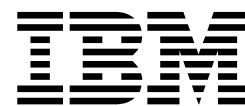


DB2 Universal Database for OS/390 and z/OS



XML Extender Administration and Programming

Version 7

DB2 Universal Database for OS/390 and z/OS



XML Extender Administration and Programming

Version 7

Note

Before using this information and the product it supports, please read the general information under "Appendix E. Notices" on page 265.

First Edition (January 2001)

This edition applies to Version 7 of DB2 Universal Database Server for OS/390 and z/OS, 5675-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© **Copyright International Business Machines Corporation 2000, 2001. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	ix
Who should use this book	ix
How to get a current version of this book	ix
How to use this book	x
Highlighting conventions	x
How to read syntax diagrams	xi
Related information	xiii
How to send your comments	xv

Part 1. Introduction 1

Chapter 1. Introduction to the XML Extender	3
XML documents	3
XML applications	4
Why XML and DB2?	4
How XML and DB2 work together	5
Administration tools.	5
Storage and access methods	5
DTD repository	6
Document Access Definitions (DADs)	6
Location path	6
XML column: structured document storage and retrieval	7
XML collection: integrated data management	8
Chapter 2. Getting started with XML Extender	11
Scenario for the lessons	12
Choosing a method to run the Getting Started lessons	12
Lesson: Store an XML document in an XML column	12
The scenario.	12
Planning	13
Setting up the lesson environment.	15
Enabling the XML column and storing the document	16
Lesson: Composing an XML document	22
The scenario.	22
Planning	23
Setting up the lesson environment.	26
Creating the XML collection: preparing the DAD file	26
Composing the XML document	31
Cleaning up the tutorial environment	32

Part 2. Administration 35

Chapter 3. Preparing to use the XML Extender: administration	37
Set-up requirements	37
Software requirements	37
Installation requirements	37
XML operating environment on OS/390 and z/OS	38
Initializing DB2 XML Extender	39
Workload management considerations	41
Table space considerations when enabling a database server.	42
Security considerations	43
Backup and recovery considerations	45

Administration tools	45
Administration planning	46
Choosing an access and storage method	46
Planning for XML columns.	48
Planning for XML collections	52
Location path	61
Chapter 4. Using the administration tools	65
Starting the administration wizard	65
Setting up the administration wizard	65
Invoking the administration wizard	66
Using the USS odb2 command line	67
Chapter 5. Managing the database server	69
Enabling a database server for XML	69
Using the administration wizard	69
Using the command line	70
Storing a DTD in the DTD repository table	70
Using the administration wizard	71
From the command line	71
Disabling a server for XML	72
Before you begin	72
Using the administration wizard	72
Using the command line	72
Chapter 6. Working with XML columns	73
Creating or editing the DAD file	73
Before you begin	73
Using the administration wizard	73
Using the command line	75
Creating or altering an XML table	76
Using the administration wizard	77
Using the command line	77
Enabling XML columns	77
Before you begin	78
Using the administration wizard	78
Using the command line	79
Indexing side tables	80
Before you begin	81
Creating the indexes	81
Disabling XML columns	81
Before you begin	81
Using the administration wizard	81
Using the command line	82
Chapter 7. Working with XML collections	83
Creating or editing the DAD file for the mapping scheme	83
Before you begin	84
Composing XML documents with SQL mapping	84
Composing XML documents with RDB_node mapping	89
Specifying a stylesheet for the XML document	95
Decomposing XML documents with RDB_node mapping	95
Enabling XML collections.	101
Using the administration wizard	102
Using the command line	102
Disabling XML collections	103

Using the administration wizard	103
Using the command line	103

Part 3. Programming 105

Chapter 8. Managing XML column data	107
User-defined types and user-defined function names	107
Storing data	108
Retrieving data	109
Retrieving an entire document	110
Retrieving element contents and attribute values	111
Updating XML data	113
Searching XML documents	115
Searching the XML document by structure	116
Using the Text Extender for structural text search	117
Deleting XML documents	119
Limitations when invoking functions from Java database (JDBC)	119
Chapter 9. Managing XML collection data	121
Composing XML documents from DB2 data	121
Before you begin.	121
Composing the XML document	121
Dynamically overriding values in the DAD file	126
Decomposing XML documents into DB2 data	130
Enabling an XML collection for decomposition	130
Decomposition table size limits	130
Before you begin.	131
Decomposing the XML document	131
Accessing an XML collection	134
Updating data in an XML collection	134
Deleting an XML document from an XML collection	135
Retrieving XML documents from an XML collection	136
Searching an XML collection	136

Part 4. Reference 139

Chapter 10. XML Extender administration command: DXXADM	141
High-level syntax.	142
Administration subcommands	142
enable_server.	143
disable_server.	145
enable_column	146
disable_column	147
enable_collection	148
disable_collection	149
Chapter 11. XML Extender user-defined types	151
Chapter 12. XML Extender user-defined functions	153
Storage functions	154
XMLVarcharFromFile().	155
XMLCLOBFromFile().	156
XMLFileFromVarchar().	157
XMLFileFromCLOB().	158
Retrieval functions	158

Content(): retrieve from XMLFILE to a CLOB	160
Content(): retrieve from XMLVARCHAR to an external server file	161
Content(): retrieval from XMLCLOB to an external server file	162
Extracting functions.	163
extractInteger() and extractIntegers()	164
extractSmallint() and extractSmallints().	165
extractDouble() and extractDoubles()	167
extractReal() and extractReals()	169
extractChar()and extractChars()	171
extractVarchar() and extractVarchars()	173
extractCLOB() and extractCLOBs()	175
extractDate() and extractDates()	177
extractTime() and extractTimes()	179
extractTimestamp() and extractTimestamps()	181
Update function	183
Purpose	183
Syntax	183
Parameters	183
Return type.	183
Example	184
Usage.	184
Generate unique function	188
Purpose	188
Syntax	188
Return value	188
Example	188
Chapter 13. XML Extender stored procedures	189
Specifying include files	189
Calling XML Extenders stored procedures	189
Increasing the CLOB limit	190
Before you begin.	191
Administration stored procedures.	191
dxxEnableSRV()	192
dxxDisableSRV().	193
dxxEnableColumn().	194
dxxDisableColumn()	195
dxxEnableCollection()	196
dxxDisableCollection().	197
Composition stored procedures	198
dxxGenXML()	199
dxxRetrieveXML()	203
Decomposition stored procedures	206
dxxShredXML()	207
dxxInsertXML()	210
Chapter 14. Administration support tables	213
DTD reference table	213
XML usage table.	213
Chapter 15. Troubleshooting	215
Handling UDF return codes	215
Handling stored procedure return codes	216
SQLSTATE codes	217
Messages	221
Error messages	221

Tracing	232
Starting the trace	233
Stopping the trace	234

Part 5. Appendixes 235

Appendix A. DTD for the DAD file	237
---	------------

Appendix B. Samples	243
XML DTD	243
XML document: getstart.xml	244
Document access definition files	244
DAD file: XML column	244
DAD file: XML collection - SQL mapping	245
DAD file: XML - RDB_node mapping	248

Appendix C. Code page considerations	251
Terminology	251
DB2 and XML Extender code page assumptions	251
Assumptions for importing an XML document	252
Assumptions for exporting an XML document	253
Encoding declaration considerations	254
Legal encoding declarations	254
Consistent encodings and encoding declarations	255
Consistent encodings in USS	257
Declaring an encoding.	258
Conversion scenarios	258
Preventing inconsistent XML documents	259
Line ending considerations	260
Processing XML documents with the linebrk utility	260

Appendix D. The XML Extender limits	263
--	------------

Appendix E. Notices	265
Trademarks.	267

Glossary	269
---------------------------	------------

Index	273
------------------------	------------

About this book

This section describes the following information:

- “Who should use this book”
- “How to use this book” on page x
- “Highlighting conventions” on page x
- “How to read syntax diagrams” on page xi
- “Related information” on page xiii

Who should use this book

This book is intended for the following people:

- Those who work with XML data in DB2 applications and who are familiar with XML concepts. Readers of this document should have a general understanding of XML and DB2. To learn more about XML, refer to the following Web site:

<http://www.w3.org/XML>

To learn more about DB2, see the following Web site:

<http://www.ibm.com/software/data/db2/library>

- DB2 database administrators who are familiar with DB2 administration concepts, tools, and techniques.
- DB2 application programmers who are familiar with SQL and with one or more programming languages that can be used for DB2 applications.

How to get a current version of this book

You can get the latest version of this book at the XML Extender Web site:

<http://www.ibm.com/software/data/db2/extenders/xmlxt/library.html>

How to use this book

This book is structured as follows:

Part 1. Introduction

This part provides an overview of the XML Extender and how you can use it in your business applications. It contains a getting-started scenario that helps you get up and running.

Part 2. Administration

This part describes how to prepare and maintain a DB2 database for XML data. Read this part if you need to administer a DB2 database that contains XML data.

Part 3. Programming

This part describes how to manage your XML data. Read this part if you need to access and manipulate XML data in a DB2 application program.

Part 4. Reference

This part describes how to use the XML Extender administration commands, user-defined types, user-defined functions, and stored procedures. It also lists the messages and codes that the XML Extender issues. Read this part if you are familiar with the XML Extender concepts and tasks, but you need information about a user-defined type (UDT), user-defined function (UDF), command, message, metadata tables, control tables, or code.

Part 5. Appendixes

The appendixes describe the DTD for the document access definition, samples for the examples and getting started scenario, and other IBM XML products.

Highlighting conventions

This book uses the following conventions:

Bold

Bold text indicates:

- Commands
- Field names
- Menu names
- Push buttons

Italic

Italic text indicates:

- Variable parameters that are to be replaced with a value
- Emphasized words
- First use of a glossary term

UPPERCASE

Uppercase letters indicate:

- Data types
- Column names
- Table names

Example

Example text indicates:

- System messages
- Values you type
- Coding examples

- Directory names
- File names
- Path names

How to read syntax diagrams

Throughout this book, the syntax of commands and SQL statements is described using syntax diagrams.

Read the syntax diagrams as follows:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The \blacktriangleright — symbol indicates the beginning of a statement.

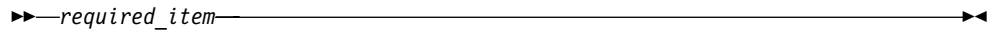
The — \blacktriangleright symbol indicates that the statement syntax is continued on the next line.

The \blacktriangleright — symbol indicates that a statement is continued from the previous line.

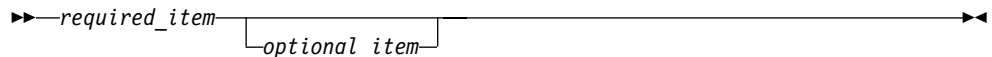
The — \blacktriangleleft symbol indicates the end of a statement.

Diagrams of syntactical units other than complete statements start with the \blacktriangleright — symbol and end with the — \blacktriangleright symbol.

- Required items appear on the horizontal line (the main path).



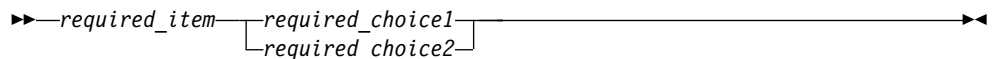
- Optional items appear below the main path.



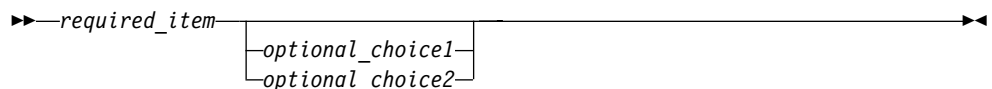
If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.



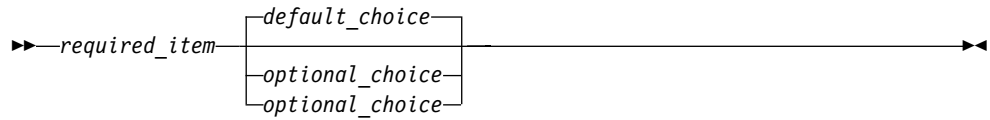
- If you can choose from two or more items, they appear vertically, in a stack. If you *must* choose one of the items, one item of the stack appears on the main path.



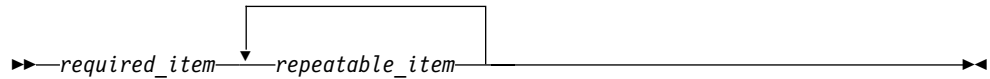
If choosing one of the items is optional, the entire stack appears below the main path.



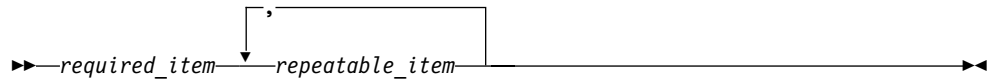
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



If the repeat arrow contains punctuation, you must separate repeated items with the specified punctuation.



A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Keywords appear in uppercase (for example, FROM). In the XML Extender, keywords can be used in any case. Terms that are not keywords appear in lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related information

The following documents might be useful when using the XML Extender and related products:

Document	Order Number	Description
<i>Program Directory for IBM Database 2 Universal Database Server for OS/390 with National Language Versions</i> Program Directory	GI10-8216	This document describes the installation of DB2 UDB Server for OS/390® and related products.
<i>DB2 Universal Database for OS/390 and z/OS Administration Guide, Version 7</i>	SC26-9931	These books describe how to design, implement, and maintain a DB2 database.
<i>DB2 Universal Database for OS/390 and z/OS Application Programming Guide and Reference for Java, Version 7</i>	SC26-9932	This books describes using Java with DB2 for OS/390 and z/OS.
<i>DB2 Universal Database for OS/390 and z/OS Application Programming and SQL Guide, Version 7</i>	SC26-9933	This book describes the application development process and how to code, compile, and execute application programs that use embedded SQL and APIs to access the database.
<i>DB2 Universal Database for OS/390 and z/OS ODBC Guide and Reference, Version 7</i>	SC26-9941	This book provides the information necessary to write applications using DB2 ODBC to access IBM DB2 servers as well as any database that supports DRDA® level 1 or DRDA level 2 protocols. This book should also be used as a supplement when writing portable ODBC applications that can be executed in a native DB2 for OS/390 and z/OS® environment using DB2 ODBC.
<i>DB2 Universal Database for OS/390 and z/OS Release Planning Guide, Version 7</i>	SC26-9943	DB2 Release Guide is intended to help you plan for the current version of the licensed program DB2 for OS/390 and z/OS.
<i>DB2 Universal Database for OS/390 and z/OS SQL Reference, Version 7</i>	SC26-9944	This book serves as a reference for SQL for DB2 Universal Database® Server for OS/390 and z/OS. It is intended for end users, application programmers, system and database administrators, and for persons involved in error detection and diagnosis.

Document	Order Number	Description
<i>DB2 Extender page:</i> http://www.ibm.com/software/data/db2/extendors		This page contains information about the DB2 Extenders as well as technologies that are pertinent to the extenders.
<i>DB2 Universal Database for OS/390 and z/OS Image, Audio, and Video, Version 7</i>	SC26-9947	This book describes how to administer a DB2 database for image, audio, and video data. It also describes how to use application programming interfaces that are provided by the extenders to access and manipulate these types of data.
<i>DB2 Universal Database for OS/390 and z/OS Text Extender Administration and Programming, Version 7</i>	SC26-9948	This book describes how to administer a DB2 database for text data. It also describes how to use application programming interfaces that are provided by the extenders to access and manipulate these types of data.
<i>Integrating XML with DB2 XML Extender and DB2 Text Extender</i>	SG24-6130	This book describes how to use XML and Text Extenders with DB2.
<i>OS/390 UNIX System Services Command Reference</i>	SC28-1892	This book describes USS commands.
<i>OS/390 UNIX System Services Programming: Assembler Callable Services Reference</i>	SC28-1899	This book describes the USS Assembler Callable Services.
<i>OS/390 UNIX System Services Planning</i>	SC28-1890	This book describes planning for USS.
<i>OS/390 UNIX System Services User's Guide</i>	SC28-1891	This book provides tasks for using USS.
<i>IBM Character Data Representation Architecture, Reference and Registry</i>	SC09-2190	This book describes IBM Character Data Representation Architecture, Reference and Registry.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other iSeries documentation, fill out the readers' comment form at the back of this book.

- If you prefer to send comments by mail, use the readers' comment form with the address that is printed on the back. If you are mailing a readers' comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:
 - United States, Canada, and Puerto Rico: 1-800-937-3430
 - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
 - Comments on books:
RCHCLERK@us.ibm.com
 - Comments on the iSeries Information Center:
RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book.
- The publication number of a book.
- The page number or topic of a book to which your comment applies.

Part 1. Introduction

This part provides an overview of the XML Extender and how you can use it in your business applications.

Chapter 1. Introduction to the XML Extender

The IBM® DB2® Extenders™ family provides data and metadata management solutions to handle traditional data types and new, or non-traditional, types of data. The DB2 XML Extender helps you integrate the power of IBM's DB2 Universal Database for OS/390 and z/OS with the flexibility of eXtensible Markup Language (XML).

DB2's XML Extender provides the ability to store and access XML documents, to generate XML documents from existing relational data, and to insert rows into relational tables from XML documents. XML Extender provides new data types, functions, and stored procedures to manage your XML data in DB2 .

The XML Extender is available for the following operating systems:

- Windows NT®
- AIX®
- Sun Solaris
- Linux
- NUMA-Q
- OS/390 and z/OS and zOS
- AS/400®

XML documents

There are many applications in the computer industry, each with its own strengths and weaknesses. Users today have the opportunity to choose whichever application best suits the need requirements of the task at hand. However, because users tend to share data between different applications, they are continually faced with the problem of replicating, transforming, exporting, or saving their data in formats that can be imported into other applications. Many of these transforming processes tend to drop some of the data, or they at least require that users go through the tedious process of ensuring that the data remained consistent. This manual checking consumes both time and money.

Today, one of the ways to address this problem is for application developers to write *Open Database Connectivity (ODBC)* applications, a standard application programming interface (API) for accessing data in both relational and nonrelational database management systems. These applications save the data in a database management system. From there, the data can be manipulated and presented in the form in which it is needed for another application. Database applications must be written to convert the data into a form that an application requires; however, applications change quickly and quickly become obsolete. Applications that convert data to HTML provide presentation solutions, but the data presented cannot be practically used for other purposes. If there were another method that separated the data from its presentation, this method could be used as a practical form of interchange between applications.

XML has emerged to address this problem. XML is an acronym for *eXtensible Markup Language*. It is extensible in that the language itself is a metalanguage that allows you to create your own language depending on the needs of your enterprise. You use XML to capture not only the data for your particular application, but also the data structure. Although XML is not the only data interchange format, XML has

emerged as the accepted standard. By adhering to this standard, applications can share data without first transforming it using proprietary formats.

XML applications

Because XML is now the accepted standard for data interchange, many applications are emerging that will be able to take advantage of it.

Suppose you are using a particular project management application and you want to share some of its data with your calendar application. Your project management application could export tasks in XML, which could then be imported as-is into your calendar application. In today's interconnected world, application providers have strong incentives to make an XML interchange format a basic feature of their application.

Why XML and DB2?

Although XML solves many problems by providing a standard format for data interchange, some challenges remain. When building an enterprise data application, you must answer questions such as:

- How often do I want to replicate the data?
- What kind of information must be shared between applications?
- How can I quickly search for the information I need?
- How can I have a particular action, such as a new entry being added, trigger an automatic data interchange between all my applications?

These kinds of issues can be addressed only by a database management system. By incorporating the XML information and meta-information directly in the database, you can more efficiently obtain the XML results that your other applications need. This is where the XML Extender can assist you. With the XML Extender, you can take advantage of the power of DB2 in many XML applications.

With the content of your structured XML documents in a DB2 database, you can combine structured XML information with traditional relational data. Based on the application, you can choose whether to store entire XML documents in DB2 as in user-defined types provided for XML data (XML data types), or you can map the XML content as base data types in relational tables. For XML data types, the XML Extender adds the power to search rich data types of XML element or attribute values, in addition to the structural text search that the DB2 Text Extender for OS/390 provides.

What XML Extender can do for your applications:

- Store entire XML documents as column data or externally as a file, while extracting desired XML element or attribute values and storing it in *side tables*, indexed subtables for high-speed searching. By storing the documents as column data, you can:
 - Perform fast search on XML elements or attributes that have been extracted and stored in side tables as SQL basic data types and indexed
 - Update the content of an XML element or the value of an XML attribute
 - Extract XML elements or attributes dynamically using SQL queries
 - Validate XML documents during insertion and update
 - Perform structural-text search with the Text Extender

Storing XML documents as column data is known as the XML column method of storage and access.

- Compose or decompose contents of XML documents with one or more relational tables, using the XML collection method of storage and access

How XML and DB2 work together

XML Extender provides the following features to help you manage and exploit XML data with DB2:

- Administration tools to help you manage the integration of XML data in relational tables
- Storage and access methods for XML data within the database
- A data type definition (DTD) repository for you to store DTDs used to validate XML data
- A mapping file called the Document Access Definition (DAD), which is used to map XML documents to relational data

Administration tools

The XML Extender administration tools help you enable your database and table columns for XML, and map XML data to DB2 relational structures. The XML Extender provides the following administration tools for your use, depending on how you want to complete your administration tasks..

XML Extender provides a command line tool, an administration wizard, and programming interfaces for administration tasks.

- The XML Extender administration wizards provide a graphical user interface for administration tasks.
- The DXXADM command can be run from UNIX System Services (USS).
- JCL based on samples provided in the SDXXJCL data set, as listed in Table 6 on page 37
- The XML Extender administration stored procedures allow you to invoke administration commands from a program.

Storage and access methods

XML Extender provides two storage and access methods for integrating XML documents with DB2 data structures: XML column and XML collection. These methods have very different uses, but can be used in the same application.

XML column method

This method helps you store intact XML documents in DB2. The XML column method works well for archiving documents. The documents are inserted into columns that are enabled for XML and can be updated, retrieved, and searched. Element and attribute data can be mapped to DB2 tables (side tables), which can be indexed for fast search.

XML collection method

This method helps you map XML document structures to DB2 tables so that you can either compose XML documents from existing DB2 data, or decompose XML documents, storing the untagged data in DB2 tables. This method is good for data interchange applications, particularly when the contents of XML documents are frequently updated.

