is no need to select it.

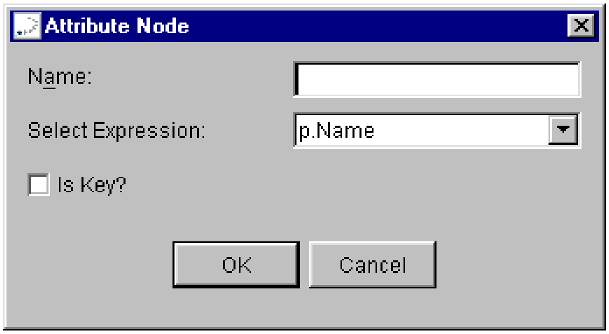
10 Click **OK**. The ATTRIBUTE node appears in the project tree.



jXTransformer Statement

```
<example4>
  Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
    ELEMENT: empinfo
      ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
      ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
      Base SQL Query: select p.Name, a.Task from Projects p, Assignments a where p.F
        ELEMENT: project
          ATTRIBUTE: name (SELECT EXPR:p.Name)
```

11 Next, add an attribute named task that retrieves the tasks the employee worked on from the database column a.Task. Because this attribute will be a child of the element named project, right-click the ELEMENT node named project. Select **Insert / Attribute Node / As Select Expression**. The Attribute Node dialog box appears.



Attribute Node

Name:

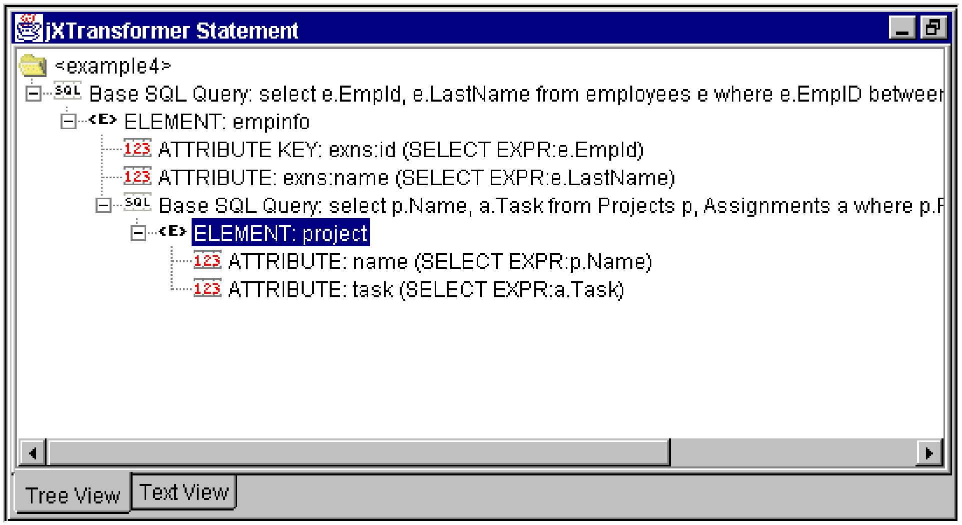
Select Expression: p.Name

□ Is Key?

12 In the Name field, type `task`.

13 In the Select Expression drop-down list, select **a.Task**.

14 Click **OK**. The ATTRIBUTE node appears in the project tree.



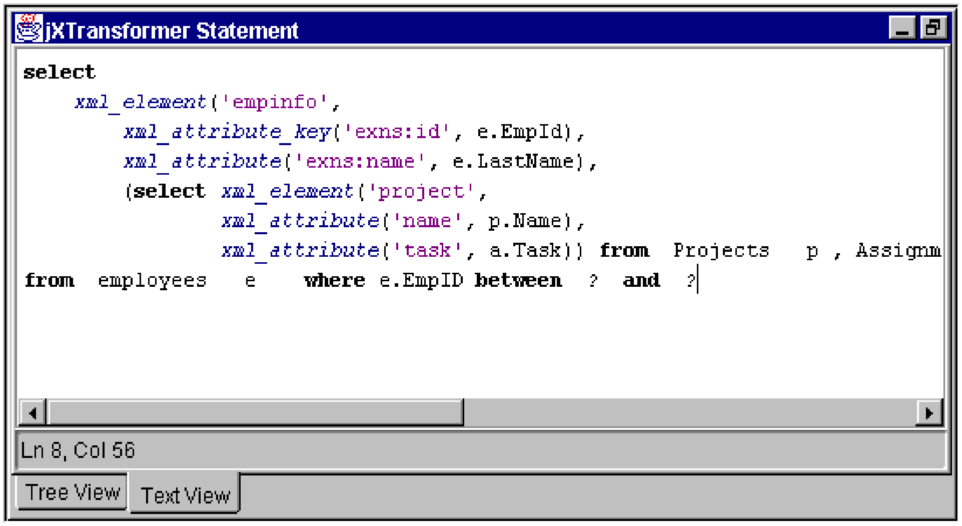
jXTransformer Statement

```
<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
  ELEMENT: empinfo
    ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
    ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
    Base SQL Query: select p.Name, a.Task from Projects p, Assignments a where p.F
      ELEMENT: project
        ATTRIBUTE: name (SELECT EXPR:p.Name)
        ATTRIBUTE: task (SELECT EXPR:a.Task)
```

15 Save the jXTransformer query by selecting **File / Save Project**.

You have completed the part of the query that specifies which data will be retrieved and defines the XML structure of the resulting XML document.

If you want to see what the jXTransformer query looks like using the jXTransformer syntax, switch to Text view in the Builder by selecting the Text View tab. If the jXTransformer query you created is correct syntactically, the Builder allows you to switch from Tree view to Text view or the reverse.

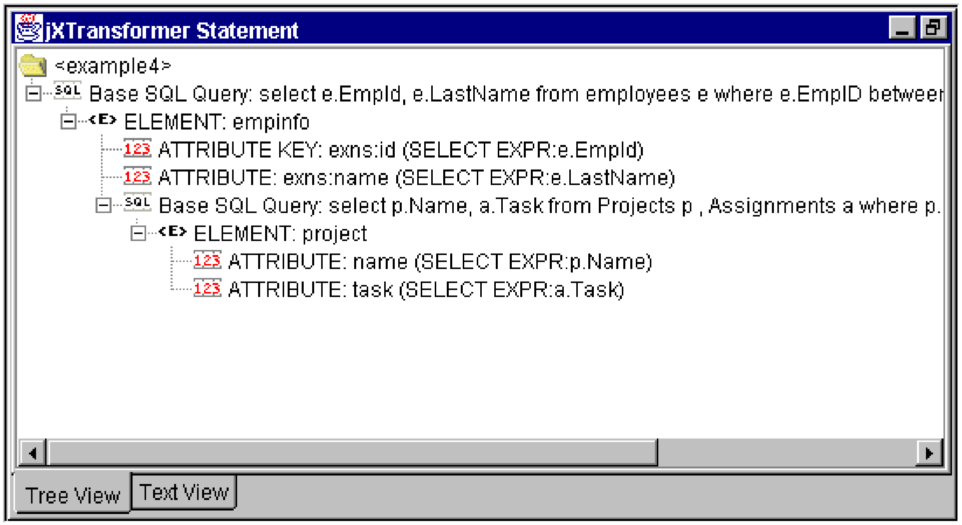


```
select
    xml_element('empinfo',
        xml_attribute_key('exns:id', e.EmpId),
        xml_attribute('exns:name', e.LastName),
        (select xml_element('project',
                xml_attribute('name', p.Name),
                xml_attribute('task', a.Task)) from  Projects   p , Assignm
from  employees   e    where e.EmpID between  ?  and  ?
```

To learn how to specify document-level constructs within the query, you can continue with the next section. Or, you can specify document-level constructs within the Java application (see "Using a jXTransformer Query in a Java Application" on page 286 for instructions).

Specifying Document-Level Constructs in the jXTransformer Query

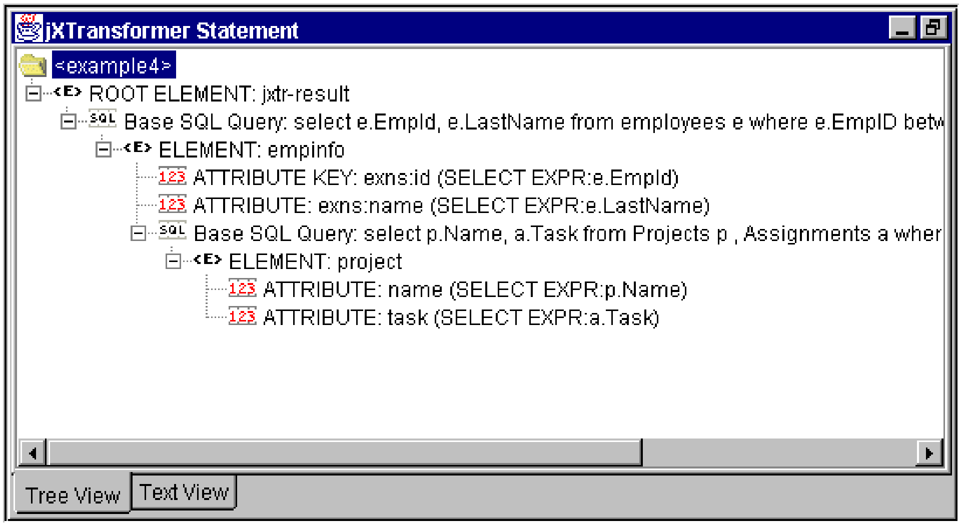
If the Builder project you created for the jXTransformer query in the previous section is not already open, open it by selecting **File / Open Project**. Navigate to the project file named example4.jxb and select it. The project appears either in Text view or Tree view, depending on the Builder view in which it was last saved. If the project appears in Text view, switch to Tree view by selecting the Tree View tab on the jXTransformer Statement window.



```
<example4>
Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betweer
  ELEMENT: empinfo
    ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
    ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
    Base SQL Query: select p.Name, a.Task from Projects p , Assignments a where p.
      ELEMENT: project
        ATTRIBUTE: name (SELECT EXPR:p.Name)
        ATTRIBUTE: task (SELECT EXPR:a.Task)
```

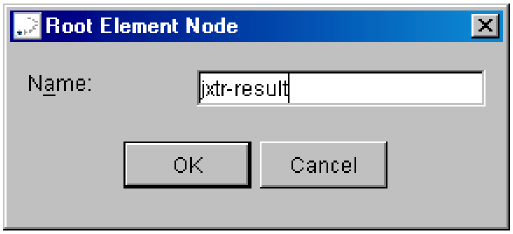
Adding the Root Element

1 Insert a root element by selecting **Insert / Document Header.** A ROOT ELEMENT node is created in the project tree with a default root element name of jxtr-result.



```
<example4>
ROOT ELEMENT: jxtr-result
  Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
    ELEMENT: empinfo
      ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
      ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
      Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
        ELEMENT: project
          ATTRIBUTE: name (SELECT EXPR:p.Name)
          ATTRIBUTE: task (SELECT EXPR:a.Task)
```

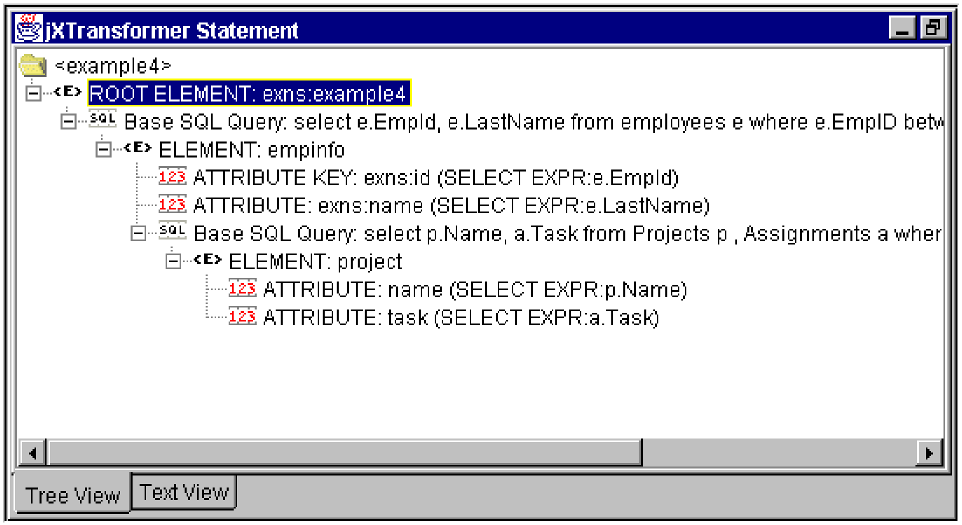
2 Because we will name the root element exns:example4, right-click the ROOT ELEMENT node and select **Edit**. The Root Element Node dialog box appears with jxtr-result in the Name field.



Name: jxtr-result

3 Highlight **jxtr-result** In the Name field, and type `exns:example4`.

4 Click **OK**. The name of the ROOT ELEMENT node in the project tree is changed to exns:example4.



jXTransformer Statement

<example4>

ROOT ELEMENT: exns:example4

Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw

ELEMENT: empinfo

ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)

ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)

Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher

ELEMENT: project

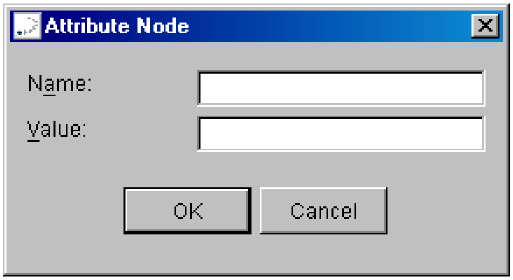
ATTRIBUTE: name (SELECT EXPR:p.Name)

ATTRIBUTE: task (SELECT EXPR:a.Task)

Notice that the root element contains an attribute named rootatt1and two namespace definitions. The first namespace definition defines the default namespace; the second defines a namespace with a prefix ID of exns.