DTD repository

The XML Extender allows you to store DTDs, the set of declarations for XML elements and attributes. When a database server is *enabled* for XML, a DTD repository table (DTD_REF) is created. Each row of this table represents a DTD with additional metadata information. Users can access this table to insert their own DTDs. The DTDs are used for validating the structure of XML documents.

Document Access Definitions (DADs)

You specify how structured XML documents are to be processed by the XML Extender using a *document access definition (DAD)* file. The DAD file is an XML-formatted document that maps the XML document structure to a DB2 table. You use a DAD file both when storing XML documents in a column, or when composing or decomposing XML data. The DAD file specifies whether you are storing documents using the XML column method, or defining an XML collection for composition or decomposition.

Location path

A *location path* specifies the location of an element or attribute within an XML document. The XML Extender uses the location path to navigate the structure of the XML document and locate elements and attributes.

For example, a location path of `/Order/Part/Shipment/Shipdate` points to the `shipdate` element, that is a child of the `Shipment`, `Part`, and `Order` elements, as shown in the following example:

```
<Order>
  <Part>
    <Shipment>
      <Shipdate>
        ...
      </Shipdate>
    </Shipment>
  </Part>
</Order>
```

Figure 1 shows an example of a location path and its relationship to the structure of the XML document.

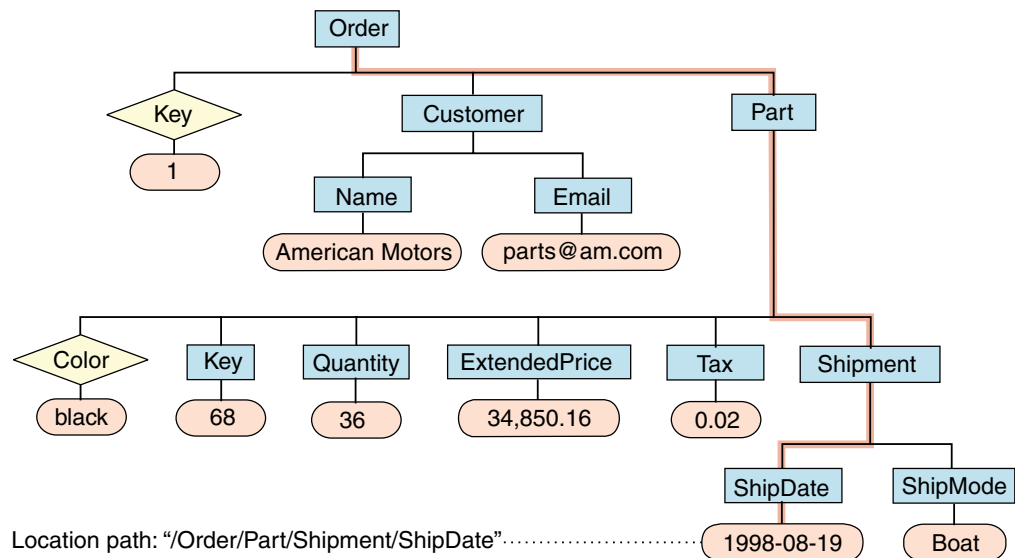


Figure 1. Storing documents as structured XML documents in a DB2 table column

The location path is used in the following situations:

- For XML columns:
 - To identify the elements and attributes to be extracted or updated when using the XML Extender user-defined functions.
 - To map the content of an XML element or attribute to a side table.
- For XML collections: To override values in the DAD file from a stored procedure.

To specify the location path, the XML Extender uses a subset of the *XML Path Language (XPath)*, the language for addressing parts of an XML document.

For more information about XPath, see the following Web page: For XPath, see: <http://www.w3.org/TR/xpath>

See “Location path” on page 61 for syntax and restrictions.

XML column: structured document storage and retrieval

Because XML contains all the necessary information to create a set of documents, there will be times when you want to store and maintain the document structure as it currently is.

For example, if you are a news publishing company that has been serving articles over the Web, you might want to maintain an archive of published articles. In such a scenario, the XML Extender lets you store your complete or partial XML articles in a column of a DB2 table. This type of XML document storage is called an *XML column*, as shown in Figure 2.

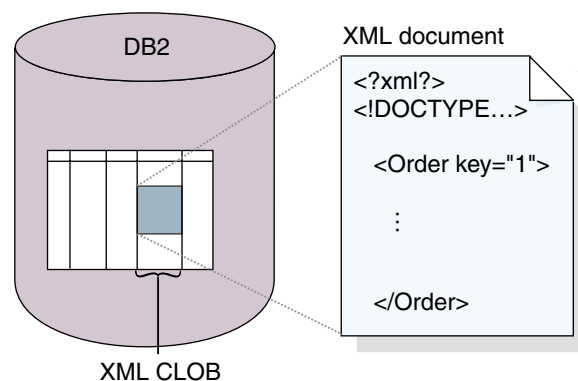


Figure 2. Storing structured XML documents in a DB2 table column

The XML column storage access method allows you to manage your XML documents using DB2. You store XML documents in a column of XML type, providing a way to query the structure and contents of document. You can associate and store a DTD in DB2 for one or more documents. Additionally, you can map element and attribute content to DB2 tables, called *side tables*, that can be indexed. The column that is used to store the document is called an XML column, specifying that the column is used for the XML column access and storage method.

You specify the XML column using the document access definition (DAD) file. The DAD file identifies the XML column and maps XML element and attribute content to be stored in the side tables that are to be indexed.

XML Extender user-defined types

The XML Extender provides the following *user-defined types* for use with XML columns:

- XMLVarchar
- XMLCLOB
- XMLFILE

User-defined types are data types created by a DB2 application or tool.

These data types are used to identify the storage type of XML documents in the application table. You can also store XML documents as files on the file system, specifying a file name.

All the XML Extender's user-defined types have the qualifier **DB2XML**, which is the *schema name* of the DB2 XML Extender user-defined types. For example:

```
db2xml.XMLVarchar
```

The DB2 XML Extender provides powerful *user-defined functions (UDFs)* to store and retrieve XML documents in XML columns, as well as to extract XML element or attribute values. A UDF is a function that is defined to the database management system and can be referenced thereafter in SQL statements. The XML Extender provides the following types of UDFs:

- Storage: Stores intact XML documents in XML-enabled columns at XML data types
- Extract: Extracts XML documents, or the values specified for elements and attributes as base data types
- Update: Updates entire XML documents or specified element and attribute values

XML Extender user-defined functions

The XML user-defined functions (UDFs) allow you to perform powerful searches on general SQL data types. Additionally, you can use the DB2 Text Extender for OS/390 with the XML Extender to perform structural and *full text searches* on text in XML documents. This powerful search capability can be used, for example, to improve the usability of a Web site that publishes large amounts of readable text, such as newspaper articles or *Electronic Data Interchange (EDI)* applications, which have frequently searchable elements or attributes.

All the XML Extender's UDFs have the qualifier **DB2XML**, which is the schema name of the DB2 XML Extender UDFs. The UDFs operate on XML UDTs when working with XML documents in the database.

XML collection: integrated data management

Relational data is either *decomposed* from incoming XML documents or used to *compose* outgoing XML documents. Decomposed data is the untagged content of an XML document stored in one or more database tables. Or, XML documents are composed from existing data in one or more database tables. If your data is to be shared with other applications, you might want to be able to compose and decompose incoming and outgoing XML documents and manage the data as necessary to take advantage of the relational capabilities of DB2. This type of XML document storage is called *XML collection*.

An example of an XML collection is shown in Figure 3 on page 9.

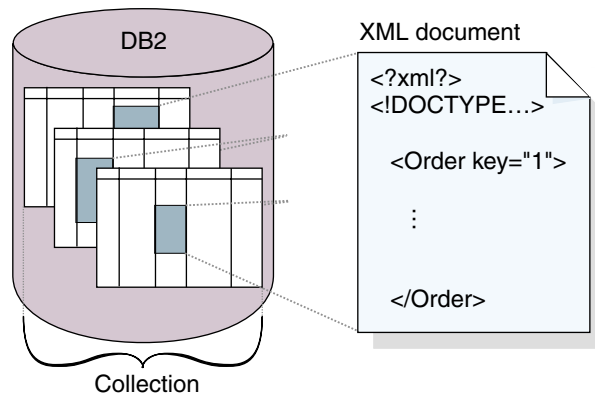


Figure 3. Storing documents as untagged data in DB2 tables

The XML collection is defined in a DAD file, which specifies how elements and attributes are mapped to one or more relational tables. The collection is a set of columns, associated with a DAD file, that contain the data in a particular XML document or set of XML documents. You can define a collection name by enabling it, and then refer to it by name when issuing a stored procedure to compose or decompose XML documents, called an enabled XML collection. The collection is given a name so that it is easily run with stored procedures when composing and decomposing the XML documents.

When you define a collection in the DAD file, you use one of two types of mapping schemes, *SQL mapping* or *RDB_node mapping*, that define the tables, columns, and conditions used to associate XML data with DB2 tables. SQL mapping uses SQL SELECT statements to define the DB2 tables and conditions used for the collection. RDB_node mapping uses an XPath-based relational database node, or RDB_node, which has child elements.

Stored procedures are provided to compose or decompose XML documents. The stored procedures use the qualifier **DB2XML**, which is the *schema name* of the XML Extender.

Chapter 2. Getting started with XML Extender

This chapter shows you how to get started using the XML Extender to access and modify XML data for your applications. By following the provided tutorial lessons, you can set up a database using provided sample data, map SQL data to an XML document, store XML documents in the database, and then search and extract data from the XML documents.

In the administration lessons, you use the *odb2* command for *UNIX System Services (USS)* with XML Extender administration commands. You can accomplish these tasks with the XML Extender administration wizard, which is also described in this book. In XML data management lessons, you will use XML Extender-provided UDFs and stored procedures. Most of the examples in the rest of the book draw on the sample data that is used in this chapter.

Required: To complete the lessons in this chapter, you must have the following prerequisites installed:

- DB2 for OS/390 and z/OS Version 7
- XML Toolkit for OS/390 and z/OS
- USS set up
- *odb2* command line
- Run the the DXXGPREP JCL job, as described in “Initializing the XML Extender environment using DXXGPREP” on page 39.

Additionally, the DB2 database server must have enabled by the DB2 administrator. See “Initializing DB2 XML Extender” on page 39 to learn what this task requires.

The lessons are as follows:

- Store an intact XML document in a DB2 table column
 - Plan the XML user-defined type (UDT) in which to store the document and the XML elements and attributes to be frequently searched.
 - Set up the database and tables
 - Insert the DTD into the DTD repository table
 - Prepare a DAD for an XML column
 - Add a column of XML type to an existing table
 - Enable the new column for XML
 - Create indexes on the side tables
 - Store an XML document in the XML column
 - Search the XML column using XML Extender UDFs
- Create an XML document from existing data
 - Plan the data structure of the XML document
 - Set up the database and tables
 - Prepare a document access definition (DAD) file for an XML collection
 - Compose the XML document from existing data
 - Retrieve the XML document from the database
- Clean up the database

Scenario for the lessons

In these lessons, you work for ACME Auto Direct, a company that distributes cars and trucks to automotive dealerships. You have been given two tasks. First you will set up a system in which orders can be archived in the SALES_DB database for querying by the sales department. The second task is to take information in an existing purchase order database, SALES_DB, and extract key information from it to be stored in XML documents.

Choosing a method to run the Getting Started lessons

Several methods for running the scripts and commands are provided. You can use USS with the odb2 command line, or execute jobs from the TSO environment. See “XML operating environment on OS/390 and z/OS” on page 38 to learn more about using XML Extender in the OS/390 and z/OS operating environment.

See “Initializing the XML Extender environment using DXXGPREP” on page 39 to learn how to set up the samples for the Getting Started lessons.

- Use the odb2 command line to run SQL statements. See “Software requirements” on page 37 to learn how to download and install this tool.

From USS prompt type:

```
odb2
```

A command prompt is displayed, from which you can enter SQL commands.

- Alternatively, start TSO to submit jobs that will issue equivalent steps.

Lesson: Store an XML document in an XML column

The XML Extender provides a method of storing and accessing whole XML documents in the database, called XML column. Using the XML column method, you can store the document using the XML file types, index the column in side tables, and then query or search the XML document. This storage method is particularly useful for archival applications in which documents are not frequently updated.

The scenario

You have been given the task of archiving sales data for the service department. The data is stored in XML documents that use the same DTD. The service department will use these XML documents when working with customer requests and complaints.

The service department has provided a recommended structure for the XML documents and specified which element data they believe will be queried most frequently. They would like the XML documents stored in the SALES_TAB table in the SALES_DB database and want to be able to search them quickly. The SALES_DB table will contain two columns with data about each sale, and a third column to contain the XML document. This column is called ORDER.

You will determine the XML Extender-provided user-defined types (UDTs) in which to store the XML document, as well as which XML elements and attributes will be frequently queried. Next, you will set up the SALES_DB database for XML, create the SALES_TAB table, and enable the ORDER column so that you can store the intact document in DB2. You will also insert a DTD for the XML document for validation and then store the document as an XMLVARCHAR data type. When you

enable the column, you will define side tables to be indexed for the structural search of the document in a document access definition (DAD) file, an XML document that specifies the structure of the side tables. To see samples of the DAD file, the DTD, and the XML document, see “Appendix B. Samples” on page 243.

The SALES_TAB is described in Table 1. The XML column to be enabled for XML, ORDER, is shown in italics.

Table 1. SALES_TAB table

Column name	Data type
INVOICE_NUM	CHAR(6) NOT NULL PRIMARY KEY
SALES_PERSON	VARCHAR(20)
<i>ORDER</i>	XMLVARCHAR

Planning

Before you begin working with the XML Extender to store your documents, you need to understand the structure of the XML document so that you can determine how to search the document. When planning how to search the document, you need to determine:

- The XML user-defined type in which you will store the XML document
- The XML elements and attributes that the service department will frequently search, so that their content can be stored in side tables and indexed to improve performance.

The following sections will describe how to make these decisions.

The XML document structure

The XML document structure for this lesson takes information for a specific order that is structured by the order key as the top level, then customer, part, and shipping information on the next level. The XML document is described in “XML document: getstart.xml” on page 244.

This lesson also provides a sample DTD for you to use in understanding and validating the XML document structure. You can see the DTD file in “XML DTD” on page 243.

Determining the XML data type for the XML column

The XML Extender provides XML user defined types in which you define a column to hold XML documents. These data types are:

- XMLVarchar: for small documents stored in DB2
- XMLCLOB: for large documents stored in DB2
- XMLFILE: for documents stored outside DB2

In this lesson, you will store a small document in DB2 and will, therefore, use the XMLVarchar data type.

Determining elements and attributes to be searched

When you understand the XML document structure and the needs of the application, you can determine which elements and attributes to be searched: such as the elements and attributes that will be searched or extracted most frequently, or those that will be the most expensive to query. The service department has indicated they will be frequently querying the order key, customer name, price, and shipping date of an order, and need quick performance for these searches. This information is contained in elements and attributes of the XML document structure.

Table 2 describes the location paths of each element and attribute.

Table 2. Elements and attributes to be searched

Data	Location path
order key	/Order/@key
customer	/Order/Customer/Name
price	/Order/Part/ExtendedPrice
shipping date	/Order/Part/Shipment/ShipDate

Mapping the XML document to the side tables

You will create a DAD file for the XML column, which is used to store the XML document in DB2. It also maps the XML element and attribute contents to DB2 side tables used for indexing, which improves search performance. In the last section, you saw which elements and attributes are to be searched. In this section, you learn more about mapping these element and attribute values to DB2 tables that can be indexed.

After identifying the elements and attributes to be searched, you determine how they should be organized in the side tables, how many tables and which columns are in what table. Typically, you organize the side tables by putting similar information in the same table. The structure is also determined by whether the location path of any elements can be repeated more than once in the document. For example in our document, the part element can be repeated multiple times, and therefore, the price and date elements can occur multiple times. Elements that can occur multiple times must each be in their own side tables.

Additionally, you also must determine what DB2 base types the element or attribute values should use. Typically, this is easily determined by the format of the data. If the data is text, choose VARCHAR; if the data is an integer, choose INTEGER; or if the data is a date, and you want to do range searches, choose DATE.

In this tutorial, you will map the elements and attributes to the following side tables:

ORDER_SIDE_TAB

Column name	Data type	Location path	Multiple occurring?
ORDER_KEY	INTEGER	/Order/@key	No
CUSTOMER	VARCHAR(16)	/Order/Customer/Name	No

PART_SIDE_TAB

Column name	Data type	Location path	Multiple occurring?
PRICE	DECIMAL(10,2)	/Order/Part/ExtendedPrice	Yes

SHIP_SIDE_TAB

Column name	Data type	Location path	Multiple occurring?
DATE	DATE	/Order/Part/Shipment/ShipDate	Yes

Getting started scripts and samples

For this tutorial, you use a set of scripts and JCL samples to set up your environment and perform the steps in the lessons. These scripts are in the `dxx_install/samples/cmd` directory (where `dxx_install` is the directory in USS, where the sample DTD, DAD, and XML files are copied by the DXXGPREP job, see “Initializing the XML Extender environment using DXXGPREP” on page 39).

Table 3 lists the USS and JCL samples that are provided to complete the getting started tasks, as well as the suggested role in the organization that might have the correct authority to run the samples.

Table 3. List of the XML column lesson samples

Role	Description	USS command files	JCL file
administrator	Creates and populates the database and tables used for the lesson	Getstart_db.cmd	dxxgdb
administrator	Binds and enables the database server	Getstart_prep.cmd	dxxgprep
application developer	Insert the dtd getstart.dtd into the dtd_ref table	Getstart_insertDTD.cmd	dxxgidtd
administrator	Creates SALES_TAB for XML column	Getstart_createTabCol.cmd	dxxgctco
administrator	Adds the ORDER column to SALES_TAB	Getstart_alterTabCol.cmd	dxxgatco
administrator	Enables the ORDER column as an XML column	Getstart_enableCol.cmd	dxxgecol
administrator	Create indexes on side tables	Getstart_createIndex.cmd	dxxgcrin
application developer	Inserts an XML document into the SALES_TAB XML column	Getstart_insertXML.cmd	dxxgixml
application developer	Queries the XML document held in the sales_tab XML column through the side tables	Getstart_queryCol.cmd	dxxgcqol
administrator	Cleans up the environment	Getstart_clean.cmd	dxxaclen

These samples are provided for your use in your applications.

Setting up the lesson environment

In this section, you create the database and tables used for the sample data.

Creating the database

In this section, you create a sample database, create the tables to hold data, and then insert sample data.

To create the database:

1. Ensure that the database server has been enabled by the DB2 administrator. See “Initializing DB2 XML Extender” on page 39 to learn how to enable the server.
2. Change to the *dxx_install/samples/cmd* directory, where *dxx_install* is directory in USS where the sample DTD, DAD, and XML files are located. The sample files contain references to files that use absolute path names. Check the sample files and change these values for your directory paths.
3. Run the GETSTART_DB command, using one of the following methods:
odb2 command line: Enter the following command:

```
getstart_db.cmd
```

See “Choosing a method to run the Getting Started lessons” on page 12 to learn how to start the odb2 command line.

TSO: Submit the dxxgdb JCL job.

Enabling the XML column and storing the document

In this lesson, you will enable a column for XML Extender and store an XML document in the column. For these tasks, you will:

1. Insert the DTD for the XML document into the DTD reference table, DTD_REF.
2. Prepare a DAD file that specifies the XML document location and side tables for structural search.
3. Add a column in the SALES_TAB table with an XML user-defined type of XMLVARCHAR.
4. Enable the column for XML.
5. Index the side tables for structural search.
6. Store the document using a user-defined function, which is provided by the XML Extender.

Storing the DTD in the DTD repository

You can use a DTD to validate XML data in an XML column. The XML Extender creates a table in the XML-enabled database, called DTD_REF. The table is known as the DTD reference and is available for you to store DTDs. When you validate XML documents, you must store the DTD in this repository. The tutorial DTD is *dxx_install/samples/dtd/getstart.dtd*.

To insert the DTD:

Enter the SQL INSERT statement using one of the following methods:

Command line:

- Connect to the database and enter the following SQL INSERT command, all on the same line:

```
DB2 INSERT into DB2XML.DTD_REF values('dxx_install/samples/dtd/getstart.dtd',
DB2XML.XMLClobFromFile('dxx_install/samples/dtd/getstart.dtd'), 0, 'user1',
'user1', 'user1')
```

- Or, run the following command file to insert the DTD:

```
getstart_insertDTD.cmd
```

TSO: Submit the dxxgidtd JCL job.

Preparing the DAD file

The DAD file for the XML column has a simple structure. You specify that the storage mode is XML column, and you define the tables and columns for indexing.

In the following steps, elements in the DAD are referred to as *tags* and the elements of your XML document structure are referred to as *elements*. A sample of a DAD file similar to the one you will create is in `dxx_install/samples/dad/getstart_xcolumn.dad`. It has some minor differences from the file generated in the following steps. If you use it for the lesson, note that the file paths might be different that for your environment, the `<validation>` value is set to NO, rather than YES.

To prepare the DAD file:

1. Open a text editor and name the file `getstart_xcolumn.dad`
Note that all the tags used in the DAD file are case sensitive.
2. Create the DAD header, with the XML and the DOCTYPE declarations.

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
```

The DAD file is an XML document and requires XML declarations.

3. Insert opening and closing `<DAD></DAD>` tags. All other tags are located inside these tags.
4. Insert opening and closing `<DTDID></DTDID>` tags with a DTD ID to specify a DTD if the document will be validated:

```
<dtid>dxx_install/samples/dtd/getstart.dtd</dtid>
```

Verify that this string matches the value used as the first parameter value when inserting the DTD in the DTD reference table in “Storing the DTD in the DTD repository” on page 16. For example, the path you used for the DTDID might be different be different that the above string if you are working on a different machine drive.

5. Specify opening and closing `<validation></validation>` tags and a keyword YES or NO to indicate whether the XML Extender is to validate the XML document structure using the DTD you inserted into the DTD repository table.

```
<validation>YES</validation>
```

The value of `<validation>` must be in uppercase.

6. Insert opening and closing `<Xcolumn></Xcolumn>` tags to define the storage method as XML column.

```
<Xcolumn>
</Xcolumn>
```

7. Insert opening and closing `<table></table>` tags for each side table that is to be generated.