Adding an Attribute to the Root Element

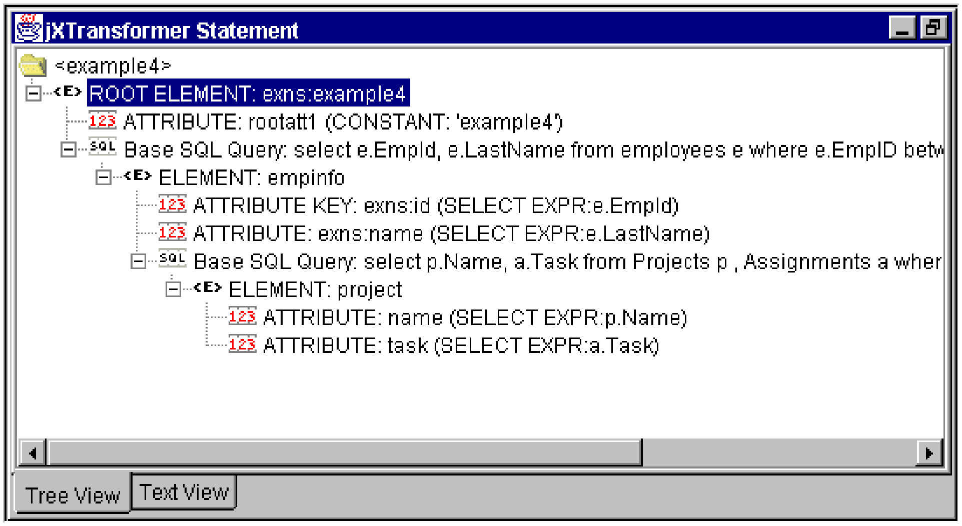
1 Add an attribute named rootatt1 to the root element. Select the ROOT ELEMENT node, and select **Insert** / **Attribute Node** / as **Constant**. The Attribute Node dialog box appears.



2 In the Name field, type `rootatt1`.

3 In the Value field, type `example4`.

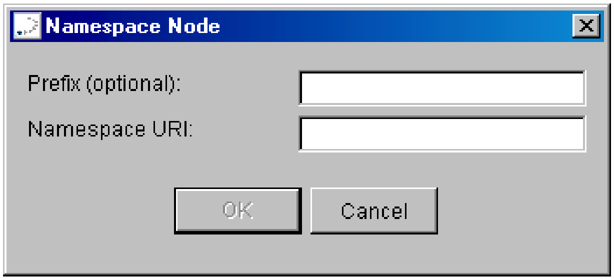
4 Click **OK**. The ATTRIBUTE node appears in the project tree.



```
<example4>
ROOT ELEMENT: exns:example4
  ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
  Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
    ELEMENT: empinfo
      ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
      ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
      Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
        ELEMENT: project
          ATTRIBUTE: name (SELECT EXPR:p.Name)
          ATTRIBUTE: task (SELECT EXPR:a.Task)
```

Add Namespaces to the Root Element

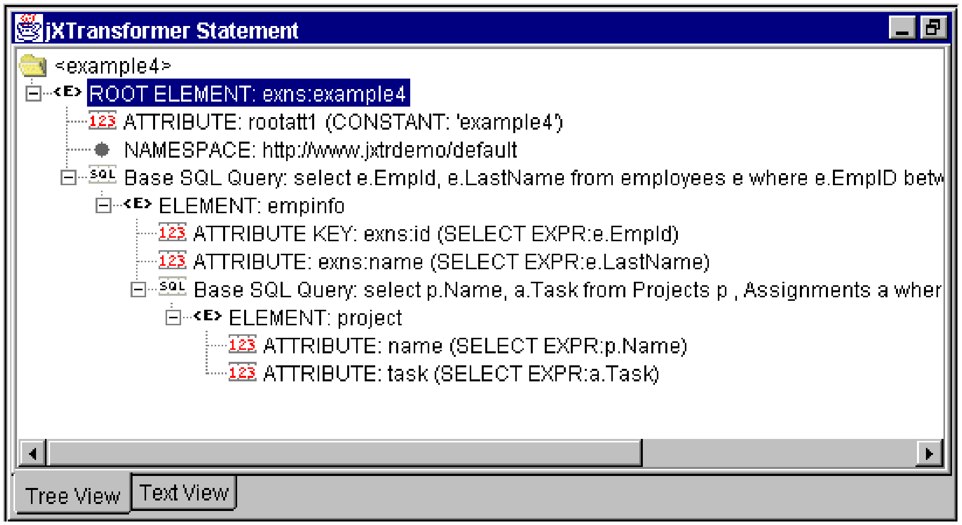
1 Define the default namespace associated with the root element. Select the ROOT ELEMENT node, and select **Insert / Namespace Node**. The Namespace Node dialog box appears.



2 Because we are defining the default namespace, there is no prefix. Leave the Prefix field blank.

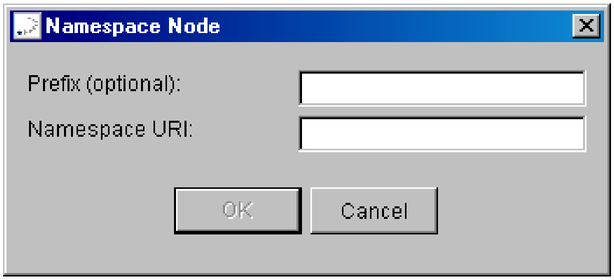
3 In the Namespace URI, type `http://www.jxtrdemo/default`.

4 Click **OK**. The NAMESPACE node appears in the project tree.



```
<example4>
ROOT ELEMENT: exns:example4
    ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
    NAMESPACE: http://www.jxtrdemo/default
    Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
        ELEMENT: empinfo
            ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
            ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
            Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
                ELEMENT: project
                    ATTRIBUTE: name (SELECT EXPR:p.Name)
                    ATTRIBUTE: task (SELECT EXPR:a.Task)
```

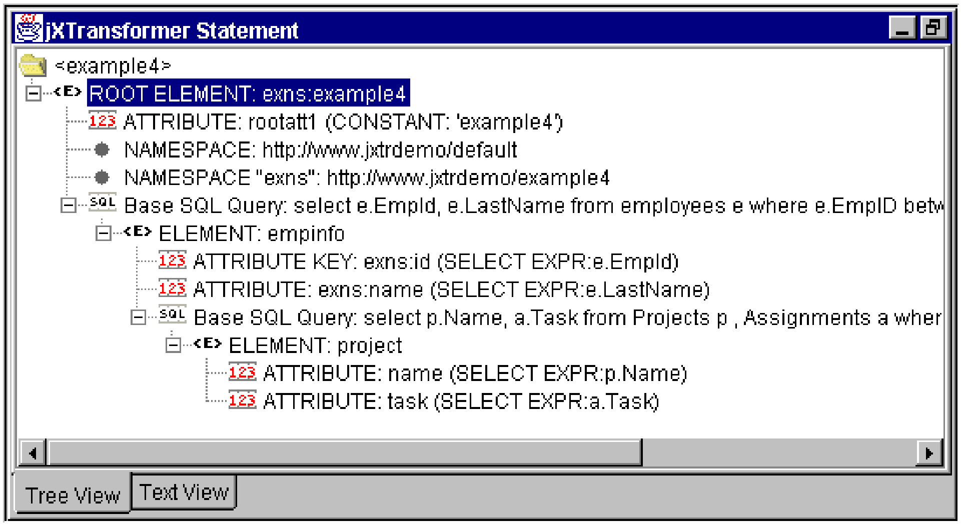
5 Next, define a namespace associated with the root element with a prefix ID of exns. Select the ROOT ELEMENT node, and select **Insert / Namespace Node**. The Namespace Node dialog box appears.



6 In the Prefix field, type `exns`.

7 In the Namespace URI field, type `http://www.jxtrdemo/example4`.

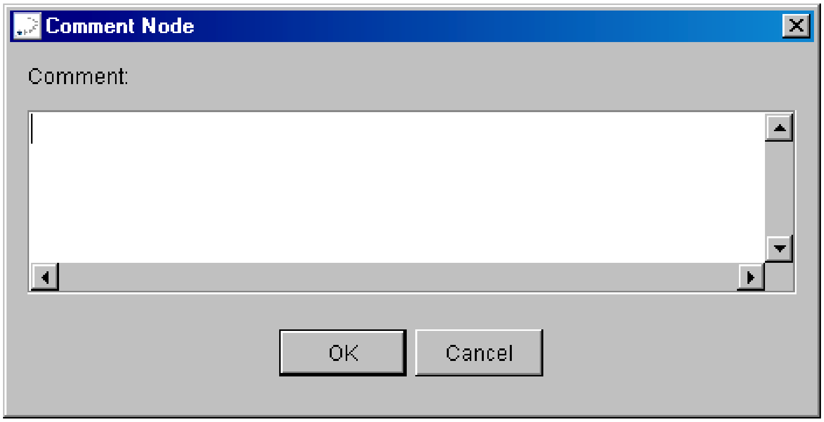
8 Click **OK**. The NAMESPACE node appears in the project tree.



```
jXTransformer Statement
<example4>
ROOT ELEMENT: exns:example4
    ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
    NAMESPACE: http://www.jxtrdemo/default
    NAMESPACE "exns": http://www.jxtrdemo/example4
    Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
        ELEMENT: empinfo
            ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
            ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
            Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
                ELEMENT: project
                    ATTRIBUTE: name (SELECT EXPR:p.Name)
                    ATTRIBUTE: task (SELECT EXPR:a.Task)
```

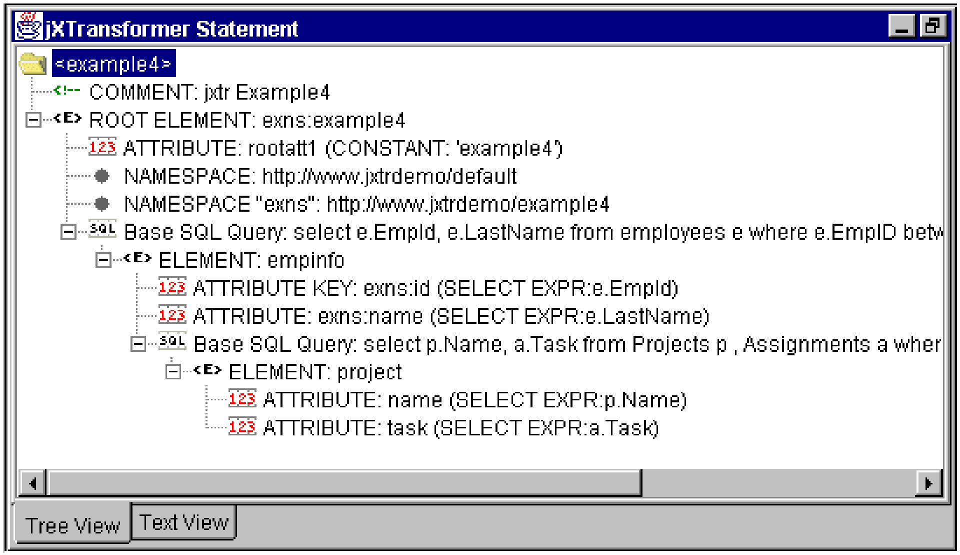
Adding a Comment

1 Next, add a comment to the document header. Select the project node, and select **Insert / Comment**. The Comment Node dialog box appears.



2 In the Comment field, type `jxtr Example4`.

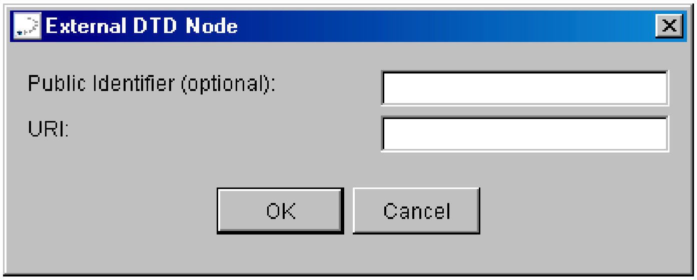
3 Click **OK**. The COMMENT node appears in the project tree.



```
<example4>
  <!-- COMMENT: jxtr Example4
  <E> ROOT ELEMENT: exns:example4
     123 ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
     ● NAMESPACE: http://www.jxtrdemo/default
     ● NAMESPACE "exns": http://www.jxtrdemo/example4
     SQL Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
        <E> ELEMENT: empinfo
           123 ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
           123 ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
           SQL Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
              <E> ELEMENT: project
                 123 ATTRIBUTE: name (SELECT EXPR:p.Name)
                 123 ATTRIBUTE: task (SELECT EXPR:a.Task)
```

Adding a Reference to an External DTD

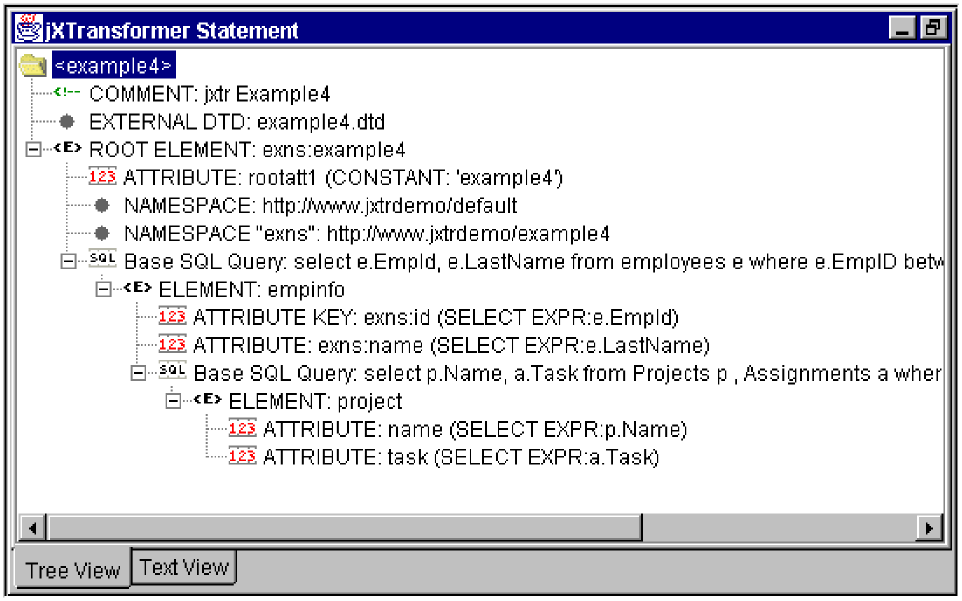
1. Add a reference to an external DTD named example4.dtd. Select the project node, and select **Insert / External DTD**. The External DTD Node dialog box appears.