```
<Xcolumn>
<table name="order_side_tab">
</table>
<table name="part_side_tab">
</table>
<table name="ship_side_tab">
</table>
</Xcolumn>
```

8. Insert `<column/>` tags for each column that is to be included in the side tables. Each `<column/>` tag has four attributes:

- **name:** the name of the column

- **type:** the SQL data type of the column
- **path:** the location path of the corresponding element in the XML document, using XPath syntax. See “Location path” on page 61 for location path syntax.
- **multi-occurrence:** indication of whether the location path of the element can occur more than once in the XML document structure

```

<Xcolumn>
<table name="order_side_tab">
  <column name="order_key"
    type="integer"
    path="/Order/@key"
    multi_occurrence="NO"/>
  <column name="customer"
    type="varchar(50)"
    path="/Order/Customer/Name"
    multi_occurrence="NO"/>
</table>
<table name="part_side_tab">
  <column name="price"
    type="decimal(10,2)"
    path="/Order/Part/ExtendedPrice"
    multi_occurrence="YES"/>
</table>
<table name="ship_side_tab">
  <column name="date"
    type="DATE"
    path="/Order/Part/Shipment/ShipDate"
    multi_occurrence="YES"/>
</table>
</Xcolumn>

```

9. Ensure that you have a closing </Xcolumn> after the last </table> tag.
10. Ensure that you have a closing </DAD> after the </Xcolumn> tag.
11. Save the file as getstart_xcolumn.dad.

You can compare the file you have just created with the sample file, *dxx_install/samples/dad/getstart_xcolumn.dad*. This file is a working copy of the DAD file required to enable the XML column and create the side tables. The sample files contain references to files that use absolute path names. Check the sample files and change these values for your directory paths.

Creating the SALES_TAB table

In this section you create the SALES_TAB table. Initially, it has two columns with the sale information for the order.

To create the table: Enter the following CREATE TABLE statement using one of the following methods: **Command line:**

- Enter the following DB2 commands:

```
DB2 CONNECT TO SALES_DB
```

```
DB2 CREATE TABLE SALES_TAB(INVOICE_NUM CHAR(6) NOT NULL PRIMARY KEY,
  SALES_PERSON VARCHAR(20))
```

- Or, run the following command file to create the table:

```
getstart_createTabCol.cmd
```

TSO: Submit the dxxgctco JCL job.

Adding the column of XML type

Next, add a new column to the SALES_TAB table. This column will contain the intact XML document that you generated earlier and must be of XML UDT. The XML

Extender provides multiple data types, described in “Chapter 11. XML Extender user-defined types” on page 151. In this tutorial, you will store the document as XMLVARCHAR.

To add the column of XML type:

Run the SQL ALTER TABLE statement using one of the following methods:

Command line:

- Enter the following SQL statement:

```
DB2 ALTER TABLE SALES_TAB ADD ORDER DB2XML.XMLVARCHAR
```
- Or, run the following command file to alter the table:

```
getstart_alterTabCol.cmd
```

TSO: Submit the dxxgatco JCL job.

Enabling the XML column

After you create the column of XML type, you enable it for the XML Extender. When you enable the column, the XML Extender reads the DAD file and creates the side tables. Before enabling the column, you must:

- Determine whether you want to create a default view of the XML column, which contains the XML document joined with the side-table columns. You can specify the default view when querying the XML document. In this lesson, you will specify a view with the `-v` parameter.
- Determine whether you want to specify a primary key as the *ROOT ID*, the column name of the primary key in the application table and a unique identifier that associates all side tables with the application table. If you do not specify a primary key, the XML Extender adds the `DXXROOT_ID` column to the application table and to the side tables.

The `ROOT_ID` column is used as key to tie the application and side tables together, allowing the XML Extender to automatically update the side tables if the XML document is updated. In this lesson, you will specify the name of the primary key in the command (`INVOICE_NUM`) with the `-r` parameter. The XML Extender will then use the specified column as the `ROOT_ID` and add the column to the side tables.

- Determine whether you want to specify a table space or use the default table space. In this lesson, you will use the default table space.

To enable the column for XML:

Run the `DXXADM ENABLE_COLUMN` command, using one of the following methods: **Command line:**

- Enter the following command:

```
dxxadm enable_column SALES_DB SALES_TAB ORDER GETSTART_XCOLUMN.DAD  
-v SALES_ORDER_VIEW -r INVOICE_NUM
```
- Or, run the following command file to enable the column:

```
dxxadm enable_column SALES_DB SALES_TAB ORDER GETSTART_XCOLUMN.DAD  
-v SALES_ORDER_VIEW -r INVOICE_NUM
```

TSO: Submit the dxxgeco1 JCL job.

The XML Extender creates the side tables with the `INVOICE_NUM` column and creates the default view.

Important: Do not modify the side tables in any way. Updates to the side tables should only be made through updates to the XML document itself. The XML Extender will automatically update the side tables when you update the XML document in the XML column.

Viewing the column and side tables

When you enabled the XML column, you created a view of the XML column and side tables. You can use this view when working with the XML column.

To view the XML column and side-table columns: Enter the following SQL SELECT statement from the command line:

```
DB2 SELECT * FROM SALES_ORDER_VIEW
```

The view shows the columns in the side tables, as specified in the `getstart_xcolumn.dad` file.

Creating indexes on the side tables

Creating indexes on side tables allows you to do fast structural searches of the XML document. In this section, you create indexes on key columns in the side tables that were created when you enabled the XML column, ORDER. The service department has specified which columns their employees are likely to query most often. Table 4 describes these columns, which you will index:

Table 4. Side-table columns to be indexed

Column	Side table
ORDER_KEY	ORDER_SIDE_TAB
CUSTOMER	ORDER_SIDE_TAB
PRICE	PART_SIDE_TAB
DATE	SHIP_SIDE_TAB

To index the side tables:

Run the following CREATE INDEX SQL commands using one of the following methods:

Command line:

- Enter the following commands:

```
DB2 CREATE INDEX KEY_IDX  
      ON ORDER_SIDE_TAB(ORDER_KEY)
```

```
DB2 CREATE INDEX CUSTOMER_IDX  
      ON ORDER_SIDE_TAB(CUSTOMER)
```

```
DB2 CREATE INDEX PRICE_IDX  
      ON PART_SIDE_TAB(PRICE)
```

```
DB2 CREATE INDEX DATE_IDX  
      ON SHIP_SIDE_TAB(DATE)
```

- Or, run the following command file to create the indexes:

```
getstart_createIndex.cmd
```

TSO: Submit the `dxxgcrin` JCL job.

Storing the XML document

Now that you have enabled a column that can contain the XML document and indexed the side tables, you can store the document using the functions that the XML Extender provides. When storing data in an XML column, you either use default casting functions or the XML Extender UDFs. Because you will be storing an object of the base type VARCHAR in a column of the XML UDT XMLVARCHAR, you will use the default casting function. See “Storing data” on page 108 for more information about the storage default casting functions and the XML Extender-provided UDFs.

To store the XML document:

1. Open the XML document `dxx_install/samples/xml/getstart.xml`. Ensure that the file path in the DOCTYPE matches the DTD ID specified in the DAD and when inserting the DTD in the DTD repository. You can verify they match by querying the DB2XML.DTD_REF table and by checking the DTDID element in the DAD file. If you are using a different drive and directory structure than the default, you might need to change the path in the DOCTYPE declaration.
2. Run the SQL INSERT command, using one of the following methods:
Command line:
 - Enter the following SQL INSERT command:

```
DB2 INSERT INTO SALES_TAB (INVOICE_NUM, SALES_PERSON, ORDER) VALUES('123456',
'Sriram Srinivasan', DB2XML.XMLVarcharFromFile('dxx_install/samples/cmd/getstart.xml'))
```
 - Or, run the following command file to store the document:
`getstart_insertXML.cmd`

TSO: Submit the `dxxgixml` JCL job.

To verify that the tables have been updated, run the following SELECT statements for the tables from the command line.

```
DB2 SELECT * FROM SALES_TAB
```

```
DB2 SELECT * FROM PART_SIDE_TAB
```

```
DB2 SELECT * FROM ORDER_SIDE_TAB
```

```
DB2 SELECT * FROM SHIP_SIDE_TAB
```

Searching the XML document

You can search the XML document with a direct query against the side tables. In this step, you will search for all orders that have a price over 2500.00.

To query the side tables:

Run the SQL SELECT statement, using one of the following methods: **Command line:**

- Enter the following SQL SELECT statement:

```
DB2 "SELECT DISTINCT SALES_PERSON FROM SALES_TAB S, PART_SIDE_TAB P
WHERE PRICE > 2500.00 AND
S.INVOICE_NUM=P.INVOICE_NUM"
```
- Or, run the following command file to search the document:
`getstart_queryCol.cmd`

TSO: Submit the `dxxgcqol` JCL job.

The result set should show the names of the salespeople who sold an item that had a price greater than 2500.00.

You have completed the getting started tutorial for storing XML documents in DB2 tables. Many of the examples in the book are based on these lessons.

Lesson: Composing an XML document

This lesson teaches you how to compose an XML document from existing DB2 data.

The scenario

You have been given the task of taking information in an existing purchase order database, SALES_DB, and extracting requested information from it to be stored in XML documents. The service department will then use these XML documents when working with customer requests and complaints. The service department has requested specific data to be included and has provided a recommended structure for the XML documents.

Using existing data, you will compose an XML document, `getstart.xml`, from data in these tables.

You will also plan and create a DAD file that maps columns from the related tables to an XML document structure that provides a purchase order record. Because this document is composed from multiple tables, you will create an XML collection, associating these tables with an XML structure and a DTD. You use this DTD to define the structure of the XML document. You can also use it to validate the composed XML document in your applications.

The existing database data for the XML document is described in the following tables. The column names in *italics* are columns that the service department has requested in the XML document structure.

ORDER_TAB

Column name	Data type
<i>ORDER_KEY</i>	INTEGER
CUSTOMER	VARCHAR(16)
<i>CUSTOMER_NAME</i>	VARCHAR(16)
<i>CUSTOMER_EMAIL</i>	VARCHAR(16)

PART_TAB

Column name	Data type
<i>PART_KEY</i>	INTEGER
<i>COLOR</i>	CHAR(6)
<i>QUANTITY</i>	INTEGER
<i>PRICE</i>	DECIMAL(10,2)
<i>TAX</i>	REAL
<i>ORDER_KEY</i>	INTEGER

SHIP_TAB

Column name	Data type
<i>DATE</i>	DATE

Column name	Data type
<i>MODE</i>	CHAR(6)
COMMENT	VARCHAR(128)
PART_KEY	INTEGER

Planning

Before you begin working with the XML Extender to compose your documents, you need to determine the structure of the XML document and how it corresponds to the structure of your database data. This section will provide an overview of the XML document structure that the service department has requested, of the DTD you will use to define the structure of the XML document, and how this document maps to the columns that contain the data used to populate the documents.

Determining the document structure

The XML document structure takes information for a specific order from multiple tables and creates an XML document for the order. These tables each contain related information about the order and can be joined on their key columns. The service department wants a document that is structured by the order number as the top level, and then customer, part, and shipping information. They want the document structure to be intuitive and flexible, with the elements describing the data, rather than the structure of the document. (For example, the customer's name should be in an element called "customer," rather than a paragraph.) Based on their request, the hierarchical structure of the DTD and the XML document should be like the one described in Figure 4 on page 24.

After you have designed the document structure, you should create a DTD to describe the structure of the XML document. This tutorial provides an XML document and a DTD for you. You can see the DTD file in "Appendix B. Samples" on page 243. Using the rules of the DTD, and the hierarchical structure of the XML document, you can map a hierarchical map of your data, as shown in Figure 4 on page 24.

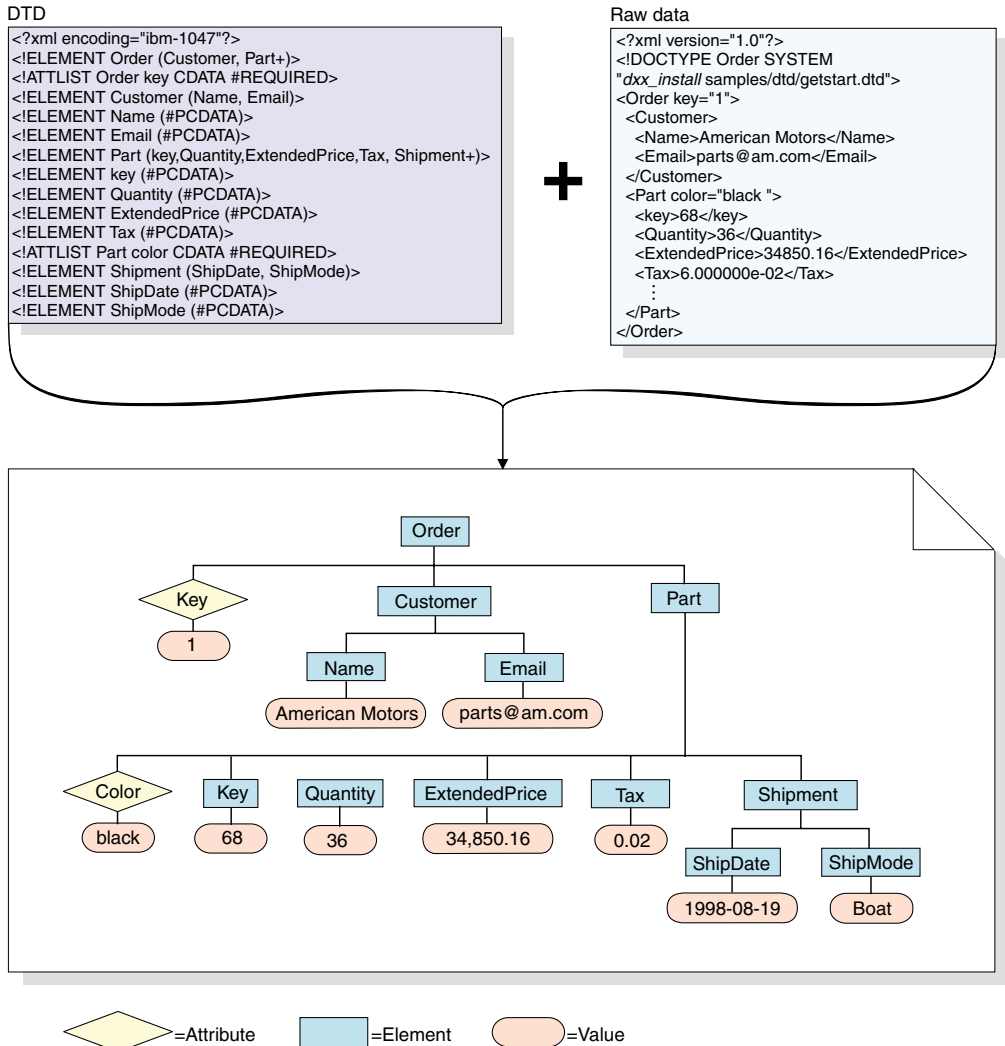


Figure 4. The hierarchical structure of the DTD and XML document

Mapping the XML document and database relationship

After you have designed the structure and created the DTD, you need to show how the structure of the document relates to the DB2 tables that you will use to populate the elements and attributes. You can map the hierarchical structure to specific columns in the relational tables, as in Figure 5 on page 25.

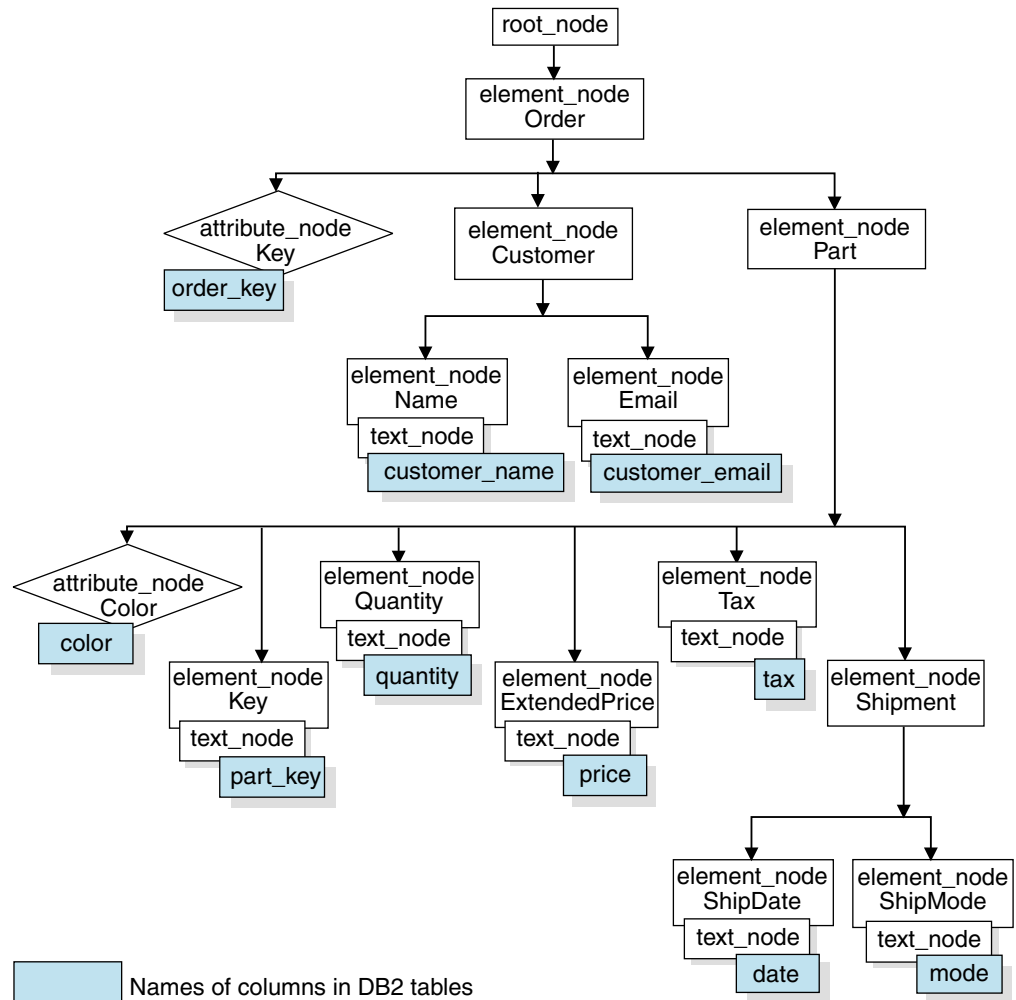


Figure 5. XML document mapped to relational table columns

This figure uses nodes to identify elements, attributes, and text within the XML document structure. These nodes are used in the DAD file and are explained more fully in later steps.

Use this relationship description to create DAD files that define the relationship between the relational data and the XML document structure.

To create the XML collection DAD file, you need to understand how the XML document corresponds to the database structure, as described in Figure 5, so that you can describe from what tables and columns the XML document structure derives data for elements and attributes. You will use this information to create the DAD file for the XML collection.

Getting started scripts and samples

For this tutorial, we provide a set of scripts for you to use to set up your environment. These scripts are in the `dxx_install/samples/cmd` directory (where `dxx_install` is the directory in USS where the sample DTD, DAD, and XML files are located).

Table 5 on page 26 lists the USS and JCL samples that are provided to complete the getting started tasks.

Table 5. List of the XML collection lesson samples

Role	Description	USS command files	JCL file
administrator	Creates and populates the tables used for the lesson	Getstart_db.cmd	dxxgdb
administrator	Binds and enables the database	Getstart_prep.cmd	dxxgprep
administrator	Run a stored procedure to compose an XML document	Getstart_stp.cmd	dxxgstp
administrator	Exports a generated XML document from DB2	Getstart_export.cmd	dxxgxml
administrator	Cleanup the environment	Getstart_clean.cmd	dxxgclen

Setting up the lesson environment

In this section, you create the database and tables used for the sample data.

Creating the database

In this section, you use a command to set up the database. This command creates a sample database, connects to it, creates the tables to hold data, and then inserts the data.

Important: If you have completed the XML column lesson and have not cleaned up your environment, you might be able to skip this step. Check to see if you have a SALES_DB database.

To create the database:

1. Ensure that the database server has been enabled by the DB2 administrator. See “Initializing DB2 XML Extender” on page 39 to learn how to enable the server.
2. Change to the *dxx_install/samples/cmd* directory, where *dxx_install* is the directory in USS where the sample DTD, DAD, and XML files are located. The sample files contain references to files that use absolute path names. Check the sample files and change these values for your directory paths.
3. Run the create database command file, using one of the following methods:

odb2 command line: Enter the following command:

```
getstart_db.cmd
```

See “Choosing a method to run the Getting Started lessons” on page 12 to learn how to start the odb2 command line.

TSO: Submit the dxxgdb JCL job.

Creating the XML collection: preparing the DAD file

Because the data already exists in multiple tables, you will create an XML collection, which associates the tables with the XML document. To create an XML collection, you define the collection by preparing a DAD file.

In “Planning” on page 23 you determined which columns are in the relational database where the data exists, and how the data from the tables will be structured in an XML document. In this section, you create the mapping scheme in the DAD file that specifies the relationship between the tables and the structure of the XML document.

In the following steps, elements in the DAD are referred to as *tags* and the elements of your XML document structure are referred to as *elements*. A sample of a DAD file similar to the one you will create is in `dxx_install/samples/dad/getstart_xcollection.dad`. It has some minor differences from the file generated in the following steps. If you use it for the lesson, note that the file paths might be different than in your environment and you might need to update the sample file.

To create the DAD file for composing an XML document:

1. From the `dxx_install/samples/cmd` directory, open a text editor and create a file called `getstart_xcollection.dad`.
2. Create the DAD header, using the following text:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
```

Change `dxx_install` to the XML Extender home directory.

3. Insert the `<DAD></DAD>` tags. All other tags are located inside these tags.
4. Specify `<validation></validation>` tags to indicate whether the XML Extender validates the XML document structure when you insert a DTD into the DTD repository table. This lesson does not require a DTD and the value is `N0`.

```
<validation>N0</validation>
```

The value of the `<validation>` tags must be uppercase.

5. Use the `<Xcollection></Xcollection>` tags to define the access and storage method as XML collection. The access and storage methods define that the XML data is stored in a collection of DB2 tables.
6. After the `<Xcollection>` tag, provide an SQL statement to specify the tables and columns used for the XML collection. This method is called SQL mapping and is one of two ways to map relational data to the XML document structure. (See “Types of mapping schemes” on page 55 to learn more about mapping schemes.) Enter the following statement:

```
<Xcollection
<SQL_stmt>
  SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
  price, tax, ship_id, date, mode from order_tab o, part_tab p,
  table (select db2xml.generate_unique()
  as ship_id, date, mode, part_key from ship_tab) s
  WHERE o.order_key = 1 and
  p.price > 20000 and
  p.order_key = o.order_key and
  s.part_key = p.part_key
  ORDER BY order_key, part_key, ship_id
</SQL_stmt>
</Xcollection>
```

This SQL statement uses the following guidelines when using SQL mapping. Refer to Figure 5 on page 25 for the document structure.