2. Because you are specifying a private external DTD, leave the Public Identifier field blank.

3. In the URI field, type `example4.dtd`.

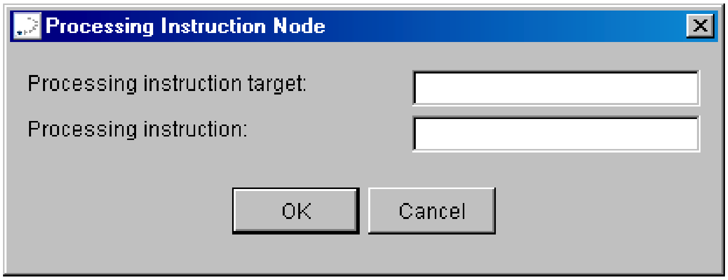
4. Click **OK**. The EXTERNAL DTD node appears in the project tree.



```
<example4>
  COMMENT: jxtr Example4
  EXTERNAL DTD: example4.dtd
  ROOT ELEMENT: exns:example4
    ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
    NAMESPACE: http://www.jxtrdemo/default
    NAMESPACE "exns": http://www.jxtrdemo/example4
    Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
      ELEMENT: empinfo
        ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
        ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
        Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
          ELEMENT: project
            ATTRIBUTE: name (SELECT EXPR:p.Name)
            ATTRIBUTE: task (SELECT EXPR:a.Task)
```

Adding a Processing Instruction

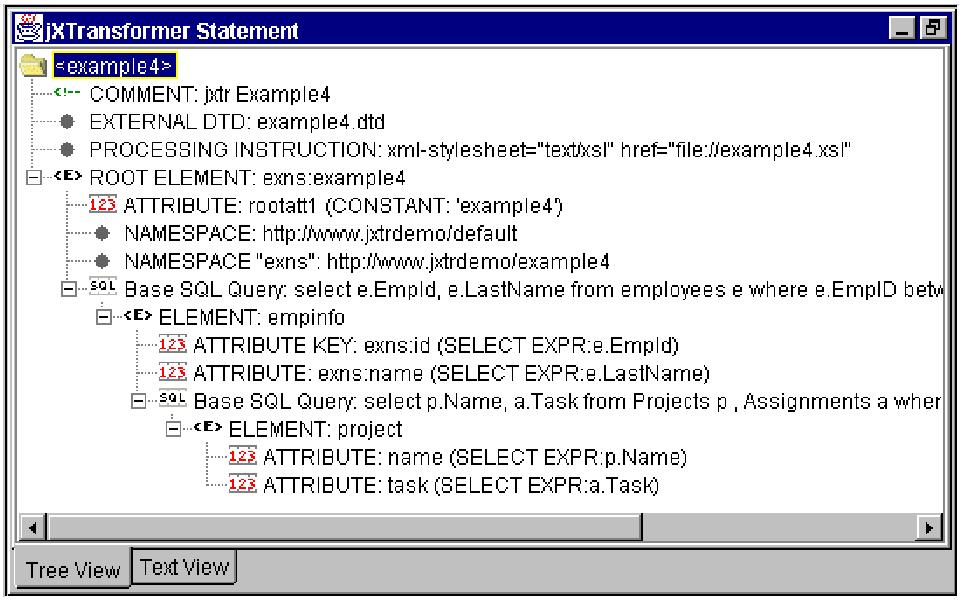
1 Add a processing instruction that specifies an XSL stylesheet named example4.xsl. Select the project node, and select **Insert / Processing Instruction**. The Processing Instruction Node dialog box appears.



2 In the Processing Instruction Target field, type `xml-stylesheet`.

3 In the Processing Instruction field, type `"text/xsl" href= "file://example4.xsl"`.

4 Click **OK**. The PROCESSING INSTRUCTION node appears in the project tree.



```
<example4>
  <!-- COMMENT: jxtr Example4
  ● EXTERNAL DTD: example4.dtd
  ● PROCESSING INSTRUCTION: xml-stylesheet="text/xsl" href="file://example4.xsl"
  <E> ROOT ELEMENT: exns:example4
      123 ATTRIBUTE: rootatt1 (CONSTANT: 'example4')
      ● NAMESPACE: http://www.jxtrdemo/default
      ● NAMESPACE "exns": http://www.jxtrdemo/example4
      SQL Base SQL Query: select e.EmpId, e.LastName from employees e where e.EmpID betw
          <E> ELEMENT: empinfo
              123 ATTRIBUTE KEY: exns:id (SELECT EXPR:e.EmpId)
              123 ATTRIBUTE: exns:name (SELECT EXPR:e.LastName)
              SQL Base SQL Query: select p.Name, a.Task from Projects p , Assignments a wher
                  <E> ELEMENT: project
                      123 ATTRIBUTE: name (SELECT EXPR:p.Name)
                      123 ATTRIBUTE: task (SELECT EXPR:a.Task)
```

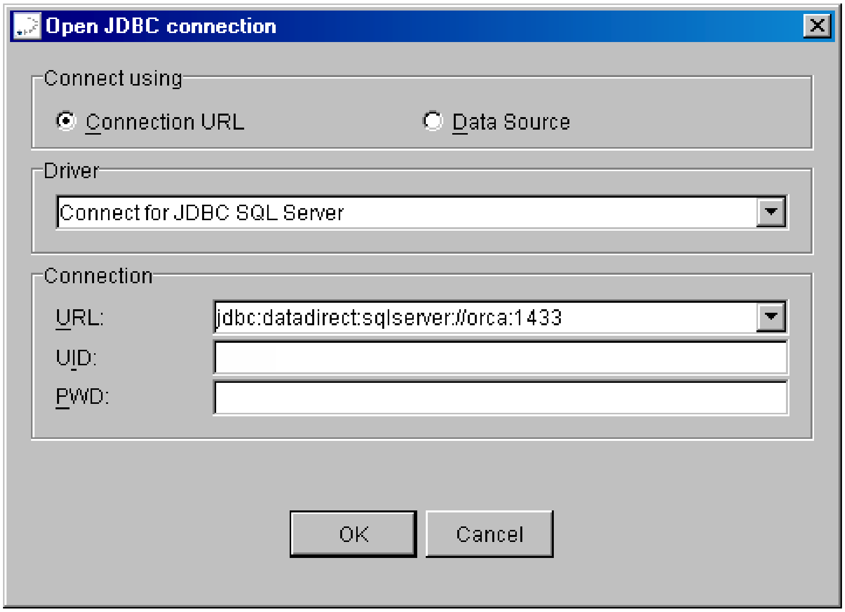
Executing the jXTransformer Query

You have now specified all the document-level constructs in our jXTransformer query example within the query. If you want to see what the XML results would look like, you can execute the query using the Builder.

NOTE: To execute the query and return valid results, you must have created the demo database tables as explained in the README in the examples directory in your Connect *for* SQL/XML installation directory.

To execute the jXTransformer query:

1 Select **Project / Execute Statement**. If you are not connected to the database, you must make a JDBC connection. The Open JDBC Connection dialog box appears.



Open JDBC connection

Connect using

● Connection URL ○ Data Source

Driver: Connect for JDBC SQL Server

Connection

URL: jdbc:datadirect:sqlserver://orca:1433

UID:

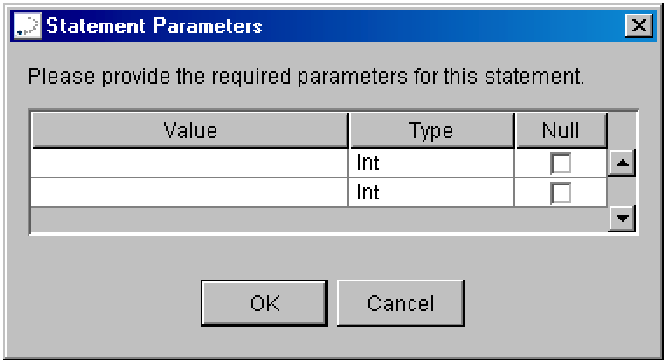
PWD:

The Connection URL option is selected by default. If you want to connect to the database using:

- JDBC connection URL, see "Connecting Using JDBC Connection URLs" on page 118
- JDBC data source, see "Connecting Using JDBC Data Sources" on page 119

Then, continue with the next step.

2 Because the jXTransformer query contains parameter markers, the Query Parameters dialog box appears, prompting you for the parameter values.