- Columns are specified in top-down order, by the hierarchy of the XML document structure. For example, the columns for the order and customer elements are first, the part element are second, and the shipment are third.
- The columns for a repeating section, or non-repeating section, of the template that requires data from the database are grouped together. Each group has an object ID column: ORDER_KEY, PART_KEY, and SHIP_ID.
- The object ID column is the first column in each group. For example, O.ORDER_KEY precedes the columns related to the key attribute and p.PART_KEY precedes the columns for the Part element.
- The SHIP_TAB table does not have a single key conditional column, and therefore, the generate_unique user-defined function is used to generate the SHIP_ID column.
- The object ID columns are then listed in top-down order in an ORDER BY statements. The columns in ORDER BY should not be qualified by any schema and table name and should match the column names in the SELECT clause.

See “Mapping scheme requirements” on page 57 for requirements when writing an SQL statement.

7. Add the following prolog information to be used in the composed XML document:

```
<prolog?xml version="1.0"?</prolog>
```

This exact text is required for all DAD files.

8. Add the <doctype></doctype> tags to be used in the XML document you are composing. The <doctype> tag contains the path to the DTD stored on the client.

```
<doctype>!DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>
```

9. Define the root element of the XML document using the <root_node></root_node> tags. Inside the root_node, you specify the elements and attributes that make up the XML document.
10. Map the XML document structure to the DB2 relational table structure using the following three types of nodes:

element_node

Specifies the element in the XML document. Element_nodes can have child element_nodes.

attribute_node

Specifies the attribute of an element in the XML document.

text_node

Specifies the text content of the element and the column data in a relational table for bottom-level element_nodes.

See “The DAD file” on page 53 for more information about these nodes. Figure 5 on page 25 shows the hierarchical structure of the XML document and the DB2 table columns, and indicates what kinds of nodes are used. The shaded boxes indicate the DB2 table column names from which the data will be extracted to compose the XML document.

The following steps have you add each type of node, one type at a time.

- a. Define an <element_node> tag for each element in the XML document.

```
<root_node>
<element_node name="Order">
  <element_node name="Customer">
```

```

        <element_node name="Name">
        </element_node>
        <element_node name="Email">
        </element_node>
    </element_node>
    <element_node name="Part">
        <element_node name="key">
        </element_node>
        <element_node name="Quantity">
        </element_node>
        <element_node name="ExtendedPrice">
        </element_node>
        <element_node name="Tax">
        </element_node>
        <element_node name="Shipment" multi_occurrence="YES">
            <element_node name="ShipDate">
            </element_node>
            <element_node name="ShipMode">
            </element_node>
        </element_node> <!-- end Shipment -->
    </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>

```

Note that the <Shipment> child element has an attribute of multi_occurrence="YES". This attribute is used for elements without an attribute, that are repeated in the document. The <Part> element does not use the multi-occurrence attribute because it has an attribute of color, which makes it unique.

- b. Define an <attribute_node> tag for each attribute in your XML document. These attributes are nested in their element_node. The added attribute_nodes are highlighted in bold:

```

<root_node>
<element_node name="Order">
    <attribute_node name="key">
    </attribute_node>
    <element_node name="Customer">
        <element_node name="Name">
        </element_node>
        <element_node name="Email">
        </element_node>
    </element_node>
    <element_node name="Part">
        <attribute_node name="color">
        </attribute_node>
        <element_node name="key">
        </element_node>
        <element_node name="Quantity">
        </element_node>
    </element_node>
    ...
    </element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>

```

- c. For each bottom-level element_node, define <text_node> tags, indicating that the XML element contains character data to be extracted from DB2 when composing the document.

```

<root_node>
<element_node name="Order">
    <attribute_node name="key">
    </attribute_node>
    <element_node name="Customer">

```

```

<element_node name="Name">
  <text_node>
  </text_node>
</element_node>
<element_node name="Email">
  <text_node>
  </text_node>
</element_node>
</element_node>
<element_node name="Part">
  <attribute_node name="color">
  </attribute_node>
  <element_node name="key">
    <text_node>
    </text_node>
  </element_node>
  <element_node name="Quantity">
    <text_node>
    </text_node>
  </element_node>
  <element_node name="ExtendedPrice">
    <text_node>
    </text_node>
  </element_node>
  <element_node name="Tax">
    <text_node>
    </text_node>
  </element_node>
  <element_node name="Shipment" multi-occurrence="YES">
    <element_node name="ShipDate">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="ShipMode">
      <text_node>
      </text_node>
    </element_node>
  </element_node> <!-- end Shipment -->
</element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>

```

- d. For each bottom-level element_node, define a <column/> tag. These tags specify from which column to extract data when composing the XML document and are typically inside the <attribute_node> or the <text_node> tags. Remember, the columns defined here must be in the <SQL_stmt> SELECT clause.

```

<root_node>
<element_node name="Order">
  <attribute_node name="key">
    <column name="order_key"/>
  </attribute_node>
  <element_node name="Customer">
    <element_node name="Name">
      <text_node>
        <column name="customer_name"/>
      </text_node>
    </element_node>
    <element_node name="Email">
      <text_node>
        <column name="customer_email"/>
      </text_node>
    </element_node>
  </element_node>
  <element_node name="Part">
    <attribute_node name="color">

```



```

        <column name="color"/>
</attribute_node>
<element_node name="key">
  <text_node>
    <column name="part_key"/>
  </text_node>
<element_node name="Quantity">
  <text_node>
    <column name="quantity"/>
  </text_node>
</element_node>
<element_node name="ExtendedPrice">
  <text_node>
    <column name="price"/>
  </text_node>
</element_node>
<element_node name="Tax">
  <text_node>
    <column name="tax"/>
  </text_node>
</element_node>
<element_node name="Shipment" multi-occurrence="YES">
  <element_node name="ShipDate">
    <text_node>
      <column name="date"/>
    </text_node>
  </element_node>
  <element_node name="ShipMode">
    <text_node>
      <column name="mode"/>
    </text_node>
  </element_node>
</element_node> <!-- end Shipment -->
</element_node> <!-- end Part -->
</element_node> <!-- end Order -->
</root_node>

```

11. Ensure that you have an ending `</root_node>` tag after the last `</element_node>` tag.
12. Ensure that you have an ending `</Xcollection>` tag after the `</root_node>` tag.
13. Ensure that you have an ending `</DAD>` tag after the `</Xcollection>` tag.
14. Save the file as `getstart_xcollection.dad`

You can compare the file you have just created with the sample file `dxx_install/samples/dad/getstart_xcollection.dad`. This file is a working copy of the DAD file required to compose the XML document. The sample file contains location paths and file path names that might need to be changed to match your environment in order to be run successfully.

In your application, if you will use an XML collection frequently to compose documents, you can define a collection name by enabling the collection. Enabling the collection registers it in the XML_USAGE table and helps improve performance when you specify the collection name (rather than the DAD file name) when running store procedures. In these lessons, you will not enable the collection. To learn more about enabling collections, see “Enabling XML collections” on page 101.

Composing the XML document

In this step, you use the `dxxGenXML()` stored procedure to compose the XML document specified by the DAD file. This stored procedure returns the document as an XMLVARCHAR UDT.

To compose the XML document:

1. Use one of the following methods to call the dxxGenXML stored procedure:

Command line: Enter the following command:

```
getstart_stp.cmd
```

TSO: Submit the dxxgstp JCL job.

The stored procedure composes the XML document and stores it in the RESULT_TAB table.

You can see samples of stored procedures that can be used in this step in the following files:

- *dxx_install/samples/c/tests2x.sqc* shows how to call the stored procedure using embedded SQL and generates the texts2x executable file, which is used by the getstart_stp.cmd.
 - *dxx_install/samples/cli/sql2xml.c* shows how to call the stored procedure using the CLI.
2. Export the XML document from the table to a file using one of the following methods to call the XML Extender retrieval function, Content():

Command line:

- Enter the following commands:

```
DB2 CONNECT TO SALES_DB
```

```
DB2 SELECT DB2XML.Content(DB2XML.xmlVarchar(doc),  
    'dxx_install/samples/cmd/getstart.xml') FROM RESULT_TAB
```

- Or, run the following command file to export the file:

```
getstart_exportXML.cmd
```

TSO: Submit the dxxgexml JCL job.

Tip: This lesson teaches you how to generate one or more composed XML documents using DB2 stored procedure's result set feature. Using a result set allows you to fetch multiple rows to generate more than one document. As you generate each document, you can export it to a file. This method is the simplest way to demonstrate using result sets. For more efficient ways of fetching data see the CLI examples in the source file *dxx_install/samples/cli*.

Cleaning up the tutorial environment

If you want to clean up the tutorial environment, you can run one of the provided scripts or enter the commands from the command line to:

- Disable the XML column, ORDER
- Drop tables created in the tutorial
- Delete the DTD from the DTD reference table

They do not disable or drop the SALES_DB database; the database is still available for use with XML Extender. You might receive error messages if you have not completed both lessons in this chapter. You can ignore these errors.

To clean up the tutorial environment:

Run the cleanup command file, using one of the following methods:

- **Command line:** Enter the following command:

getstart_clean.cmd

TSO: Submit the dxxgeclen JCL job.

- If you want to disable the database, you can run the following XML Extender command from the command line:

```
dxxadm disable_server SALES_DB
```

This command drops the administration control tables DTD_REF and XML_USAGE, as well as removes the user-defined types and functions provided by XML Extender.

Part 2. Administration

This part describes how to perform administration tasks for the XML Extender.

Chapter 3. Preparing to use the XML Extender: administration

This chapter describes the requirements for setting up and planning for the XML Extender administration tasks.

Set-up requirements

The following sections describe set-up requirements for the XML Extender.

Software requirements

To use the XML Extender, you must have the following software installed.

- DB2 Universal Database for OS/390 and z/OS Version 7
- XML Toolkit for OS/390 and z/OS V1R2
- UNIX System Services set up; see *OS/390 UNIX System Services Planning*.
- Optional: odb2 command line to run the supplied samples in UNIX System Services (USS). Download the odb2 command line from the IBM OS/390 UNIX System Services Tools and Toys Web site. Go to:
<http://www.s390.ibm.com/products/oe/> and select **Tools and Toys**. See also the XML Extender Web site for new about changes to access of this product:
<http://www.ibm.com/software/data/db2/extenders/xmltext>
- To run the XML Extender administration wizard, the following software is required on a Windows or supported UNIX operating system server:
 - XML Extender administration wizard, downloaded to the client operating system
 - DB2 Universal Database Connect Personal or Enterprise Edition
 - Java Development Kit (JDK) 1.1.7 or higher

Before using the XML Extender, you must have the following z/OS and OS/390 options installed and set up:

- Workload Manager (WLM)
- Recoverable Resource Services (RRS)
- USS and Hierarchical File System (HFS)

Installation requirements

See the *Program Directory for IBM Database 2 Universal Database Server for OS/390 with National Language Versions* for information about software installation for the XML Extender for OS/390 and z/OS.

Table 6 lists the data sets that are installed with the XML Extender.

Table 6. The XML Extender data sets

data set	Description
SDXXADM	The XML Extender Administration Wizard, for execution on UNIX or Windows
SDXXC	C sample invoker source programs: <ul style="list-style-type: none">• INSERT {alias of DXXINS}• RETRIEVE {alias of DXXRET}• SHRED {alias of DXXSHR}• TESTS2X {alias of DXXTES}

Table 6. The XML Extender data sets (continued)

data set	Description
SDXXCLI	CLI samples Not currently available. Will be supported for General Availability.
SDXXCLP	CLP samples for use with the odb2 command line for running the Getting Started lessons, from USS
SDXXCMD	CMD samples for use with the odb2 command line for running the Getting Started lessons, from USS
SDXXDAD	The DAD files used in the Getting Started lessons
SDXXDBRM	The DBRMs for the XML Extender UDFs and stored procedures and the C sample invokers
SDXXDTD	The DTD for the XML Extender DAD and the DTD used in the Getting Started lessons
SDXXH	The C header files for use in C XML Extender invoker applications
SDXXJCL	<ul style="list-style-type: none"> • DXXGPREP - for initializing the XML Extender environment • The JCL jobs to run the Getting Started samples
SDXXJDBC	JDBC samples
SDXXLOAD	The XML Extender
SDXXXML	The XML files used in the Getting Started lessons

XML operating environment on OS/390 and z/OS

The following sections describe the XML operating environment for z/OS and OS/390.

Application programming

All the XML Extender facilities supplied for application programs run in the OS/390 MVS environment as stored procedures or user-defined functions (UDFs). Some of the UDFs that refer to the XMLFile data type, require access to an HFS system. The DB2 XML trace file, is also written to an HFS file.

Administration environment

You can use either an administration wizard from Windows or UNIX client, or an OS/390 and z/OS environment to complete administration tasks. This section describes the OS/390 and z/OS operating environment. See “Starting the administration wizard” on page 65 to learn how to use the administration wizard.

When performing administration tasks from the OS/390 and z/OS environment, you use the USS command line and HFS, or the MVS/TSO environment. The XML Extender installation creates sample files and executable files in MVS partitioned data sets. After these partitioned data sets are installed, it is recommended that you create HFS files in your USS environment by running the DXXGPREP JCL job. DXXGPREP runs essential bind steps, creates sample DB2 tables, and copies sample files to HFS. See “Initializing the XML Extender environment using

DXXGPREP” to learn what DXXGPREP does and how you should edit it before executing the job. Table 7 describes how the two OS/390 and z/OS environments are related.

Table 7. z/OS and OS/390 operating environment

Environment	MVS/TSO	USS
Sample files	DAD, DTD, and XML files, stored in partitioned data sets containing the sample files. Copied to USS by DXXGPREFP	DAD, DTD, and XML files are stored under the <i>dxx_install</i> directory. <ul style="list-style-type: none"> <i>dxx_install/dtd/dad.dtd</i> <i>dxx_install/samples/dtd/*.*</i> <i>dxx_install/samples/dad/*.*</i> <i>dxx_install/samples/xml/*.*</i> DXXGPREP copies sample files to USS using OPUT and MKDIR.
Program executable files	Stored in SDXXLOAD PDS, containing executable load modules for XML Extender	Turn on the sticky bit in HFS for the MVS/TSO SDXXLOAD file, to be able to access executable LOAD modules in this file from the USS command line.
Command scripts	Stored in SDXXJCL PDS, containing sample command JCL jobs executed in TSO batch	Command scripts are stored under the <i>dxx_install</i> directory. DXXGPREP copies sample files to USS from SDXXCMD PDS, using MKDIR and OPUT.
Command environment	MVS batch or TSO batch	USS command line and odb2 command line

After the USS environment is initialized, you can use either the USS command line and odb2 command line, or MVS batch and TSO batch to perform administration tasks.

MVS batch and TSO batch

Running the sample JCL will execute the sample programs in the MVS/TSO environment. The JCL accesses the sample files stored in the USS file system (DAD, DTDs, XML documents).

USS command line and odb2 command line

You can run command scripts used for the getting started lessons from the odb2 command line. These scripts can be used as models for running administration commands in your application environment. To run programs from the odb2 command line, you need to:

- Download and install the odb2 command line.
- Turn on the sticky bit in HFS for the MVS/TSO SDXXLOAD file, so that SDXXLOAD is accessible to your USS environment.

Initializing DB2 XML Extender

The following sections describe the initialization procedures required to use the DB2 XML Extender.

Initializing the XML Extender environment using DXXGPREP

DXXGPREP is a JCL job script used to initialize the database server for XML. This task is performed by a DB2 administrator for each DB2 system where you want to use the XML Extender.

Scripts and sample XML documents (DTD, DAD, and XML files) are provided for you to use in the Getting Started lessons. To install the scripts and sample files, run the DXXGPREP job.

To initialize the database server:

- Read the following sections for planning considerations:
 - “Workload management considerations” on page 41
 - “Table space considerations when enabling a database server” on page 42
- Edit the DXXGPREP job script in the XML Extender JCL PDS (SDXXJCL) in accordance with local requirements and naming conventions.

Bind change when running in USS: When you enable an XML column or collection and specify a DAD file that is stored in HFS, modify the BIND command for the DXXADM package with the ENCODING option. The ENCODING option must specify the actual code page of the DAD file. For example, if the DAD file has a code page of 1047, specify this value on the ENCODING option:

```
BIND PACKAGE (DB2XML) MEMBER(DXXADM) ENCODING(1047);
```

- Submit the DXXGPREP JCL file.

DXXGPREP specifies the following values:

- The utility program, DSNTEP2, for issuing SQL statements
- The schema name, DB2XML
- Two XML Extender collections:

DB2XML

For XML Extender administration, such as enabling and disabling XML columns or collections

DB2XML_RUN

For running the supplied XML Extender sample invoker applications in JCL PDS (SDXXDBRM)

Tasks that the DXXGPREP job performs:

- Creates the following objects:
 - The LOB table spaces, XMLLOBTS and XMLLOBT2, for the CLOB data in the DTD_REF table, which must be specified on the corresponding enable_server command
 - The XML Extender stored procedure that enables the database server, DXXENABLEDB
- Performs the necessary binds for the XML Extender and creates the following plans with ownership, DB2XML:
 - Plan DXXADM for performing administration through DXXADM. This plan uses the collection DB2XML and includes the ODBC packages
 - Plans INSERT, RETRIEVE, SHRED, TESTS2X for running the Getting Started sample applications. This plan uses the collection DB2XML_RUN and includes the ODBC packages
- Grants to user DB2XML:
 - DBADM rights with grant option on schema DB2XML
 - Required rights to the catalog.
 - BINDADD and PACKADM to the two XML Extender collections, DB2XML and DB2XML_RUN
- Grants to public EXECUTE and BIND on:

- Collection DB2XML_RUN
- Plan DXXADM, INSERT, RETRIEVE, SHRED, and TESTS2X
- Enables the DB2 system to XML specifying the following options:
ENABLE_SERVER -a V71A using XMLLOBTS,XMLLOBT2 wlm environment WLMENV1

Important: The ENABLE_SERVER command must be run by SYSADM or DB2XML or users with equivalent authority. See “enable_server” on page 143 for the full syntax before submitting DXXGPREP.

After editing and running DXXGPREP, it is possible to work through the Getting Started lessons in this book, to try the samples, provided you have set up the following OS/390 and z/OS options:

- The WLM environment
- The RRS environment
- The HFS file system

You can use either JCL or the USS command line to run the Getting Started samples.

Initializing the XML Extender administration wizard

If you want to use the XML Extender administration wizard from a Windows or UNIX environment, complete the following initialization steps:

- Download the file DXXADMIN in JCL PDS (SDXXADM) to your PC.
- Unzip the packaging using a zip tool, such as PKUNZIP.
You can use FTP to download the file, but you must remember to specify BIN.
A Java jar file is extracted, which runs the XML Extender administration wizard.
- Set up the jar files and CLASSPATH as described in “Setting up the administration wizard” on page 65.
- Ensure that you have the following applications installed:
 - DB2 Connect Personal or Enterprise Edition
 - JDK 1.1.7 or higher

Workload management considerations

A Work Load Management (WLM) Application Environment is a set of parameters describing how to create address spaces which can run a particular kind of work. The XML Extender uses WLM Application Environments for user-defined functions and for stored procedures. DB2 allows various options for stored procedures, but to attain all DB2 functionality that is required, XML Extender UDFs and stored procedures must use WLM environments.

After the XML Extender is installed, you need to establish WLM environments for the UDFs and stored procedures. When you enable a database server for the XML Extender, you specify the name application environment that DB2 should use for XML Extender UDFs and stored procedures. You can enable the server by using the DXXADM ENABLE_SERVER command (see “enable_server” on page 143), by using the administration wizard, or by running the DXXGPREP JCL job file as described in “Initializing the XML Extender environment using DXXGPREP” on page 39. See the DB2 for OS/390 and z/OS library documentation installation and administration books for additional installation, setup, and troubleshooting information. Each WLM environment is associated with a JCL procedure that starts an address space for executing the XML Extender UDFs or stored procedures.

You need to decide how many WLM environments to establish. You also need to decide what performance objectives to specify for these environments.

The number of WLM environments

You can establish multiple WLM environments for running XML Extender UDFs. When you enable a database server for the XML Extender, you specify the WLM environment names (see “enable_server” on page 143). If you specify a single WLM environment name, then all of the XML Extender’s UDFs run in that WLM environment. If you specify two WLM environment names, all stored procedures will be run in the first WLM environment, all UDFs will be run in the second environment. Using two separate WLM environments for stored procedures and UDFs can improve performance.

Performance objectives for WLM environments

WLM can operate in either of two modes: compatibility mode or goal mode. In compatibility mode, work requests are given a service class by the classification rules in the active WLM service policy.

In goal mode, work requests are also assigned a service class by the classification rules in the active WLM service policy. However each service class period has a performance objective, that is, a goal. WLM raises or lowers that period’s access to system resources as needed to meet the specified goal. For example, the goal might be “application APPL8 should run in less than 3 seconds of elapsed time 90 percent of the time”.

Specify goal mode: In goal mode, WLM automatically starts WLM-established address spaces for user-defined functions to help meet the service class goals that you set. By comparison, in compatibility mode, WLM cannot automatically start a new address space to handle high-priority requests. Instead, you must monitor the performance of UDFs to determine how many WLM-managed address spaces are needed, and the operator must start and stop them manually. As a result, goal mode is recommended for running XML Extender UDFs.

Table space considerations when enabling a database server

The XML Extender stores data on DTD and XML document usage in administrative support tables that are contained in DB2 table spaces. When you enable a database server for the XML Extender, you specify the table spaces for CLOB content of each of the administrative support tables. You can enable the server by using the DXXADM ENABLE_SERVER command (see “enable_server” on page 143), by using the administration wizard, or by running the DXXGPREP JCL job file as described in “Initializing the XML Extender environment using DXXGPREP” on page 39.

When you enable using the DXXGPREP job file, the table spaces are created for you and the default names are used. You can edit the job for your application. When you enable using the DXXADM ENABLE_SERVER command or the administration wizard, you must create the table spaces in advanced and specify the table spaces in the command syntax, as described in “enable_server” on page 143 or in “Enabling a database server for XML” on page 69.