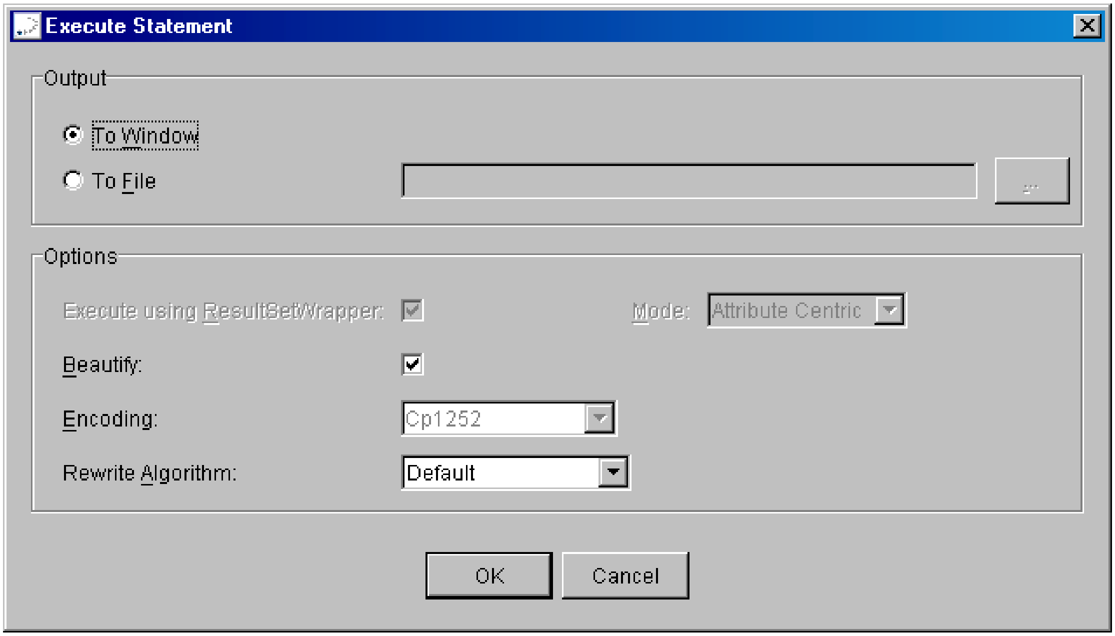
Statement Parameters

Please provide the required parameters for this statement.

Value	Type	Null
	Int	□
	Int	□

In the first Value field, type `1` and in the second Value field, type `5`. Then, click **OK**.

The Execute Statement window appears.



Execute Statement

Output

● To Window

○ To File

Options

Execute using ResultSetWrapper: ☑

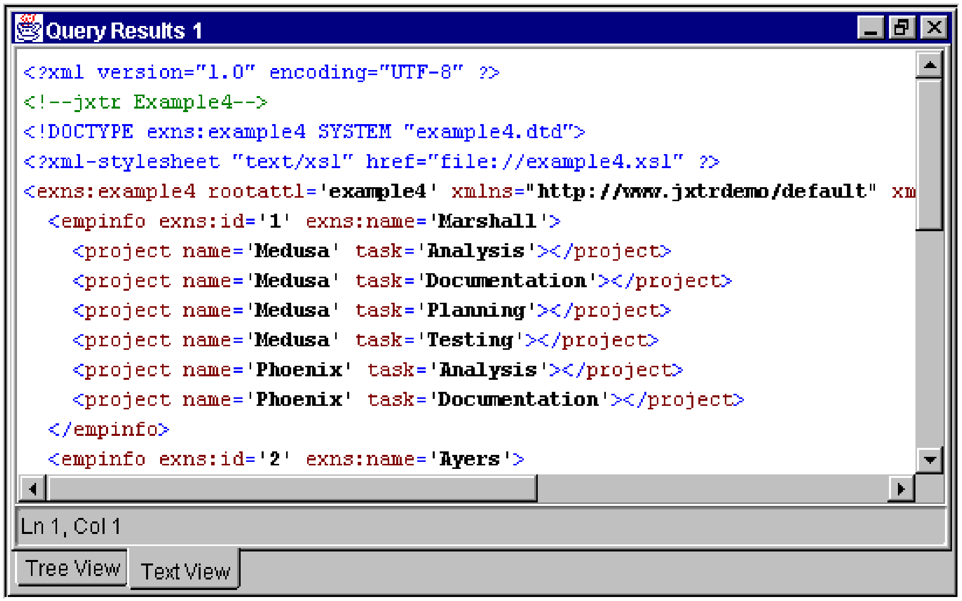
Mode: Attribute Centric

Beautify: ☑

Encoding: Cp1252

Rewrite Algorithm: Default

3 Click **OK** to execute the query. The query results appear in a separate window.



```
<?xml version="1.0" encoding="UTF-8" ?>
<!--jxtr Example4-->
<!DOCTYPE exns:example4 SYSTEM "example4.dtd">
<?xml-stylesheet "text/xsl" href="file://example4.xsl" ?>
<exns:example4 rootattl='example4' xmlns="http://www.jxtrdemo/default" xm
  <empinfo exns:id='1' exns:name='Marshall'>
    <project name='Medusa' task='Analysis'></project>
    <project name='Medusa' task='Documentation'></project>
    <project name='Medusa' task='Planning'></project>
    <project name='Medusa' task='Testing'></project>
    <project name='Phoenix' task='Analysis'></project>
    <project name='Phoenix' task='Documentation'></project>
  </empinfo>
  <empinfo exns:id='2' exns:name='Ayers'>
```

Continue with the next section to learn how to code the jXTransformer query in a Java application.

Using a jXTransformer Query in a Java Application

You can cut and paste the jXTransformer query from the Text view of the Builder into a Java application. For more information about how to use a query in a Java application, refer to the following examples:

- "Example A: Document-Level Constructs in the jXTransformer Query" on page 286 demonstrates how to embed a jXTransformer query that contains document-level constructs in a Java application.
- "Example B: Document-Level Constructs in the jXTransformer API" on page 291 demonstrates how to code the same document-level constructs using the jXTransformer API instead of coding them in the query. The advantage of setting document-level constructs through the jXTransformer API is that you can make changes to them without editing the query.

Example A: Document-Level Constructs in the jXTransformer Query

The code example in this section can be found in the JXTRExample4.java file in the examples/src/examples/jxtrapi directory in your Connect *for* SQL/XML installation directory.

The following code example performs the following tasks:

- Imports the necessary Java classes.
- Sends a description of the example to System.out.
- Loads the necessary Java properties and creates a JDBC connection.

- Embeds the jXTransformer query that you created in the preceding sections. Notice that the query itself specifies the document-level constructs and that each line of the query begins with:

  ```
  jxtrQ.append ( "
  ```

  and ends with:

  ```
  ");
  ```

  Also, notice that quotes in a Java string constant must be preceded by the Java escape character \, for example:

  ```
   jxtrQ.append ( "  xml_pi('xml-stylesheet', ");
   jxtrQ.append ( "  'type=\"text/xsl\"
      href=\"file://example4.xsl\"'), ");
  ```

- Constructs a new jXTransformer query object named JXTRQuery based on the specified JDBC connection, jXTransformer query, and rewrite algorithm.

- Sets the values for the parameter markers that are represented by question marks in the Where clause of the jXTransformer query by calling the jXTransformer method setInt. The format of the jXTransformer setInt method of the JXTRQuery class is:

  `setInt (`*`paramIx`*`, `*`paramValue`*`)`

  where *`paramIx`* identifies the parameter and *`paramValue`* sets the value of the parameter. So, you can see that in our example:

  ```
  JXTRQuery.setInt ( 1, 1 );
  ```

  sets the value of the first parameter marker to 1, and:

  ```
  JXTRQuery.setInt ( 2, 5 );
  ```

  sets the value of the second parameter marker to 5.

TIP: You also can generate a DTD that describes the resulting XML using the Builder.

- Generates a DTD describing the resulting XML named example4.dtd by calling the jXTransformer generateDTD method of the JXTRQuery class.
- Sends the resulting XML document to System.out.
- Closes the JDBC connection and declares the main method.

```
import java.io.OutputStreamWriter;
import java.sql.*;

import com.ddtek.jxtr.*;
import examples.*;

public class JXTRExample4 extends Example
{

    /**
     * Executes the 'JXTRExample4' example
     */
    public void execute() throws Exception
    {
        // Output example description
        System.out.println( "===============================" );
        System.out.println( "= JXTR Example4 demonstrates =" );
        System.out.println( "===============================" );
        System.out.println( "  (a) Creating XML documents that contain 
document level processing instructions," );
        System.out.println( "      comments, namespaces, an external DTD 
reference and a root element with attributes." );
        System.out.println( "  (b) Use of a bind marker in a jxtr statement." 
);
        System.out.println( "  (c) Create a DTD describing the XML that 
results from executing the jxtr query." );

        // Load properties from the resource
        loadProperties();

        // Create JDBC connection
        connectWithStandardJDBC();
        // Build jxtr query
        StringBuffer jxtrQ = new StringBuffer();
        jxtrQ.append ( "xml_document( ");
        jxtrQ.append ( "  xml_comment('jxtr Example4' ), ");
        jxtrQ.append ( "  xml_external_dtd('example4.dtd' ), ");
        jxtrQ.append ( "  xml_pi('xml-stylesheet', ");
        jxtrQ.append ( "         'type=\"text/xsl\" href=
\"file://example4.xsl\"'), ");
        jxtrQ.append ( "  xml_element('exns:example4', ");
        jxtrQ.append ( "    xml_attribute('rootatt1','example4'), ");
        jxtrQ.append ( "    xml_namespace('http://www.jxtrdemo/default'),
");
        jxtrQ.append ( "
xml_namespace('exns','http://www.jxtrdemo/example4'), ");
        jxtrQ.append ( "    select ");
        jxtrQ.append ( "     xml_element('empinfo', ");
        jxtrQ.append ( "       xml_attribute('exns:id',e.EmpId ), ");
        jxtrQ.append ( "       xml_attribute('exns:name',e.LastName)) ");
        jxtrQ.append ( "    from Employees e where e.EmpId between ? and ?))
");

        // Construct new JXTRQuery object
        JXTRQuery JXTRQuery = new JXTRQuery ( conn, new String ( jxtrQ ) );
        // Set bind marker value
        JXTRQuery.setInt ( 1, 1 );
        JXTRQuery.setInt ( 2, 5 );

        // Create dtd
        java.io.FileWriter dtdFile = new java.io.FileWriter ( "example4.dtd"
);
        JXTRQuery.generateDTD ( dtdFile );
        dtdFile.close();

        // Output result header
        System.out.println( "------------" );
        System.out.println( "Query result" );
        System.out.println( "------------" );
        // Execute
        OutputStreamWriter systemOutWriter = new OutputStreamWriter (
System.out );
        boolean outputDocHeader = true;
        JXTRQuery.executeWriter ( systemOutWriter, outputDocHeader,
systemOutWriter.getEncoding ( ), 2 );

        // Close JDBC connection
        disconnect();

    }

    /**
     * Main method.
     */
    public static void main ( String[] args ) throws Exception
    {
        Example thisDemo = new JXTRExample4();
        thisDemo.execute();
    }


}
```

Example B: Document-Level Constructs in the jXTransformer API

The following code example can be found in the JXTRExample5.java file in the examples/src/examples/jxtrapi directory in your Connect *for* SQL/XML installation directory.

The following code example performs the following tasks:

- Imports the necessary Java classes.
- Sends a description of the example to System.out.
- Loads the necessary Java properties and creates a JDBC connection.
- Embeds a version of the jXTransformer query that does not contain the document-level constructs in the jXTransformer query. Instead, they are specified using the jXTransformer API. Again, notice that each line of the query begins with:

  ```
  jxtrQ.append ( "
  ```

  and ends with

  ```
  ");
  ```

- Constructs a new jXTransformer query object named JXTRQuery based on the specified JDBC connection, jXTransformer query, and rewrite algorithm.
- Specifies the document-level constructs through the jXTransformer API by calling the following methods:
  - addDocumentComment adds the XML comment `<!--jxtr Example5-->` in the document header.
  - setExternalDTD creates a reference to an external DTD named example4.dtd. Notice that this DTD was generated by calling the jXTransformer generateDTD method. See “Example A: Document-Level Constructs in

the jXTransformer Query" on page 286 to see how this method was specified.

- addDocumentPI specifies that a stylesheet named example4.xsl be used to format the resulting XML document.
- setRootTag creates a root element named exns:example5.
- addRootAttribute creates an attribute associated with the root element exns:example5 that has a value of example5.
- addRootNameSpace creates a default namespace named http://www.jxtrdemo/default. Then, the addRootNameSpace method is repeated to define a namespace named http://www.jxtrdemo/example5 with the prefix exns.

■ Sets the values for the parameter markers that are represented by question marks in the Where clause of our child query. For this second version of the query, the following code:

```
JXTRQuery.setInt ( 1, 6 );
```

sets the value of the first parameter marker to 6, and:

```
JXTRQuery.setInt ( 2, 10 );
```

sets the value of the second parameter marker to 10.

■ Sends the resulting XML document to System.out.

■ Closes the JDBC connection and declares the main method.

```
import java.io.OutputStreamWriter;
import java.sql.*;

import com.ddtek.jxtr.*;
import examples.*;

public class JXTRExample5 extends Example
```

```
{

    /**
     * Executes the 'JXTRExample5' example
     */
    public void execute() throws Exception
    {
        // Output example description
        System.out.println( "==============================" );
        System.out.println( "= JXTR Example5 demonstrates =" );
        System.out.println( "==============================" );
        System.out.println( "  (a) Creating XML documents that contain 
document level processing instructions," );
        System.out.println( "      comments, namespaces, an external DTD 
reference and a root element with attributes." );
        System.out.println( "  (b) How to accomplish this through the jxtr 
API." );
        System.out.println( "  (c) Use of a bind marker in a jxtr statement." 
);

        // Load properties from the resource
        loadProperties();

        // Create JDBC connection
        connectWithStandardJDBC();

        // Build jxtr query
        StringBuffer jxtrQ = new StringBuffer();
        jxtrQ.append ( "select ");
        jxtrQ.append ( "  xml_element('empinfo', ");
        jxtrQ.append ( "  xml_attribute('exns:id',e.EmpId ), ");
        jxtrQ.append ( "  xml_attribute('exns:name',e.LastName)) ");
        jxtrQ.append ( "from Employees e where e.EmpId between ? and ? ");

        // Construct new JXTRQuery object
        JXTRQuery JXTRQuery = new JXTRQuery ( conn, new String ( jxtrQ ) );
        // Set document level comment,dtd reference and pi
        JXTRQuery.addDocumentComment ( "jxtr Example5" );
        JXTRQuery.setExternalDTD ( "example5.dtd" );
        JXTRQuery.addDocumentPI ( "xml-stylesheet", "type=\"text/xsl\" href=
\"file://example5.xsl\"");
```

```
        // Set root element name, root attribute and namespaces
        JXTRQuery.setRootTag ( "exns:example5" );
        JXTRQuery.addRootAttribute ( "rootatt1", "example5" );
        JXTRQuery.addRootNameSpace ( "http://www.jxtrdemo/default" );
        JXTRQuery.addRootNameSpace ( "exns", "http://www.jxtrdemo/example5"
);
        // Set bind marker value
        JXTRQuery.setInt ( 1, 6 );
        JXTRQuery.setInt ( 2, 10 );

        // Output result header
        System.out.println( "------------" );
        System.out.println( "Query result" );
        System.out.println( "------------" );

        // Execute
        OutputStreamWriter systemOutWriter = new OutputStreamWriter (
System.out );
        boolean outputDocHeader = true;
        JXTRQuery.executeWriter ( systemOutWriter, outputDocHeader,
systemOutWriter.getEncoding ( ), 2 );

        // Close JDBC connection
        disconnect();

    }

    /**
     * Main method.
     */
    public static void main ( String[] args ) throws Exception
    {
        Example thisDemo = new JXTRExample5();
        thisDemo.execute();
    }


}
```