Additionally, you can also create and specify table spaces for XML column application and side tables when you enable an XML column. You enable a column using the DXXADM ENABLE_COLUMN command or using the administration wizard.

The following sections describe table space considerations when you create your own table spaces for the administration tables and for XML column application and side tables.

Creating table spaces for the administration support tables

The table space that you specify for the administration support tables, should be in the default database, DSNDDB04, which is created as part of the setup done after the XML Extender is installed. The default table spaces are XMLLOBTS and XMLLOBT2. Specify a table space for the administrative support tables with a 4K buffer pool.

Creating table spaces for XML tables

When you enable a column for the XML Extender, you can specify a table space for side tables. The table space should be a segmented table space. Specify LOCKSIZE ROW when you create the table space if you expect any enable or disable operations to occur frequently, occur in complex transactions, or occur in units of work that are not immediately committed. Specify a table space for XML column application and side tables with a 4K buffer pool.

Security considerations

Before you use the XML Extender, you must consider the implications the XML Extenders have on security. For example, you need to determine what controls (if any) to put in place for access to XML column data. You also need to determine whether you want to restrict privileges that the XML Extender automatically grant to users.

Access to XML columns in tables

XML documents stored as columns in a DB2 database are afforded the same security protection as traditional numeric and character data. Users must have the required privilege to select objects from, insert objects into, update objects in, or delete objects from a DB2 database. For example, to select objects from a user table, a user must have SELECT privilege on the table. For information about DB2 security, see the *DB2 Universal Database for OS/390 and z/OS Administration Guide, Version 7*.

Access to content in files

The XML documents, DADs, and DTDs of XMLFile type, that you store in a table, can point to content stored in files including external entities or DTDs. The files can be in a partitioned data set or in file system that is compatible with OS/390 and z/OS USS, for example, HFS.

When an administrator enables a database server for the XML Extender, the administrator can specify a SECURITY option (see “enable_server” on page 143). The option indicates how UDFs that store, retrieve, and update objects interact with an external security product such as RACF to control access to files. The administrator can specify SECURITY USER or SECURITY DB2.

If SECURITY USER is specified, the XML Extender UDF execution environment is assigned the primary authorization ID of the process that called the function. This is the ID that is used for non-SQL requests. The primary authorization ID of the process is used rather than other DB2 authorization IDs, such as the authorization ID of the package or plan owner. The primary authorization ID is subject to distributed database security operations such as inbound authorization ID translation.

If SECURITY DB2 is specified, the XML Extender UDFs access files using the authorization ID associated with the WLM environment address spaces that are established for running the UDFs. In this case, all XML Extender users have access to the same files. If you use two WLM environments, then a separate ID can be assigned to each environment.

When a UDF attempts to access a file, USS calls an external security product such as Security Server (RACF) to get the user ID (UID) and group ID (GID) associated with the UDF. For SECURITY USER, the UID and GID are those that are assigned to the authorization ID in effect for the process that calls the UDF. For SECURITY DB2, the UID and GID are those that are assigned to the authorization ID of the WLM application environment address spaces for the UDF. The system then compares the UID and GID assignments to the user, group, and other permission bits in the file's directory entry. The file can be accessed only if the user's UID and GID are compatible with the permissions in the file's directory entry.

SECURITY USER gives greater control over file access: If you specify SECURITY USER, file system checks are made against the primary authorization ID of the process that calls the UDF. Because you can assign different UIDs and GIDs to different users, you can control access to files on a user-by-user or group-by-group basis. By comparison, SECURITY DB2 gives you one level of control because all UDFs run with the same UID and GID, that is, the UID and GID assigned to the WLM environment address spaces. For this reason, SECURITY DB2 is a good choice for applications where file read protection is not required, for example Web applications.

SECURITY DB2 requires less administration: If you specify SECURITY DB2, you need to assign an authorization ID, UID, and GID to the WLM address spaces for the XML Extender UDFs. By comparison, if you specify SECURITY USER, you must assign a UID and GID for every legitimate user of the files. In both cases, you need to coordinate the UID and GID assignments with the file system permissions.

SECURITY DB2 results in better UDF performance: This is because the individual nature of performing the security checks for SECURITY USER incurs more overhead in the database server than SECURITY DB2.

For more information about security, see the XML Extender section of the *Program Directory for IBM Database 2 Universal Database Server for OS/390 with National Language Versions*.

EXECUTE authority

When a database server is enabled for the XML Extender, use privilege on the XML Extender's UDT (and related CAST functions) and use privilege on all of its UDFs and stored procedures are granted to PUBLIC. You can revoke the use privilege on the UDT and UDFs that was granted to PUBLIC, and grant the privilege to use the UDT and UDFs to specific authorization IDs. This does not affect the way the XML Extender operates. However maintaining authority lists could become tedious; consider controlling access to files that are used (or potentially used) by XML Extender UDFs. In effect, this method limits the ability to successfully retrieve objects of the associated user-defined type to specific authorization IDs.

This has implications for external security: If you specify SECURITY DB2, UDF access to files is controlled by authorization ID, UID, and GID specifications made for the WLM environment address spaces in which the UDFs run.

Authority to administer the XML Extender

Some XML Extender-related administrative operations require special authority.

Enable and disable server

It is recommended that an administrator have SYSADM to DBADM authority for the user ID DB2XML to enable or disable a server.

Enable and disable column

An administrator must have table owner privileges on the table containing the column to be enabled, and must have privileges for buffer pools and table spaces.

Enable and disable collection

An administrator must have table owner privileges for all existing tables in the collection and use on the buffer pools and table spaces.

Composition and decomposition

An application developer needs access to the following tables:

Composition

- DTD_REF (to access the DTD)
- XML_USAGE (to access the DAD files)
- SELECT on all tables referenced
- INSERT on the result table

Decomposition

- DTD_REF (to access the DTD)
- XML_USAGE (to access the DAD files)
- INSERT on all tables to be modified
- UPDATE on all tables to be modified
- SELECT on any table referenced in the DAD file
- Access to the DB2 catalog

Backup and recovery considerations

You need to back up the **DSNDB04** database, which contains the tables spaces for the administration support tables used by the XML Extender. The administration support tables, XML_USAGE and DTD_REF, are required for many XML Extender activities; they are essential and should always be backed up. For further information about the XML Extender initialization, see “Initializing the XML Extender environment using DXXGPREP” on page 39.

Administration tools

The XML Extender provides several methods for administration: the XML Extender administration command and the XML Extender *stored procedures*. You can also use the XML Extender administration wizard, if you have the XML Extender installed on a client workstation.

- The administration command, **dxxadm**, provides subcommands for the various administration tasks. Use of this command is described in the administration tasks in “Chapter 5. Managing the database server” on page 69 and in “Chapter 10. XML Extender administration command: DXXADM” on page 141.
- The administration stored procedures also provide options for various administration tasks. These stored procedures are described in “Administration stored procedures” on page 191.

- The XML Extender administration wizard prompts you through the administration tasks. You can use it from a client workstation to perform these tasks. Use of this tool is described in “Chapter 5. Managing the database server” on page 69.

Administration planning

When planning an application that uses XML documents, you first need to make the following design decisions:

- If you will be composing XML documents from data in the database
- If you will be storing existing XML documents, and if you want them to be stored as intact XML documents in a column or decomposed into regular DB2 data

After you make these decisions, you can then plan the rest of your administration tasks:

- Whether to validate your XML documents
- Whether to index XML column data for fast search and retrieval
- How to map the structure of the XML document to DB2 relational tables

How you use the XML Extender depends on what your application requires. As indicated in “Chapter 1. Introduction to the XML Extender” on page 3, you can compose XML documents from existing DB2 data and store XML documents in DB2, either as intact documents or as DB2 data. Each of these storage and access methods have different planning requirements. The following sections discuss each of these planning considerations.

Choosing an access and storage method

The XML Extender provides two access and storage methods to use DB2 as an XML repository: XML column and XML collection. You first need to decide which of the methods best matches your application needs for accessing and manipulating XML data.

XML column

Stores and retrieves entire XML documents as DB2 column data. The XML data is represented by an XML column.

XML collection

Decomposes XML documents into a collection of relational tables or composes XML documents from a collection of relational tables.

The nature of your application determines the type of access and storage method to use and how to structure your XML data. The following scenarios describe situations in which each access and storage method is the most appropriate.

When to use XML columns

Use XML columns in the following situations:

- The XML documents already exist or come from some external source and you prefer to store the documents in the native XML format. You want to store them in DB2 for integrity and for archival and auditing purposes.
- The XML documents are generally read, but not updated.
- You want to use file name data types to store the XML documents external to DB2 in the local or remote file system and to use DB2 for management and search operations.
- You need range search based on the values of XML elements or attributes, and you know what elements or attributes will frequently be the search arguments.

- The documents have elements with large text blocks and you want to use the DB2 Text Extender for structural text search while keeping the entire documents intact.

When to use XML collections

Use XML collections in the following situations:

- You have data in your existing relational tables and you want to compose XML documents based on a certain DTD.
- You have XML documents that need to be stored with collections of data that map well to relational tables.
- You want to create different views of your relational data using different mapping schemes.
- You have XML documents that come from other data sources. You care about the data but not the tags, and want to store pure data in your database. You want the flexibility to decide whether to store the data in some existing tables or in new tables.
- A small subset of your XML documents needs to be updated often, and update performance is critical.
- You need to store the data of entire incoming XML documents but often only want to retrieve a subset of them.

You use the document access definition (DAD) file to associate XML data with DB2 tables through these two access and storage methods. Figure 6 shows how the DAD specifies the access and storage methods.

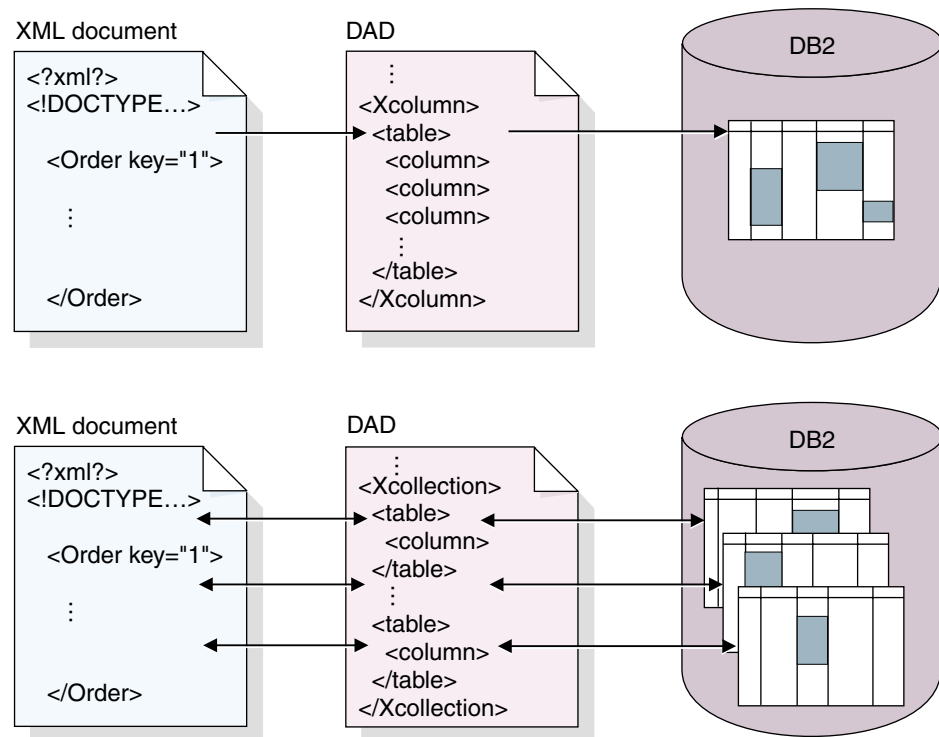


Figure 6. The DAD file maps the XML document structure to DB2 and specifies the access and storage method.

The DAD file is an important part of administering the XML Extender. It defines the location of key files like the DTD, and specifies how the XML document structure relates to your DB2 data. Most important, it defines the access and storage methods you use in your application.

Planning for XML columns

Before you begin working with the XML Extender to store your documents, you need to understand the structure of the XML document so that you can determine how to index elements and attributes in the document. When planning how to index the document, you need to determine:

- The XML user-defined type in which you will store the XML document
- The XML elements and attributes that your application will frequently search, so that their content can be stored in side tables and indexed to improve performance
- Whether or not to validate XML documents in the column with a DTD
- The structure of the side tables and how they will be indexed

Determining the XML data type for the XML column

The XML Extender provides XML user defined types in which you define a column to hold XML documents. These data types are described in Table 8.

Table 8. The XML Extender UDTs

User-defined type column	Source data type	Usage description
XMLVARCHAR	VARCHAR(<i>varchar_len</i>)	Stores an entire XML document as VARCHAR inside DB2. Used for small documents stored in DB2.
XMLCLOB	CLOB(<i>clob_len</i>)	Stores an entire XML document as CLOB inside DB2. Used for large documents stored in DB2.
XMLFILE	VARCHAR(1024)	Stores the file name of an XML document in DB2, and stores the XML document in a file local to the DB2 server. Used for documents stored outside DB2.

Determining elements and attributes to be indexed

When you understand the XML document structure and the needs of the application, you can determine which elements and attributes to be searched. These are usually the elements and attributes that will be searched or extracted most frequently, or those that will be the most expensive to query. The location paths of each element and attribute can be mapped to relational tables (side tables) that contain the content of these objects, in the DAD file for XML columns. The side tables are then indexed.

For example, Table 9 on page 49 shows an example of types of data and location paths of element and attribute from the Getting Started scenario for XML columns. The data was specified as information to be frequently searched and the location paths point to elements and attributes that contain the data. These location paths can then be mapped to side tables in the DAD file.

Table 9. Elements and attributes to be searched

Data	Location path
order key	/Order/@key
customer	/Order/Customer/Name
price	/Order/Part/ExtendedPrice
shipping date	/Order/Part/Shipment/ShipDate

Planning side tables

Side tables are DB2 subtables used to extract the content of an XML document that will be searched frequently. The location path of the element or attribute is mapped to a table and column, indexed, and used for searches. When the XML document is updated in the application table, the values in the side tables are automatically updated.

Figure 7 shows an XML column with side tables.

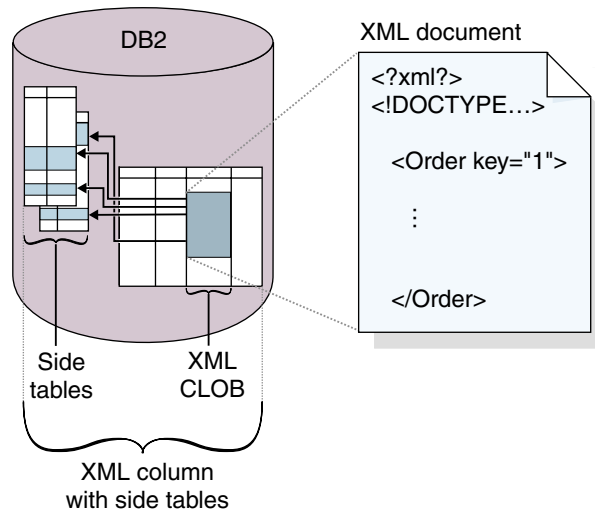


Figure 7. An XML column with side tables

When planning for side tables, you must consider how to organize the tables, how many tables to create, and whether to create a default view for the side tables. These decisions are partly based on several issues: whether elements and attributes can occur multiple times, and the requirements for query performance. Additionally, do not plan to update the side tables in any way; they will be automatically updated when the document is updated in the XML column.

Multiple occurrence: When using multiple occurring location paths, consider the following issues in your planning:

- For elements or attributes in an XML document that have *multiple occurrences*, you must create a separate side table for each XML element or attribute with multiple occurrences, due to the complex structure of XML documents. This means that elements or attributes that have multiple occurring location paths must be mapped to a table with only one column, the column for that location path. You cannot have any other columns in the table, whether or not they have multiple occurrence.

- When a document has multiple occurring location paths, XML Extender will add a column DXX_SEQNO of type INTEGER in each side table to keep track of the order of elements that occur more than once. With DXX_SEQNO, you can retrieve a list of the elements using the same order as the original XML document by specifying ORDER BY DXX_SEQNO in an SQL query.

Default views and query performance: When you enable an XML column, you can specify a default, read-only view that joins the application table with the side tables using a unique ID, called the ROOT ID. With the default view, you can search XML documents by querying the side tables. For example, if you have the application table SALES_TAB, and the side tables ORDER_TAB, PART_TAB and SHIP_TAB:

```
SELECT sales_person FROM sales_order_view
      WHERE price > 2500.00
```

The SQL statement returns the names of sales people in SALES_TAB who have orders stored in the column ORDER, and where the PRICE is greater than 2500.00.

The advantage of querying the default view is that it provides a virtual single view of the application table and side tables. However, the more side tables that are created, the more expensive the query. Therefore, creating the default view is only recommended when the total number of side-table columns is small. Applications can create their own views, joining the important side table columns.

Indexes for XML column data

An important planning decision is whether to index your XML column document. This decision should be made based on how often you need to access the data and how critical performance is during structural searches.

When using XML columns, which contain entire XML documents, you can create side tables to contain columns of XML element or attribute values, then create indexes on these columns. You must determine for which elements and attributes you need to create the index.

XML column indexing allows frequently queried data of general data types, such as integer, decimal, or date, to be indexed using the native DB2 index support from the database engine. The XML Extender extracts the values of XML elements or attributes from XML documents and stores them in the side tables, allowing you to create indexes on these side tables. You can specify each column of a side table with a location path that identifies an XML element or attribute and an SQL data type.

The XML Extender automatically populates the side table when you store XML documents in the XML column.

For fast search, create indexes on these columns using the DB2 *B-tree indexing* technology. The methods that are used to create an index vary on different operating systems, and the XML Extender supports these methods.

Considerations:

- For elements or attributes in an XML document that have *multiple occurrences*, you must create a separate side table for each XML element or attribute with multiple occurrences due to the complex structure of XML documents.
- You can create multiple indexes on an XML column.
- You can associate side tables with the application table using the ROOT ID, the column name of the primary key in the application table and a unique identifier

that associates all side tables with the application table. You can decide whether you want the primary key of the application table to be the ROOT ID, although it cannot be the composite key. This method is recommended.

If the single primary key does not exist in the application table, or for some reason you don't want to use it, the XML Extender alters the application table to add a column DXXROOT_ID, which stores a unique ID that is created at the insertion time. All side tables have a DXXROOT_ID column with the unique ID. If the primary key is used as the ROOT ID, all side tables have a column with the same name and type as the primary key column in the application table, and the values of the primary keys are stored.

- If you enable an XML column for the DB2 Text Extender, you can also use the Text Extender's structural-text feature. The Text Extender has "section search" support, which extends the capability of a conventional full-text search by allowing search words to be matched within a specific document context that is specified by location paths. The *structural-text index* can be used with the XML Extender's indexing on general SQL data types.

Validation

After you choose an access and storage method, you can determine whether to *validate* the XML documents that are stored in the column. You validate XML data using a DTD. The DTD is stored in the DTD repository, or can be stored in the file system that the DB2 server has access to.

Recommendation: Validate XML data with a DTD, unless you are storing XML documents for archival purposes. To validate, you need to have a DTD in the XML Extender repository. See "Storing a DTD in the DTD repository table" on page 70 to learn how to insert a DTD into the repository.

You can validate documents in the same XML column using different DTDs. In other words, you can have documents that have a similar structure, with similar elements and attributes, that call DTDs that are different. To reference multiple DTDs, use the following guidelines:

- The system ID of the XML document in the DOCTYPE definition must specify the DTD file using a full path name.
- You must specify YES for validation in the DAD file.
- At least one of the DTDs must be stored in the DTD_REF table. All of the DTDs can be stored in this table.
- The DTDs should have a common structure, with differences only in subelements.
- The DAD file should specify elements or attributes that are common to all of the DTDs referenced by documents in that column.

Important: Make the decision whether to validate before inserting XML data into DB2. The XML Extender does not support the validation of data that has already been inserted into DB2.

Considerations:

- If you do not choose to validate a document, the DTD specified by the XML document is not processed. It is important that DTDs be processed to resolve entity values and attribute defaults even when processing document fragments that cannot be validated.
- You do not need a DTD to store or archive XML documents.
- Validating your XML data might have a small performance impact.

- You can use multiple DTDs, but can only index common elements and attributes.

The DAD file

For XML columns, the DAD file primarily specifies how documents that are stored in an XML column are to be indexed, and is an XML-formatted document, residing at the client. The DAD file specifies a DTD to use for validating documents inserted into the XML column. The DAD file has a data type of CLOB. This file can be up to 100 KB.

The DAD file for XML columns contains an XML header, specifies the directory paths on the client for the DAD file and DTD, and provides a map of any XML data that is to be stored in side tables for indexing.

To specify the XML column access and storage method, you use the following tag in the DAD file.

<Xcolumn>

Specifies that the XML data is to be stored and retrieved as entire XML documents in DB2 columns that are enabled for XML data.

An XML-enabled column is of the XML Extender's UDT. Applications can include the column in any *user table*. You access the XML column data mainly through SQL statements and the XML Extender's UDFs.

You can use the XML Extender administration wizard or an editor to create and update the DAD.

Planning for XML collections

When planning for XML collections, you have different considerations for composing documents from DB2 data, decomposing XML document into DB2 data, or both. The following sections address planning issues for XML collections, and address composition and decomposition considerations.

Validation

After you choose an access and storage method, you can determine whether to validate your data. You validate XML data using a DTD. Using a DTD ensures that the XML document is valid and lets you perform structured searches on your XML data. The DTD is stored in the DTD repository.

Recommendation: Validate XML data with a DTD. To validate, you need to have a DTD in the XML Extender repository. See "Storing a DTD in the DTD repository table" on page 70 to learn how to insert a DTD into the repository. The DTD requirements differ depending on whether you are composing or decomposing XML documents. The following list describes these requirements:

- For composition, you can only validate generated XML documents against one DTD. The DTD to be used is specified in the DAD file.
- For decomposition, you can validate documents for composition using different DTDs. In other words, you can decompose documents, using the same DAD file, but call DTDs that are different. To reference multiple DTDs, you must use the following guidelines:
 - At least one of the DTDs must be stored in the DTD_REF table. All of the DTDs can be stored in this table.
 - The DTDs should have a common structure, with differences in subelements.
 - You must specify validation in the DAD file.

- The SYSTEM ID of the XML document must specify the DTD file using a full path name.
- The DAD file contains the specification for how to decompose the document, and therefore, you can specify only common elements and attributes for decomposition. Elements and attributes that are unique to a DTD cannot be decomposed.

Important: Make the decision whether to validate XML data before inserting XML data into DB2. The XML Extender does not support the validation of data that has already been inserted into DB2.

Considerations:

- You should use a DTD when using XML as interchange format.
- Validating your XML data might have a small performance impact.
- You can decompose only common elements and attributes when using multiple DTDs for decomposition.
- You can decompose all elements and attributes when using one DTD.
- You can use only one DTD for composition.

The DAD file

For XML collections, the DAD file maps the structure of the XML document to the DB2 tables from which you either compose the document, or to where you decompose the document.

For example, if you have an element called <Tax> in your XML document, you might need to map <Tax> to a column called TAX. You define the relationship between the XML data and the relational data in the DAD.

The DAD file is specified either while enabling a collection, or when you use the DAD file in XML collection *stored procedures*. The DAD is an XML-formatted document, residing at the client. If you choose to validate XML documents with a DTD, the DAD file can be associated with that DTD. When used as the input parameter of the XML Extender stored procedures, the DAD file has a data type of CLOB. This file can be up to 100 KB.

To specify the XML collection access and storage method, you use the following tag in the DAD file:

<Xcollection>

Specifies that the XML data is either to be decomposed from XML documents into a collection of relational tables, or to be composed into XML documents from a collection of relational tables.

An XML collection is a virtual name for a set of relational tables that contains XML data. Applications can enable an XML collection of any user tables. These user tables can be existing tables of legacy business data or tables that the XML Extender recently created. You access XML collection data mainly through the stored procedures that the XML Extender provides.

The DAD file defines the XML document tree structure, using the following kinds of nodes:

root_node

Specifies the root element of the document.

element_node

Identifies an element, while can be the root element or a child element.

text_node

Represents the CDATA text of an element.

attribute_node

Represents an attribute of an element.

Figure 8 shows a fragment of the mapping that is used in a DAD file. The nodes map the XML document content to table columns in a relational table.

```

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
  ...
  <Xcollection>
  <SQL_stmt>
    ...
  </SQL_stmt>
  <prolog?xml version="1.0"?/prolog>
  <doctype!DOCTYPE Order SYSTEM "dxx_install/sample/dtd/getstart.dtd"</doctype>
  <root_node>
    <element_node name="Order">          --> Identifies the element <Order>
      <attribute_node name="key">        --> Identifies the attribute "key"
        <column name="order_key"/>      --> Defines the name of the column,
                                          "order_key", to which the element and
                                          attribute are mapped
      </attribute_node>
      <element_node name="Customer">    --> Identifies a child element of
                                          <Order> as <Customer>
        <text_node>                     --> Specifies the CDATA text for the element
                                          <Customer>
          <column name="customer">      --> Defines the name of the column, "customer",
                                          to which the child element is mapped
        </text_node>
      </element_node>
      ...
    </element_node>
    ...
  </root_node>
</Xcollection>
</DAD>

```

Figure 8. Node definitions

In this example, the first two columns in the SQL statement have elements and attributes mapped to them. The XML Extender also supports processing instructions for stylesheets, using the `<stylesheet>` element. It must be inside the root node of the DAD file, with the doctype and prolog defined for the XML document. For example:

```

<Xcollection>
  ...
  <prolog>...</prolog>
  <doctype>...</doctype>
  <stylesheet?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
  <root_node>...</root_node>
  ...
</Xcollection>

```

You can use the XML Extender administration wizard or an editor to create and update the DAD file. The `<stylesheet>` element is not currently supported by the XML Extender administration wizard.

Mapping schemes for XML collections

If you are using an XML collection, you must select a *mapping scheme* that defines how XML data is represented in a relational database. Because XML collections must match a hierarchical structure that is used in XML documents with a relational structure, you should understand how the two structures compare. Figure 9 shows how the hierarchical structure can be mapped to relational table columns.

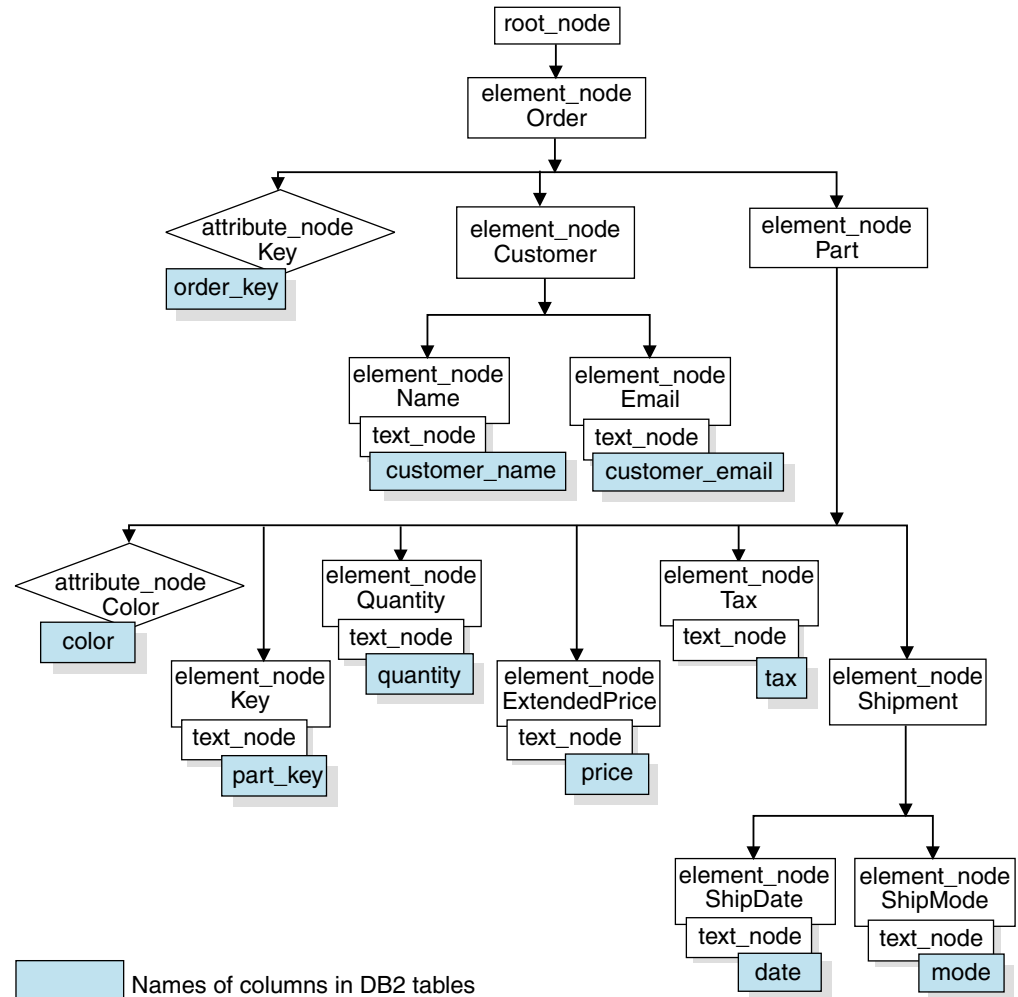


Figure 9. XML document structured mapped to relational table columns

The XML Extender uses the mapping scheme when composing or decomposing XML documents that are located in multiple relational tables. The XML Extender provides a wizard that assists you in creating the DAD file. However, before you create the DAD file, you must think about how your XML data is mapped to the XML collection.

Types of mapping schemes: The mapping scheme is specified in the `<Xcollection>` element in the DAD file. The XML Extender provides two types of mapping schemes: *SQL mapping* and *Relational Database (RDB_node) mapping*. Both methods use the XPath model to define the hierarchy of the XML document.

SQL mapping

Allows direct mapping from relational data to XML documents through a single SQL statement and the *XPath data model*. SQL mapping is used for

composition; it is not used for decomposition. SQL mapping is defined with the `SQL_stmt` element in the DAD file. The content of the `SQL_stmt` is a valid SQL statement. The `SQL_stmt` maps the columns in the `SELECT` clause to XML elements or attributes that are used in the XML document. When defined for composing XML documents, the column names in the SQL statement's `SELECT` clause are used to define the value of an *attribute_node* or a content of *text_node*. The `FROM` clause defines the tables containing the data; the `WHERE` clause specifies the *join* and search *condition*.

The SQL mapping gives DB2 users the power to map the data using SQL. When using SQL mapping, you must be able to join all tables in one `SELECT` statement to form a query. If one SQL statement is not sufficient, consider using `RDB_node` mapping. To tie all tables together, the *primary key* and *foreign key* relationship is recommended among these tables.

RDB_node mapping

Defines the location of the content of an XML element or the value of an XML attribute so that the XML Extender can determine where to store or retrieve the XML data.

This method uses the XML Extender-provided *RDB_node*, which contains one or more node definitions for tables, optional columns, and optional conditions. The tables and columns are used to define how the XML data is to be stored in the database. The condition specifies the criteria for selecting XML data or the way to join the XML collection tables.

To define a mapping scheme, you create a DAD with an `<Xcollection>` element. Figure 10 on page 57 shows a fragment of a sample DAD file with an XML collection SQL mapping that composes a set of XML documents from data in three relational tables.

```

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
  <dtdid>dxx_install/samples/dad/getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
      SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
             ship_id, mode, comment
      FROM order_tab o, part_tab p,
           table(select db2xml.generate_unique()
                as ship_id, date, mode, from ship_tab)
      S
      WHERE p.price > 2500.00 and s.date > "1996-06-01" AND
            p.order_key = o.order_key and s.part_key = p.part_key
    </SQL_stmt>
    <prolog?xml version="1.0"?</prolog>
    <doctype!DOCTYPE DAD SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>
    <root_node>
      <element_node name="Order">
        <attribute_node name="key">
          <column_name="order_key"/>
        </attribute_node>
        <element_node name="Customer">
          <text_node>
            <column name="customer"/>
          </text_node>
        </element_node>
      ...
    </element_node><!--end Part-->
  </element_node><!--end Order-->
  </root_node>
</Xcollection>
</DAD>

```

Figure 10. SQL mapping scheme

The XML Extender provides several stored procedures that manage data in an XML collection. These stored procedures support both types of mapping, but require that the DAD file follow the rules that are described in “Mapping scheme requirements”.

Mapping scheme requirements: The following sections describe requirements for each type of the XML collection mapping schemes.

Requirements when using SQL mapping

In this mapping scheme, you must specify the SQL_stmt element in the DAD <Xcollection> element. The SQL_stmt should contain a single SQL statement that can join multiple relational tables with the query *predicate*. In addition, the following clauses are required:

- **SELECT clause**
 - Ensure that the name of the column is unique. If two tables have the same column name, use the AS keyword to create an alias name for one of them.
 - Group the columns of the same table together, and use the logical hierarchical level of the relational tables. This means group the tables according to the level of importance as they map to the hierarchical structure of your XML document. In the SELECT clause, the columns

of the higher-level tables should precede the columns of lower-level tables. The following example demonstrates the hierarchical relationship among tables:

```
SELECT o.order_key, customer, p.part_key, quantity, price, tax,  
       ship_id, date, mode
```

In this example, `order_key` and `customer` from table `ORDER_TAB` have the highest relational level because they are higher on the hierarchical tree of the XML document. The `ship_id`, `date`, and `mode` from table `SHIP_TAB` are at the lowest relational level.

- Use a single-column candidate key to begin each level. If such a key is not available in a table, the query should generate one for that table using a table expression and the user-defined function, `generate_unique()`. In the above example, the `o.order_key` is the primary key for `ORDER_TAB`, and the `part_key` is the primary key of `PART_TAB`. They appear at the beginning of their own group of columns that are to be selected. Because the `SHIP_TAB` table does not have a primary key, one needs to be generated, in this case, `ship_id`. It is listed as the first column for the `SHIP_TAB` table group. Use the `FROM` clause to generate the primary key column, as shown in the following example.

- **FROM clause**

- Use a table expression and the user-defined function, `generate_unique()`, to generate a single key for tables that do not have a primary single key. For example:

```
FROM order_tab as o, part_tab as p,  
     table(select db2xml.generate_unique() as  
           ship_id, date, mode from ship_tab) as s
```

In this example, a single column candidate key is generated with the function, `generate_unique()` and given an alias named `ship_id`.

- Use an alias name when needed to make a column distinct. For example, you could use `o` for `ORDER_TAB`, `p` for `PART_TAB`, and `s` for `SHIP_TAB`.

- **WHERE clause**

- Specify a primary and foreign key relationship as the join condition that ties tables in the collection together. For example:

```
WHERE p.price > 2500.00 AND s.date > "1996-06-01" AND  
      p.order_key = o.order_key AND s.part_key = p.part_key
```

- Specify any other search condition in the predicate. Any valid predicate can be used.

- **ORDER BY clause**

- Define the `ORDER BY` clause at the end of the `SQL_stmt`.
- Ensure that the column names match the column names in the `SELECT` clause.
- Specify the column names or identifiers that uniquely identify entities in the entity-relationship design of the database. An identifier can be generated using a table expression and the function `generate_unique`, or a user-defined function (UDF).
- Maintain the top-down order of the hierarchy of the entities. The column specified in the `ORDER BY` clause must be the first column listed for each entity. Keeping the order ensures that the XML documents to be generated do not contain incorrect duplicates.

- Do not qualify the columns in ORDER BY by any schema or table name.

Although the SQL_stmt has the preceding requirements, it is powerful because you can specify any predicate in your WHERE clause, as long as the expression in the predicate uses the columns in the tables.

Requirements when using RDB_node mapping

When using this mapping method, do not use the element SQL_stmt in the <Xcollection> element of the DAD file. Instead, use the RDB_node element in each of the top nodes for *element_node* and for each *attribute_node* and *text_node*.

- **RDB_node for the top element_node**

The *top element_node* in the DAD file represents the root element of the XML document. Specify an RDB_node for the top element_node as follows:

- Specify all tables that are associated with the XML documents. For example, the following mapping specifies three tables in the RDB_node of the element_node <Order>, which is the top element_node:

```
<element_node name="Order">
  <RDB_node>
    <table name="order_tab"/>
    <table name="part_tab"/>
    <table name="ship_tab"/>
    <condition>
      order_tab.order_key = part_tab.order_key AND
      part_tab.part_key = ship_tab.part_key
    </condition>
  </RDB_node>
```

The condition element can be empty or missing if there is only one table in the collection.

- If you are decomposing, or are enabling the XML collection specified by the DAD file, you must specify a primary key for each table. The primary key can consist of a single column or multiple columns, called a composite key. The primary key is specified by adding an attribute key to the table element of the RDB_node. When a composite key is supplied, the key attribute is specified by the names of key columns separated by a space. For example:

```
<table name="part_tab" key="part_key price"/>
```

The information specified for decomposition is ignored when composing a document.

- Use the orderBy attribute to recompose XML documents containing elements or attributes with multiple occurrence back to their original structure. This attribute allows you to specify the name of a column that will be the key used to preserve the order of the document. The orderBy attribute is part of the table element in the DAD file, and it is an optional attribute.

You must explicitly spell out the table name and the column name.

- **RDB_node for each attribute_node and text_node**

In this mapping scheme, the data resides in the attribute_node and text_node for each element_node. Therefore, the XML Extender needs to know from where in the database it needs to find the data. You need to

specify an RDB_node for each attribute_node and text_node, telling the stored procedure from which table, which column, and under which query condition to get the data. You must specify the table and column values; the condition value is optional.

- Specify the name of the table containing the column data. The table name must be included in the RDB_node of the top element_node. In this example, for text_node of element <Price>, the table is specified as PART_TAB.

```
<element_node name="Price">
  <text_node>
    <RDB_node>
      <table name="part_tab"/>
      <column name="price"/>
      <condition>
        price > 2500.00
      </condition>
    </RDB_node>
  </text_node>
</element_node>
```

- Specify the name of the column that contains the data for the element text. In the previous example, the column is specified as PRICE.
- Specify a condition if you want XML documents to be generated using the query condition. In the example above, the condition is specified as price > 2500.00. Only the data meeting the condition is in the generated XML documents. The condition must be a valid WHERE clause.
- If you are decomposing a document, or are enabling the XML collection specified by the DAD file, you must specify the column type for each attribute_node and text_node. This ensures the correct data type for each column when new tables are created during the enabling of an XML collection. Column types are specified by adding the attribute type to the column element. For example,

```
<column name="order_key" type="integer"/>
```

The information specified for decomposition is ignored when composing a document.

With the RDB_node mapping approach, you don't need to supply SQL statements. However, putting complex query conditions in the RDB_node element can be more difficult.

Decomposition table size requirements

Decomposition uses RDB_node mapping to specify how an XML document is decomposed into DB2 tables by extracting the element and attribute values into table rows. The values from each XML document are stored in one or more DB2 tables. Each table can have a maximum of 1024 rows decomposed from each document.

For example, if an XML document is decomposed into five tables, each of the five tables can have up to 1024 rows for that particular document. If the table has rows for multiple documents, it can have up to 1024 rows for each document. If the table has 20 documents, it can have 20,480 rows, 1024 for each document.

Using multiple-occurring elements (elements with location paths that can occur more than once in the XML structure) affects the number of rows inserted for each document. For example, a document that contains an element <Part> that occurs

20 times, might be decomposed as 20 rows in a table. When using multiple occurring elements, consider that a maximum of 1024 rows can be decomposed into one table from a single document.

Location path

A *location path* defines the location of an XML element or attribute within the structure of the XML document. The XML Extender uses the location path in the following situations:

- To locate the elements and attributes to be extracted when using extraction UDFs
- To specify the mapping between an XML element or attribute and a DB2 column when defining the indexing scheme in the DAD for XML columns
- For structural text search, using the Text Extender
- To override the XML collection DAD file values in a stored procedure.

Figure 11 shows an example of a location path and its relationship to the structure of the XML document.

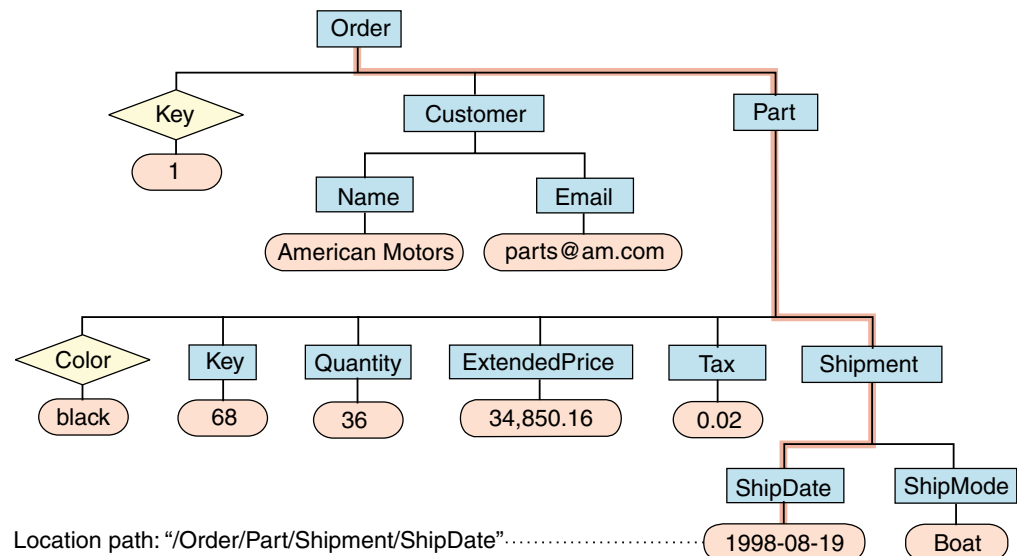


Figure 11. Storing documents as structured XML documents in a DB2 table column

Location path syntax

The following list describes the location path syntax that is supported by the XML Extender. A single slash (/) path indicates that the context is the whole document.

1. / Represents the XML *root element*, the element that contains all other elements in the document.
2. /tag1 Represents the element *tag1* under root.
3. /tag1/tag2/.../tagn Represents an element with the name *tagn* as the child of the descending chain from root, *tag1*, *tag2*, through *tagn-1*.
4. //tagn Represents any element with the name *tagn*, where double slashes (//) denote zero or more arbitrary tags.

5. */tag1//tagn*

Represents any element with the name *tagn*, a child of an element with the name *tag1* under root, where double slashes (//) denote zero or more arbitrary tags.

6. */tag1/tag2/@attr1*

Represents the attribute *attr1* of an element with the name *tag2*, which is a child of element *tag1* under root.

7. */tag1/tag2[@attr1="5"]*

Represents an element with the name *tag2* whose attribute *attr1* has the value 5. *tag2* is a child of element with the name *tag1* under root.

8. */tag1/tag2[@attr1="5"]/.../tagn*

Represents an element with the name *tagn*, which is a child of the descending chain from root, *tag1*, *tag2*, through *tagn-1*, where the attribute *attr1* of *tag2* has the value 5.

Wildcards: You can substitute an asterisk for an element in a location path to match any string. For example

/tag1//tagn/tagn+1 ?*

Simple location path

Simple location path is the location path syntax used to specify elements and attributes for side tables, defined in the XML column DAD file. Simple location path is represented as a sequence of element type names that are connected by a single slash (/). The attribute values are enclosed within square brackets following its element type. Table 10 summarizes the syntax for simple location path.

Table 10. Simple location path syntax

Subject	Location path	Description
XML element	<i>/tag1/tag2/.../tagn-1/tagn</i>	An element content identified by the element named <i>tagn</i> and its parents
XML attribute	<i>/tag_1/tag_2/.../tag_n-1/tag_n/@attr1</i>	An attribute with name <i>attr1</i> of the element identified by <i>tagn</i> and its parents

Location path usage

The syntax of the location path depends in which context you use it to access the location of an element or attribute. Because the XML Extender uses one-to-one mapping between an element or attribute, and a DB2 column, it restricts the syntax rules that are allowed in the DAD file and in functions. Table 11 describes in which context the syntax options are used. The numbers that are specified in the "Location path supported" column refer to the syntax representations in "Location path syntax" on page 61.

Table 11. The XML Extender's restrictions using location path

Use of the location path	Location path supported
Element in the XML column DAD mapping for side tables	3, 6 (simple location path described in Table 10)
Extracting UDFs	1-8 ¹
Update UDF	1-8 ¹

Table 11. The XML Extender's restrictions using location path (continued)

Use of the location path	Location path supported
Text Extender's search UDF	3 – Exception: the root mark is specified without the slash. For example: tag1/tag2/.../tagn

¹ The extracting and update UDFs support location paths that have predicates with attributes, but not elements.