A JDBC Data Types

Table A-1 lists the JDBC data types supported by Connect *for* SQL/XML and the XML representations to which they are converted by the Connect *for* SQL/XML JDBC driver (when SQL/XML queries are used) or the DataDirect JDBC driver (when jXTransformer queries are used).

When jXTransformer write statements are used, the inverse relationship of JDBC data types to XML representations shown in Table A-1 is used. For example, for jXTransformer queries, a JDBC data type of numeric is converted to a string representation of number; for jXTransformer write statements, an XML representation of number is converted to the JDBC data type numeric.

Table A-1. Supported JDBC Data Type Conversions to XML Representations

JDBC Data Type	XML Representation
Char	As-is
Varchar	As-is
Longvarchar	As-is
Numeric	String representation of number
Decimal	String representation of number
Bit	0 or 1
Tinyint	String representation of number
Smallint	String representation of number
Integer	String representation of number
Bigint	String representation of number
Real	String representation of number
Float	String representation of number

Table A-1. Supported JDBC Data Type Conversions to XML Representations (Continued)

JDBC Data Type	XML Representation
Double	String representation of number
Binary	String containing base64 or hexadecimal encoded binary value
Varbinary	String containing base64 or hexadecimal encoded binary value
Longvarbinary	String containing base64 or hexadecimal encoded binary value
Date	YYY-MM-DD
Time	HH:MI:SS
Timestamp	YYYY-MM-DD HH:MI:SS.*Fractional_part* (ODBC) or YYYY-MM-DDTHH: MI:SS.*Fractional_part* (ISO8601)
Clob	As-is
Blob	String containing base64 or hexadecimal encoded binary value

B jXTransformer Query and Statement Processing

This appendix provides additional information that you need to know about how Connect *for* SQL/XML processes jXTransformer query and write statements.

Handling of JDBC PreparedStatement Objects

By default, Connect *for* SQL/XML automatically closes all JDBC preparedStatement objects that have been created to execute a jXTransformer query or write statement after the jXTransformer statement is executed.

NOTE: This does not apply to JXTRQuery or JXTRResultSetWrapper objects for which the JDBC statement type is set to JDBC_STAT using the setPreferredJDBCStatType method in the JXTRQueryBase class.

After invoking JXTRBase.setPreparedStatementCloseBehavior(false), the JDBC preparedStatement objects remain open after execution and are reused when the jXTransformer statement is re-executed. In that case, an explicit call must be made to close all JDBC preparedStatement objects and release their associated resources.

Using Namespaces in XPath Expressions for jXTransformer Write Statements

When using namespaces in an XPath expression, you must map the namespace prefixes to URIs so that matches in the input XML document (which may use other prefixes for the same URIs) can be made correctly. You can map namespace prefixes to URIs using either of the following methods:

- Specifying the jXTransformer xml_namespace constructor (see Chapter 6, "Syntax of jXTransformer Write Statements" on page 173 for more information)
- Specifying the setXMLNamespaces method in the JXTRSingleUpdateStatement class

Using NULL Replacement Values for jXTransformer Write Statements

When an xml_xpath expression does not return text content, by default, the jXTransformer API interprets the value as a NULL value. If you want to change this behavior, use the setNullReplacementValue method in the JXTRBase class. When this method is used, the specified value is used instead of a NULL value.

Glossary

Connect *for* SQL/XML hint
A processing instruction for SQL/XML queries that is handled through a hint to the Connect *for* SQL/XML JDBC driver.

Connect *for* SQL/XML JDBC Driver
The Connect *for* SQL/XML driver that executes SQL/XML queries. This driver uses the JDBC API.

DataDirect Query Builder *for* SQL/XML
A GUI tool that helps you create and modify Connect *for* SQL/XML queries without having to know details about the SQL/XML or jXTransformer syntax. You can also use the Builder to test your Connect *for* SQL/XML queries and jXTransformer write statements before you use them in your application.

DataDirect Query Builder *for* SQL/XML project
A file with either the extension .cfb (for SQL/XML queries) or .jxb (for jXTransformer queries or write statements) that can be opened and saved by the DataDirect Query Builder *for* SQL/XML. One or multiple SQL/XML or jXTransformer queries comprises a Builder project.

DTD (Document Type Definition)
The statement of rules for an XML document that specify which XML elements and attributes are allowed in the document.

DOM (Document Object Model)
A specification for how objects in XML are represented. The DOM defines what attributes are associated with each XML object, and how the objects and attributes can be manipulated.

JAXP
A Java API that supports processing of XML documents using DOM, SAX, and XSLT.

JDOM
A Java API for manipulating XML documents.

jXTransformer query
A query specified in the jXTransformer query syntax. jXTransformer queries return XML results in the form of an XML document or directly to your Java application. jXTransformer queries are executed through the proprietary jXTransformer API.

jXTransformer write statements
Insert, Update, and Delete statements specified in the jXTransformer write statement syntax. jXTransformer write statements allow you to update data from an XML document into a relational database and ar executed through the proprietary jXTransformer API.

mixed content Content for an XML element that is mixed, that is, contains character data or character data with child elements.

namespace A unique identifier that is used to group a set of XML names (elements or attributes).

processing instruction A method of sending instructions to computer applications.

root element In XML, the element that is the single top-level tag. In a jXTransformer query, if you omit the root element, an XML document fragment is created instead of a complete XML document.

SAX The *Simple API for XML* is a standard interface for event-based XML parsing.

schema A pattern that defines the elements, their attributes, and the relationships between different elements.

SQL/XML JDBC driver See Connect *for* SQL/XML JDBC driver.

SQL/XML query SQL/XML queries return JDBC result sets that can contain XML values. SQL/XML queries are executed through the Connect *for* SQL/XML JDBC driver (the SQL/XML JDBC driver), which uses the JDBC API.

URI (Uniform Resource Identifier) A character string that identifies the type and location of an Internet resource.

XML attribute A property associated with an XML element that is a named characteristic of that element.

XML element A section of an XML document that is defined by start- and end-tags.

XPath A language that describes a way to locate and process items in XML documents by using an addressing syntax based on a path through the document's logical structure or hierarchy.

Index