Chapter 4. Using the administration tools

To complete administration tasks, you can use one or more of the following tools:

- The XML Extender administration wizard for all administration tasks
- The DXXADM command from TSO, or USS with the odb2 command line
- SQL statements run from the odb2 command line
- JCL based on samples provided in the SDXXJCL dataset, as listed in Table 6 on page 37
- Custom applications using the XML Extender administration stored procedures

The following chapters show how to complete administration tasks using the following methods:

- XML Extender administration wizard. See “Starting the administration wizard” to learn how to set up and use the wizard.
- The DXXADM command from TSO, or USS with the odb2 command line
- DB2 command line, called the “command line”. If you are using UNIX System Services (USS), you can use the odb2 command line tool. odb2 is a command line for DB2 on OS/390 and z/OS UNIX. See “Using the USS odb2 command line” on page 67 to learn how to use the odb2 command line.

The administration stored procedures are described in “Administration stored procedures” on page 191.

Starting the administration wizard

The following sections describe how to set up and invoke the XML Extender administration wizard.

Setting up the administration wizard

Ensure that you have followed the installation and configuration steps for the administration wizard in the readme file for your operating system. This includes ensuring that you have run the bind statement and included the required software in your CLASSPATH statements.

- The bind statements are provided in the wizard readme files. and in the getting started sample file:

```
/dxx_install/samples/cmd/getstart_prep.cmd
```

- The CLASSPATH statement should look something like the following example (line breaks are for presentation only):

```
.;C:\java\db2java.zip;C:\java\runtime.zip;C:\java\sqlj.zip;  
C:\dxx\dxxadmin\dxxadmin.jar;C:\dxx\dxxadmin\dxxadmin.cmd;  
C:\dxx\dxxadmin\html\dxxahelp*.htm;C:\java\jdk\lib\classes.zip;  
C:\java\swingall.jar
```

Important: The wizard requires a path name without a space. If you have the IBM DB2 Universal Database V7.1 default installation, SQLLIB\java is under the Program Files directory, copy the Java code to a simpler path. Do not move the Java code and change CLASSPATH; the Control Center requires the CLASSPATH specified during installation.

The XML Extender Administration wizard uses a class file. The complete file name of the main XML Extender Administration class file is:

```
com.ibm.dxx.admin.Admin.
```

Modify this file for your system to invoke the wizard.

- To invoke using the JDK, type:

```
java -classpath classpath com.ibm.dxx.admin.Admin
```
- To invoke using the JRE, type:

```
jre -classpath classpath com.ibm.dxx.admin.Admin
```

where *classpath* specifies either:

- The %CLASSPATH% environment variable to specify where the administration wizard class files are located. When using this option, your system CLASSPATH must point to the *dxx_install/dxxadmin* directory, which contains the following files: *dxxadmin.jar*, *xml4j.jar*, and *db2java.zip*. For example:

```
java -classpath %CLASSPATH% com.ibm.dxx.admin.Admin
```

- An override of the %CLASSPATH% environment variable with pointers to files in the *dxx_install/dxxadmin* directory, from which you are running the XML Extender administration wizard. For example:

```
java -classpath dxxadmin.jar;xml4j.jar;db2java.zip com.ibm.dxx.admin.Admin  
url=jdbc:db2://mydb2 userid=db2xml password=db2xml  
driver=COM.ibm.db2.jdbc.app.DB2Driver
```

Optionally, you can specify the following parameters at runtime:

- url** Fully-qualified URL path to the IBM DB2 UDB data source to connect to. For example: `jdbc:db2://dxx.stl.ibm.com:8080/guidb`. Labeled “Address” in the wizard.
- userid** Userid to use to access the above data source. For example: `db2guest`.
- password**
Password for the above user ID. For example: `guest`.
- driver** JDBC driver name for the above URL. Default: `COM.ibm.db2.jdbc.net.DB2Driver`. Labeled “JDBC driver” in the wizard.

See “Invoking the administration wizard” for more information about these values.

Important: You must have DB2 Connect Personal or Enterprise Edition to use the Wizard with XML Extender for OS/390 and z/OS.

Invoking the administration wizard

Follow these steps to invoke the XML Extender administration wizard.

1. Invoke the wizard.

For Windows NT:

Double click on the XML Extender administration wizard icon from the desktop.

For UNIX:

Run the *dxxadmin* file.

The administration wizard Logon window opens.

When you invoke the XML Extender administration wizard, the Logon window opens. Log in to the database that you want to use when working with XML data. XML Extender connects to the current database.

2. In the **Address** field, enter the fully-qualified JDBC URL to the IBM DB2 UDB data source to which you are connecting. The address has the following syntax:

```
jdbc:db2:database_name
```

Where *database_name* is the database to which you are connecting and storing XML documents.

For example:

```
jdbc:db2:sales_db
```

3. In the **User ID** and **Password** fields, enter or verify the DB2 user ID and password for the database to which you are connecting.
4. In the **JDBC Driver** field, verify the JDBC driver name for the specified address using the following values:
COM.ibm.db2.jdbc.app.DB2DRIVER
5. Click **Finish** to connect to the wizard and advance to the LaunchPad window.

The LaunchPad window provides access to five administration wizards. With these wizards, you can:

- Enable server
- Add a DTD to the DTD repository
- Work with DAD files for:
 - XML columns
 - XML collections
- Work with XML columns
- Work with XML collections

Using the USS odb2 command line

You use the odb2 command line to enter DB2 commands from USS. The odb2 command line uses dynamic SQL and the Call Attach Facility (CAF) to allow the execution SQL commands from the OS/390 and z/OS UNIX shell against an OS/390 and z/OS DB2 database.

See “Software requirements” on page 37 to learn how to download and install the odb2 command line.

It is recommended that you create a symbolic link from DB2 to the odb2 command line in a directory that is in your PATH environment setting in order to have scripts provided for the examples in this book run seamlessly.

To start the odb2 command line:

From USS command shell, type:

```
odb2
```

A command prompt is displayed, from which you can enter DB2 commands.

Chapter 5. Managing the database server

The XML Extender administration tasks consist of enabling your database and table columns for XML and mapping XML data to DB2 relational structures. This chapter describes administration tasks for managing the database server:

1. “Enabling a database server for XML”
2. “Storing a DTD in the DTD repository table” on page 70
3. “Disabling a server for XML” on page 72

To complete the tasks in this chapter, you should be familiar with the concepts and planning tasks that are described in “Administration planning” on page 46.

In addition to choosing tools and setting up the database environment, you must define XML columns or XML collections. These tasks are described in the following chapters:

- “Chapter 6. Working with XML columns” on page 73
- “Chapter 7. Working with XML collections” on page 83

Enabling a database server for XML

To store or retrieve XML documents from DB2 with XML Extender, you enable the database for XML. XML Extender enables the database you are connected to.

When you enable a database for XML, the XML Extender:

- Creates all the user-defined types (UDTs), user-defined functions (UDFs), and stored procedures
- Creates and populates control tables with the necessary metadata that the XML Extender requires
- Creates the DB2XML schema and assigns the necessary privileges

The full name of an XML function is *schema-name.function-name*, where *schema-name* is an identifier that provides a logical grouping for SQL objects. You can use the full name anywhere you refer to a UDF or a UDT. You can also omit the schema name when you refer to a UDF or a UDT; in this case, DB2 uses the function path to determine the function or data type that you want.

Using the administration wizard

Use the following steps to enable a database for XML data:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Enable Server** from the LaunchPad window to enable the current database.

If a database is already enabled, only **Disable Server** is selectable.

When the database is enabled, you are returned to the LaunchPad window.

Using the command line

Enter DXXADM from the command line, specifying the database that is to be enabled.

Syntax:

```
► dxxadm enable_server -a subsystem_name [security security_level] ►
► using tablespace_DTD_REF, tablespace_XML_USAGE ►
► WLM environment WLM_name1 [WLM_name2] ►
```

Parameters:

-a *subsystem_name*

The name of the database server that is to be enabled.

security_level

Determines the user ID that is authorized to access external resources when running stored procedures. Choices are DB2, USER, DEFINER. DB2 is the default. See *DB2 Universal Database for OS/390 and z/OS SQL Reference, Version 7* for more information

tablespace_DTD_REF

The name of the table space in which the CLOB column, CONTENT, of the DTD_REF table, is stored.

tablespace_XML_USAGE

The name of table space in which the CLOB column, DAD, of the XML_USAGE table, is stored.

WLM_name

The names of the WLM environments. At least one name is required. If one is specified, the name is for all stored procedures and UDFs. If two are specified, the first name is for the stored procedures, the second name is for the UDFs.

Example: Enables an existing database server, called SALES_DB.

```
dxxadm enable_server -a SUBSYS1 using tbspc1,tbspc2 wlm environment envir233
```

Storing a DTD in the DTD repository table

You can use a DTD to validate XML data in an XML column or in an XML collection. All DTDs are stored in the DTD repository table, a DB2 table called DTD_REF. It has a schema name of DB2XML. Each DTD in the DTD_REF table has a unique ID. The XML Extender creates the DTD_REF table when you enable a database for XML.

See “Planning for XML columns” on page 48 and “Planning for XML collections” on page 52 to learn more about using DTDs.

You can insert the DTD from the command line or by using the administration wizard.

Using the administration wizard

Use the following steps to insert a DTD:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Import a DTD** from the LaunchPad window to import an existing DTD file into the DTD repository of the current database. The Import a DTD window opens.
3. Type the DTD file name in the **DTD file name** field or click ... to browse for an existing DTD file.
4. Type the DTD ID in the **DTD ID** field.
The DTD ID is an identifier for the DTD and can be the path specifying the location of the DTD on the local system. The DTD ID must match the value that is specified in the DAD file for the <DTDID> element.
5. Optionally, type the name of the author of the DTD in the **Author** field.
The XML Extender automatically displays the author’s name if it is specified in the DTD.
6. Click **Finish** to insert the DTD into the DTD repository table, DB2XML.DTD_REF and return to the LaunchPad window.

From the command line

Issue an SQL INSERT statement for the DTD_REF table using the schema in Table 12:

Table 12. The column definitions for the DTD Reference table

Column name	Data type	Description
DTDID	VARCHAR(128)	Primary key (unique and not NULL). The primary key is used to identify the DTD and must be the same as the SYSTEM ID on the DOCTYPE line in each XML document, when validation is used. When the primary key is specified in the DAD file, the DAD file must follow the schema that is defined by the DTD.
CONTENT	XMLCLOB	The content of the DTD.
USAGE_COUNT	INTEGER	The number of XML columns and XML collections in the database that use this DTD to define a DAD.
AUTHOR	VARCHAR(128)	Author of the DTD, optional information for user to input.
CREATOR	VARCHAR(128)	The user ID that does the first insertion.
UPDATOR	VARCHAR(128)	The user ID that does the last update.

For example:

```
DB2 INSERT into DB2XML.DTD_REF values('dxx_install/samples/dtd/getstart.dtd',
DB2XML.XMLClobFromFile('dxx_install/samples/dtd/getstart.dtd'), 0, 'user1',
'user1', 'user1')
```

Important for XML collections: The DTD ID is a path specifying the location of the DTD on the local system. The DTD ID must match the value that is specified in the DAD file for the <DTDID> element.

Disabling a server for XML

You disable the server when you want to clean up your XML Extender environment and drop the XML Extender UDTs, UDFs, stored procedures, and administration support tables. XML Extender disables the server to which you are connected.

When you disable a server for XML, the XML Extender takes the following actions:

- Deletes all the user-defined types (UDTs), user-defined functions (UDFs), and stored procedures
- Deletes control tables with the metadata for the XML Extender
- Deletes the DB2XML schema.

Before you begin

Disable any XML columns or collections.

Using the administration wizard

Use the following steps to disable a server for XML data:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Disable server** from the LaunchPad window to disable the current database.

If a database is not current enabled, only **Enable server** is selectable.

When the server is disabled, you are returned to the LaunchPad window.

Using the command line

Enter DXXADM from the command line, specifying the database that is to be disabled.

Syntax:

```
►►—disable_server—-a—subsystem_name—◄◄
```

Parameters:

-a *subsystem_name*

The name of the DB2 subsystem that is to be disabled.

Example: disables the server.

```
dxxadm disable_server -a SUBSYS1
```

Chapter 6. Working with XML columns

An XML column contains XML documents that can be updated, searched, and extracted, and is created as an XML user data type (such as XMLVARCHAR). To store XML documents in an XML column, you need to complete the following tasks:

- Create a document access definition (DAD) file. See “Creating or editing the DAD file”.
- Create or alter the table in which the XML documents are stored. See “Creating or altering an XML table” on page 76.
- Enable a column for XML data. See “Enabling XML columns” on page 77.
- Index side tables. See “Indexing side tables” on page 80.

To drop the table that contains the XML column, disable the XML column. See “Disabling XML columns” on page 81.

Creating or editing the DAD file

To set up XML columns, you need to define the DAD file to access your XML data and to enable columns for XML data in an XML table. An important concept in creating the DAD is understanding location path syntax because it is used to map the element and attribute values that you want to index to DB2 tables. See “Location path” on page 61 to learn more about location path and its syntax.

When you specify a DAD file, you define the attributes and key elements of your data that need to be searched. The XML Extender uses this information to create side tables so that you can index your data to retrieve it quickly. See “The DAD file” on page 52 to learn about planning issues for creating the DAD file.

Before you begin

- Understand the hierarchical structure of your XML data so that you can define key elements and attributes for indexing and fast search.
- Prepare and insert the XML document’s DTD into the DTD_REF table. This step is required for validation.

Using the administration wizard

Use the following steps to create a DAD file:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with DAD files** from the LaunchPad window to edit or create an XML DAD file. The Specify a DAD file window opens.

3. Choose whether to edit an existing DAD file or to create a new DAD file.
 - **To edit an existing DAD:**
 - a. Click ... to browse for an existing DAD file in the pull-down menu, or type the DAD file name in the **File name** field.
 - b. Verify that the wizard recognizes the specified DAD file.
 - If the wizard recognizes the specified DAD file, **Next** is selectable, and XML column is displayed in the **Type** field.
 - If the wizard does not recognize the specified DAD file, **Next** is not selectable. Either retype the DAD file name in the **File name** field, or click **Open** to browse again for an existing DAD file. Continue until **Next** is selectable.
 - c. Click **Next**.
 - **To create a new DAD:**
 - a. Leave the **File name** field blank.
 - b. From the **Type** menu, click **XML column**.
 - c. Click **Next**.

4. Choose whether to validate your XML documents with a DTD from the Select Validation window.
 - To validate:
 - a. Click **Validate XML documents with the DTD**.
 - b. Select the DTD to be used for validation from the **DTD ID** menu.

If XML Extender does not find the specified DTD in the DTD reference table, it searches for the specified DTD on the file system and uses it to validate.

- Click **Do NOT validate XML documents with the DTD** to continue without validating your XML documents.
5. Click **Next**.
 6. Choose whether to add a new side table, edit an existing side table, or remove an existing side table from the Side tables window.

- **To add a new side table or side-table column:**

To add a new side table, you define the columns in the table. Complete the following steps for each column in a side table.

- a. Complete the fields of the **Details** box of the Side tables window.
 - 1) **Table name:** Type the name of the table containing the column. For example:
ORDER_SIDE_TAB
 - 2) **Column name:** Type the name of the column. For example:
CUSTOMER_NAME
 - 3) **Type:** Select the type of the column from the menu. For example:
XMLVARCHAR
 - 4) **Length (VARCHAR type only):** Type the maximum number of VARCHAR characters. For example:
30
 - 5) **Path:** Type the location path of the element or attribute. For example:
/ORDER/CUSTOMER/NAME

See “Location path” on page 61 for location path syntax.

- 6) **Multi occur:** Select **No** or **Yes** from the menu.
Indicates whether the location path of this element or attribute can be used more than once in a document.
Important If you specify multiple occurrence for a column, you can specify only one column in the side table.
 - b. Click **Add** to add a column.
 - c. Continue adding, editing, or removing columns for the side table, or click **Next**.
- **To edit an existing side-table column:**
You can update a side table by changing the definitions of the existing columns.
 - a. Click on the side-table name and column name you want to edit.
 - b. Edit the fields of the **Details** box.
 - c. Click **Change** to save changes.
 - d. Continue adding, editing, or removing columns for each side table, or click **Next**.
- **To remove an existing side-table column:**
 - a. Click on the side table and column you want to remove.
 - b. Click **Remove**.
 - c. Continue adding, editing, or removing side-tables columns, or click **Next**.
- **To remove an existing side table:**
To remove an entire side table, you delete each column in the table.
 - a. Click on each side-table column for the table you want to remove.
 - b. Click **Remove**.
 - c. Continue adding, editing, or removing side tables columns, or click **Next**.
7. Type an output file name for the modified DAD file in the **File name** field of the Specify a DAD window.
8. Click **Finish** to save the DAD file and to return to the LaunchPad window.

Using the command line

The DAD file is an XML file that can be created in any text editor.

Use the following steps to create a DAD file:

1. Open a text editor.
2. Create the DAD file header, using the following syntax:


```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path/dtd/dad.dtd"> --> the path and file name of
the DTD for the DAD file
```
3. Insert the <DAD></DAD> tags.
4. Inside the <DAD> tag, optionally specify the DTD ID identifier that associates the DAD file with the XML document DTD for validation:


```
<dtdid>path/dtd_name.dtd</dtdid> --> the path and file
name of the DTD
for your application
```

The DTD ID is required for validation and must match the DTD ID value used when inserting the DTD into the DTD reference table (DB2XML.DTD_REF).

5. Specify whether to validate (that is, to use the specified DTD to ensure that the XML document is a valid XML document). For example:

```
<validation>YES</validation> --> specify YES or NO
```

If you specify YES, you must have specified a DTD ID in the previous step as well as inserted a DTD into the DTD_REF table.

6. Use the <Xcolumn> element to define the access and storage method as XML column.

```
<Xcolumn>  
</Xcolumn>
```

7. Define each side table and the important elements and attributes to be indexed for structural search. Perform the following steps for each table. The following steps use examples taken from a sample DAD file shown in "DAD file: XML column" on page 244:

- a. Insert the <table></table> tags and the name attribute.

```
<table name="order_tab">  
</table>
```

- b. After the <table> tag, insert a <column> tag and its attributes for each column in the table:

- **name**: the name of the column
- **type**: the type of column
- **path**: the location path of the element or attribute. See "Location path" on page 61 for location path syntax.
- **multi_occurrence**: an indication of whether this element or attribute can appear more than once in a document. Note that for a location path target that is an attribute, a value of "YES" indicates that the attribute appears more than once because the element to which the attribute is attached appears more than once.

```
<table ...>  
  <column name="order_key"  
    type="integer"  
    path="/Order/@key"  
    multi_occurrence="NO"/>  
  <column name="customer"  
    type="varchar(50)"  
    path="/Order/Customer/Name"  
    multi_occurrence="NO"/>  
</table>
```

8. Ensure that you have an ending </table> tag after the last column definition.
9. Ensure that you have an ending </Xcolumn> tag after the last </table> tag.
10. Ensure that you have an ending </dad> tag after the </Xcolumn> tag.

Creating or altering an XML table

To store intact XML documents in a table, you must create or alter a table so that it contains a column with an XML user-defined type (UDT). The table is known as an *XML table*, a table that contains XML documents. The table can be an altered table or a new table. When a table contains a column of XML type, you can enable the column for XML.

You can alter an existing table to add a column of XML type using the administration wizard, or using the command line.

Using the administration wizard

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with XML columns** from the LaunchPad window. The Select a task window opens.
3. Click **Add an XML Column**. The Add an XML column window opens.
4. Select the name of the table from the **Table name** pull-down menu, or type the name of the table you want to alter. For example:
SALES_TAB
5. Type the name of the column to be added to the table in the **Column name** field. For example:
ORDER
6. Select the UDT for the column from the **Column type** pull-down menu. For example:
XMLVARCHAR
7. Click **Finish** to add the column of XML type.

Using the command line

Create or alter a table with a column of an XML type in the column clause of the CREATE TABLE or ALTER TABLE statement.

Example: In the sales application, you might want to store an XML-formatted line item order in a column called ORDER of an application table called SALES_TAB. This table also has the columns INVOICE_NUM and SALES_PERSON. Because it is a small order, you store it using the XMLVARCHAR type. The primary key is INVOICE_NUM. The following CREATE TABLE statement creates the table with a column of XML type:

```
CREATE TABLE sales_tab(  
    invoice_num char(6) NOT NULL PRIMARY KEY,  
    sales_person varchar(20),  
    order XMLVarchar);
```

Enabling XML columns

To store an XML document in a DB2 database, you must enable a column for XML. Enabling a column prepares it for indexing so that it can be searched quickly. You can enable a column by using the XML Extender administration wizard or using the command line. The column must be of XML type.

When the XML Extender enables an XML column, it:

- Reads the DAD file to optionally:
 - Validate the DAD file against the DTD for the DAD file.
 - Retrieve the DTD ID from the DTD_REF table, if specified.
 - Create side tables for indexing on the XML column.
 - Prepare the column to contain XML data.
- Optionally creates a *default view* of the XML table and side tables. The default view displays the application table and the side tables.
- Specifies a ROOT ID value, if one has not been specified.

After you enable the XML column, you can

- Create indexes on the side tables

- Insert XML documents in the XML column
- Query, update, or search the XML documents in the XML column.

Before you begin

- Create an XML table by creating or altering a DB2 table with a column of XML type.
- Create a DAD file specifying both the column to be enabled and the side tables to be created for indexing frequently searched elements and attributes.

Using the administration wizard

Use the following steps to enable XML columns:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with XML Columns** from the LaunchPad window to view the XML Extender column related tasks. The Select a Task window opens.
3. Click **Enable a Column** and then **Next** to enable an existing table column in the database.
4. Select the table that contains the XML column from the **Table name** field. For example:
SALES_TAB
5. Select the column being enabled from the **Column name** field. For example:
ORDER

The column must exist and be of XML type.

6. Type the DAD path and file name in the **DAD file name** field, or click ... to browse for an existing DAD file. For example:
dxx_install/samples/dad/getstart.dad
7. Optionally, type the name of an existing table space in the **Table space** field. The table space contains side tables that the XML Extender created. If you specify a table space, the side tables are created in the specified table space. If you do not specify a table space, the side tables are created in the default table space.
8. Optionally, type the name of the default view in the **Default view** field. When specified, the default view is automatically created when the column is enabled and joins the XML table and all of the related side tables.
9. Optionally, type the column name of the primary key in the application table in the **Root ID** field. This is recommended. The XML Extender uses the value of ROOT ID as a unique identifier to associate all side tables with the application table. If not specified, the XML Extender adds the DXXROOT_ID column to the application table and generates an identifier.
10. Click **Finish** to enable the XML column, create the side tables, and return to the LaunchPad window.
 - If the column is successfully enabled, an Enabled column is successful message is displayed.
 - If the column is not successfully enabled, an error box is displayed. Correct the values of the entry field until the column is successfully enabled.

Using the command line

To enable an XML column, enter the following command:

Syntax:

```
►► dxxadm enable_column -a subsystem_name tbName colName DAD_file ►►  
  
└─t─tablespace┘ └─v─default_view┘ └─r─root_id┘
```

Parameters:

-a *subsystem_name*

The name of the DB2 subsystem.

tbName

The name of the table that contains the column that is to be enabled.

colName

The name of the XML column that is being enabled.

DAD_file

The name of the file that contains the document access definition (DAD).

tablespace

A previously created table space that contains side tables that the XML Extender created. If not specified, the default table space is used.

default_view

Optional. The name of the default view that the XML Extender created to join an application table and all of the related side tables.

root_id

Optional, but recommended. The column name of the primary key in the application table and a unique identifier that associates all side tables with the application table. Known as ROOT_ID. The XML Extender uses the value of ROOT_ID as a unique identifier to associate all side tables with the application table. If the ROOT ID is not specified, the XML Extender adds the DXXROOT_ID column to the application table and generates an identifier.

Restriction: If the application table has a column name of DXXROOT_ID, but this column does not contain the value for *root_id*, you must specify the *root_id* parameter; otherwise, an error occurs.

Example: The following example enables a column using the command line. The DAD file and XML document can be found in “Appendix B. Samples” on page 243.

```
dxxadm enable_column -a SUBSYS1 SALES_TAB ORDER -v SALODVW -r INVOICE_NUMBER
```

In this example, the column ORDER is enabled in the table SALES_TAB. The DAD file is getstart.dad, the default view is sales_order_view, and the ROOT ID is INVOICE_NUM.

Using this example, the SALES_TAB table has the following columns:

Column name	INVOICE_NUM	SALES_PERSON	ORDER
-------------	-------------	--------------	-------

Data type	CHAR(6)	VARCHAR(20)	XMLVARCHAR
------------------	---------	-------------	------------

The following side tables are created based on the DAD specification:

ORDER_SIDE_TAB:

Column name	ORDER_KEY	CUSTOMER	INVOICE_NUM
Data type	INTEGER	VARCHAR(50)	CHAR(6)
Path expression	/Order/@key	/Order/Customer/Name	N/A

PART_SIDE_TAB:

Column name	PART_KEY	PRICE	INVOICE_NUM
Data type	INTEGER	DOUBLE	CHAR(6)
Path expression	/Order/Part/@key	/Order/Part/ExtendedPrice	N/A

SHIP_SIDE_TAB:

Column name	DATE	INVOICE_NUM
Data type	DATE	CHAR(6)
Path expression	/Order/Part/Shipment/ShipDate	N/A

All the side tables have the column INVOICE_NUM of the same type, because the ROOT ID is specified by the primary key INVOICE_NUM in the application table. After the column is enabled, the value of the INVOICE_NUM is inserted into the side tables. Specifying the *default_view* parameter when enabling the XML column, ORDER, creates a default view, sales_order_view. The view joins the above tables using the following statement:

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
                           order_key, customer, part_key, price, date)
AS
SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
       order_tab.order_key, order_tab.customer,
       part_tab.part_key, part_tab.price,
       ship_tab.date
FROM sales_tab, order_tab, part_tab, ship_tab
WHERE sales_tab.invoice_num = order_tab.invoice_num
      AND sales_tab.invoice_num = part_tab.invoice_num
      AND sales_tab.invoice_num = ship_tab.invoice_num
```

If the table space is specified in the ENABLE_COLUMN command, the side tables are created in the specified table space. If the table space is not specified, the side tables are created in the default table space.

Indexing side tables

After you have enabled an XML column and created the side tables, you can index the side tables. Side tables contain the XML data in columns you specified while creating the DAD file. Indexing these tables helps you improve the performance of the queries against the XML documents.

Before you begin

- Create a DAD file that specifies side tables for the XML document structure.
- Enable the XML column using the DAD file; which creates the side tables.

Creating the indexes

From the command line, use the DB2 CREATE INDEX command to index the side tables.

Example:

The following example creates indexes on four side tables:

```
DB2 CREATE INDEX KEY_IDX
      ON ORDER_SIDE_TAB(ORDER_KEY)

DB2 CREATE INDEX CUSTOMER_IDX
      ON ORDER_SIDE_TAB(CUSTOMER)

DB2 CREATE INDEX PRICE_IDX
      ON PART_SIDE_TAB(PRICE)

DB2 CREATE INDEX DATE_IDX
      ON SHIP_SIDE_TAB(DATE)
```

Disabling XML columns

Disable a column if you need to update a DAD file for the XML column, or if you want to delete the XML column or the table that contains the column. After the column is disabled, you can re-enable the column with the updated DAD file, delete the column, or other tasks. You can disable a column by using the XML Extender administration wizard or using the command line.

When the XML Extender disables an XML column, it:

- Deletes the column's entry from XML_USAGE table.
- Drops the side tables associated with this column.

Important: If you drop a table with an XML column, without first disabling the column, XML Extender cannot drop any side tables associated with the XML column, which might cause unexpected results.

Before you begin

Ensure that the XML column to be disabled exists in the current DB2 database.

Using the administration wizard

Use the following steps to disable XML columns:

1. Set up and start the administration wizard. See "Starting the administration wizard" on page 65 for details.
2. Click **Working with XML Columns** from the LaunchPad window to view the XML Extender column related tasks. The Select a Task window opens.
3. Click **Disable a Column** and then **Next** to disable an existing table column in the database.
4. Select the table that contains the XML column from the **Table name** field.
5. Select the column being disabled from the **Column name** field.
6. Click **Finish**.

- If the column is successfully disabled, an Disabled column is successful message is displayed.
- If the column is not successfully disabled, an error box is displayed. Correct the values of the entry field until the column is successfully disabled.

Using the command line

To disable an XML column, enter the following command:

Syntax:

```
►► dxxadm disable_column -a subsystem_name tbName colName ◀◀
```

Parameters:

-a *subsystem_name*

The name of the DB2 subsystem.

tbName

The name of the table that contains the column that is to be disabled.

colName

The name of the XML column that is being disabled.

Example: The following example disables a column using the command line. The DAD file and XML document can be found in “Appendix B. Samples” on page 243.

```
dxxadm disable_column -a SUBSYS1 SALES_TAB ORDER
```

In this example, the column ORDER is disabled in the table SALES_TAB.

When the column is disabled, the side tables are dropped.

Chapter 7. Working with XML collections

XML collections are collections of tables that associated by a common XML document structure. The tables are either associated because they contain data that you will use to populate XML documents, or columns in which you will store data decomposed from XML documents.

Setting up XML collections requires creating a mapping scheme and optionally enabling the collection with a name that associates the DB2 tables with a DAD file. Although enabling the XML collection is not required, it does provide better performance.

To set up an XML collection, you must complete the following tasks:

- Create a document access definition (DAD) file. See “Creating or editing the DAD file for the mapping scheme”
- (Optional) Enable the collection. See “Enabling XML collections” on page 101.

To redefine or delete the collection, disable the XML collection. See “Disabling XML collections” on page 103.

Creating or editing the DAD file for the mapping scheme

Creating a DAD file is required when using XML collections. A DAD file defines the relationship between XML data and multiple relational tables. The XML Extender uses the DAD file to:

- Compose an XML document from relational data
- Decompose an XML document to relational data

You can use either of two methods to map the data between the XML tables and the DB2 table: SQL mapping and RDB_node mapping:

SQL mapping

Uses an SQL statement element to specify the SQL query for tables and columns that are used to contain the XML data. SQL mapping can only be used for composing XML documents.

RDB_node mapping

Uses an XML Extender-unique element, Relational Database node, or RDB_node, which specifies tables, columns, conditions, and the order for XML data. RDB_node mapping supports more complex mappings than an SQL statement can provide. RDB_node mapping can be used for both composing and decomposing XML documents.

The following sections describe how to create the DAD file, depending on the task and the method you are using:

- Compose documents with SQL mapping. See “Composing XML documents with SQL mapping” on page 84.
- Compose documents with RDB node mapping. See “Composing XML documents with RDB_node mapping” on page 89.

When creating DAD files for composition, you can specify stylesheets. See “Specifying a stylesheet for the XML document” on page 95.

- Decompose documents with RDB node mapping. See “Decomposing XML documents with RDB_node mapping” on page 95.

Before you begin

- Map the relationship between your DB2 tables and the XML document. This step should include mapping the hierarchy of the XML document and specifying how the data in the document maps to a DB2 table.
- If you plan to validate the XML documents, insert the DTD for the XML document you are composing or decomposing into the DTD reference table, DB2XML.DTD_REF.

Composing XML documents with SQL mapping

Use SQL mapping when you are composing XML documents and want to use SQL.

Using the administration wizard

Use the following steps to create a DAD file using XML collection SQL mapping

To create a DAD file for composition using SQL mapping:

Use SQL mapping when you are composing XML documents and you want to use an SQL statement to define the table and columns from which you will derive the data in the XML document.

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with DAD files** from the LaunchPad window. The Specify a DAD windows opens.
3. Choose whether to edit an existing DAD file or to create a new DAD file.

To create a new DAD file:

- a. Leave the **File name** field blank.
- b. From the **Type** menu, select **XML collection SQL mapping**.
- c. Click **Next** to open the Select Validation window.

To edit an existing DAD file:

- a. Type the DAD file name in the **File name** field, or click ... to browse for an existing DAD file.
 - b. Verify that the wizard recognizes the specified DAD file.
 - If the wizard recognizes the specified DAD file, **Next** is selectable and XML collection SQL mapping is displayed in the **Type** field.
 - If the wizard does not recognize the specified DAD file, **Next** is not selectable. Either retype the DAD file name, or click ... to browse again for an existing DAD file. Correct the values of the entry field until **Next** is selectable.
 - c. Click **Next** to open the Select Validation window.
4. In the Select Validation window, choose whether to validate your XML documents with a DTD.
 - To validate:
 - a. Click **Validate XML documents with the DTD**.
 - b. Select the DTD to be used for validation from the **DTD ID** menu.

If XML Extender does not find the specified DTD in the DTD reference table, it searches for the specified DTD on the file system and uses it to validate.

- Click **Do NOT validate XML documents with the DTD** to continue without validating your XML documents.

5. Click **Next** to open the Specify Text window.
6. Type the prolog name in the **Prolog** field, to specify the prolog of the XML document to be composed.

```
<?xml version="1.0" ?>
```

If you are editing an existing DAD, the prolog is automatically displayed in the **Prolog** field.

7. Type the document type of the XML document in the **Doctype** field of the Specify Text window, pointing to the DTD for the XML document. For example:


```
! DOCTYPE ORDER SYSTEM "dxx_install/samples/dtd/getstart.dtd"
```

If you are editing an existing DAD, the document type is automatically displayed in the **Doctype** field.

8. Click **Next** to open the Specify SQL Statement window.
9. Type a valid SQL SELECT statement in the **SQL statement** field. For example:

```
SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
       price, tax, ship_id, date, mode from order_tab o, part_tab p,
table (select db2xml.generate_unique()
       as ship_id, date, mode, part_key from ship_tab) s
WHERE o.order_key = 1 and
      p.price > 20000 and
      p.order_key = o.order_key and
      s.part_key = p.part_key
ORDER BY order_key, part_key, ship_id
```

If you are editing an existing DAD, the SQL statement is automatically displayed in the **SQL statement** field.

10. Click **Test SQL** to test the validity of the SQL statement.
 - If your SQL statement is valid, sample results are displayed in the **Sample results** field.
 - If your SQL statement is not valid, an error message is displayed in the **Sample results** field. The error message instructs you to correct your SQL SELECT statement and to try again.
11. Click **Next** to open the SQL Mapping window.
12. Select an element or attribute node to map from by clicking on it in the field on the left of the SQL Mapping window.

Map the elements and attributes in the XML document to element and attribute nodes that correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.

 - **To add the root node:**
 - a. Select the **Root** icon.
 - b. Click **New Element** to define a new node.
 - c. In the **Details** box, specify **Node type** as **Element**.
 - d. Enter the name of the top level node in the **Node name** field.
 - e. Click **Add** to create the new node.

You have created the root node or element, which is the parent to all the other element and attribute nodes in the map. You can now add child elements and attributes to this node.
 - **To add a child element or attribute node:**
 - a. Click on a parent node in the field on the left to add a child element or attribute.

If you have not selected a parent node, **New Element** is not selectable.

- b. Click **New Element**.
- c. Select the node type from the **Node type** menu in the **Details** box.

The **Node type** menu displays only the node types that are valid at that point in the map:

Element

Represents an XML element defined in the DTD associated with the XML document. Used to associate the XML element with a column in a DB2 table. An element node can have attribute nodes, child element nodes, or text nodes. A bottom-level node has a text node and column name associated with it in the tree view.

Attribute

Represents an XML attribute defined in the DTD associated with the XML document. It is used to associate the XML attribute with a column in a DB2 table. An attribute node can have a text node and has a column name associated with it in the tree view.

Text Specifies text content for an element or attribute node that has content to be mapped to a relational table. A text node has a column name associated with it in the tree view.

Table Specifies the table name for an element or attribute value to be mapped to a relational table.

Column

Specifies the column name for an element or attribute value to be mapped to a relational table.

Condition

Specifies a condition for the column.

- d. Type the node name in the **Node name** field in the **Details** box. For example:

Order

- e. If you specified **Attribute** , **Element** or **Text** for a bottom-level element as the Node type, select a column from the **Column** field in the **Details** box. For example:

Customer_Name

Restriction: New columns cannot be created using the administration wizard. If you specify **Column** as the node type, you can only select a column that already exists in your DB2 database.

- f. Click **Add** to add the new node.

You can modify a node later by clicking on it in the field on the left and making any needed modifications to it in the **Details** box. Click **Change** to update the element.

You can also add child elements or attributes to the node by highlighting the node repeating the add process.

- g. Continue editing the SQL map, or click **Next** to open the Specify a DAD window.

• **To remove a node:**

- a. Click on a node in the field on the left.
- b. Click **Remove**.

- c. Continue editing the SQL map, or click **Next** to open the Specify a DAD window.

Note that if you remove a bottom-level node, another element will become a bottom-level node and might need a column name defined for it.

13. Type the name of an output file for the modified DAD file in the **File name** field of the Specify a DAD window.
14. Click **Finish** to return to the LaunchPad window.

Using the command line

Use SQL mapping notation when you are composing XML document and want to use SQL.

The DAD file is an XML file that you can create using any text editor. The following steps show fragments from the samples appendix, “Document access definition files” on page 244. Please refer to these examples for more comprehensive information and context.

1. Open a text editor.
2. Create the DAD header:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path/dad.dtd" --> the path and file name of the DTD
for the DAD
```

3. Insert the <DAD></DAD> tags.
4. After the <DAD> tag, specify the DTD ID that associates the DAD file with the XML document DTD.

```
<dtdid>path/dtd_name.dtd --> the path and file name
of the DTD for your application
```

5. Specify whether to validate (that is, to use a DTD to ensure that the XML document is a valid XML document). For example:

```
<validation>NO</validation> --> specify YES or NO
```

6. Use the <Xcollection> element to define the access and storage method as XML collection. The access and storage methods define that the XML document will have content derived from data stored in DB2 tables.

```
<Xcollection>
</Xcollection>
```

7. Specify one or more SQL statements to query or insert data from or into DB2 tables. See “Mapping scheme requirements” on page 57 for guidelines. For example, you specify a single SQL query like in the following example:

```
<SQL_stmt>
SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
price, tax, ship_id, date, mode from order_tab o, part_tab p,
table (select db2xml.generate_unique()
as ship_id, date, mode, part_key from ship_tab) s
WHERE o.order_key = 1 and
p.price > 20000 and
p.order_key = o.order_key and
s.part_key = p.part_key
ORDER BY order_key, part_key, ship_id
</SQL_stmt>
```

8. Add the following prolog information:

```
<prolog>?xml version="1.0"?</prolog>
```

This exact text is required.

9. Add the <doctype></doctype> tags. For example:

```
<doctype>! DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>
```

If you need to specify an encoding value for internationalization, add the ENCODING attribute and value. See “Appendix C. Code page considerations” on page 251 to learn about encoding issues in a client/server environment.

10. Define the root node using the <root_node></root_node> tags. Inside the root_node, you specify the elements and attributes that make up the XML document.
11. Map the elements and attributes in the XML document to element and attribute nodes that correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.

- a. Define an <element_node> for each element in your XML document that maps to a column in a DB2 table.

```
<element_node name="name"></element_node>
```

An element_node can have the following nodes:

- attribute_node
 - child element_node
 - text_node
- b. Define an <attribute_node> for each attribute in your XML document that maps to a column in a DB2 table. See the example DTDs at the beginning of this section for SQL mapping, as well as the DTD for the DAD file in “Appendix A. DTD for the DAD file” on page 237, which provides the full syntax for the DAD file.

For example, you need an attribute key for an element <Order>. The value of key is stored in a column PART_KEY.

DAD file: In the DAD file, create an attribute node for key and indicate the table where the value of key is stored.

```
<attribute_node name="key">  
  <column name="part_key"/>  
</attribute_node>
```

Composed XML document: The value of key is taken from the PART_KEY column.

```
<Order key="1">
```

12. Create a <text_node> for every element or attribute that has content that will be derived from a DB2 table. The text node has a <column> element that specifies from which column the content is provided.

For example, you might have an XML element <Tax> with a value that will be taken from a column called TAX:

DAD element:

```
<element_node name="Tax">  
  <text_node>  
    <column name="tax"/>  
  </text_node>  
</element_node>
```

The column name must be in the SQL statement at the beginning of the DAD file.

Composed XML document:

```
<Tax>0.02</Tax>
```

The value 0.02 will be derived from the column TAX.

13. Ensure that you have an ending `</root_node>` tag after the last `</element_node>` tag.
14. Ensure that you have an ending `</Xcollection>` tag after the `</root_node>` tag.
15. Ensure that you have an ending `</DAD>` tag after the `</Xcollection>` tag.

Composing XML documents with RDB_node mapping

Use RDB_node mapping to compose XML documents using a XML-like structure.

This method uses the `<RDB_node>` to specify DB2 tables, column, and conditions for an element or attribute node. The `<RDB_node>` uses the following elements:

- `<table>`: defines the table corresponding to the element
- `<column>`: defines the column containing the corresponding element
- `<condition>`: optionally specifies a condition on the column

The child elements that are used in the `<RDB_node>` depend on the context of the node and use the following rules:

If the node type is:	RDB child element is used:		
	Table	Column	Condition ¹
Root element	Y	N	Y
Attribute	Y	Y	optional
Text	Y	Y	optional

(1) Required with multiple tables

Using the administration wizard

To create a DAD for composition, using RDB_node mapping:

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with DAD files** from the LaunchPad window. The Specify a DAD window opens.
3. Choose whether to edit an existing DAD file or to create a new DAD.

To edit an existing DAD:

- a. Type the DAD file name in the **File name** field or click ... to browse for an existing DAD.
- b. Verify that the wizard recognizes the specified DAD file.
 - If the wizard recognizes the specified DAD file, **Next** is selectable, and XML collection RDB node mapping is displayed in the **Type** field.
 - If the wizard does not recognize the specified DAD file, **Next** is not selectable. Either retype the DAD file name in the **File name** field or click ... to browse again for an existing DAD file. Continue these steps until **Next** is selectable.
- c. Click **Next** to open the Select Validation window.

To create a new DAD:

- a. Leave the **File name** field blank.
- b. Select XML collection RDB_node mapping from the **Type** menu.
- c. Click **Next** to open the Select Validation window.

4. In the Select Validation window, choose whether to validate your XML documents with a DTD.
 - To validate:
 - a. Click **Validate XML documents with the DTD**.
 - b. Select the DTD to be used for validation from the **DTD ID** menu.

If XML Extender does not find the specified DTD in the DTD reference table, it searches for the specified DTD on the file system and uses it to validate.

- Click **Do NOT validate XML documents with the DTD** to continue without validating your XML documents.
5. Click **Next** to open the Specify Text window.
 6. Type the prolog name in the **Prolog** field of the Specify Text window.
`<?xml version="1.0" ?>`

If you are editing an existing DAD, the prolog is automatically displayed in the **Prolog** field.

7. Enter the document type of the XML document in the **Doctype** field of the Specify Text window.

If you are editing an existing DAD, the document type is automatically displayed in the **Doctype** field.

8. Click **NEXT** to open the RDB Mapping window.
9. Select an element or attribute node to map from by clicking on it in the field on the left of the RDB Mapping window.

Map the elements and attributes in the XML document to element and attribute nodes which correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.

10. **To add the root node:**
 - a. Select the **Root** icon.
 - b. Click **New Element** to define a new node.
 - c. In the **Details** box, specify **Node type** as **Element**.
 - d. Enter the name of the top level node in the **Node name** field.
 - e. Click **Add** to create the new node.

You have created the root node or element, which is the parent to all the other element and attribute nodes in the map. The root node has table child elements and a join condition.

- f. Add table nodes for each table that is part of the collection.
 - 1) Highlight the root node name and select **New Element**.
 - 2) In the **Details** box, specify **Node type** as **Table**.
 - 3) Select the name of the table from **Table name**. The table must already exist.
 - 4) Click **Add** to add the table node.
 - 5) Repeat these steps for each table.
- g. Add a join condition for the table nodes.
 - 1) Highlight the root node name and select **New Element**.
 - 2) In the **Details** box, specify **Node type** as **Condition**.
 - 3) In the **Condition** field, enter the join condition using the following syntax:

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

- 4) Click **Add** to add the condition.
11. **To add an element or attribute node:**
 - a. Click on a parent node in the field on the left to add a child element or attribute.
 - b. Click **New Element**. If you have not selected a parent node, **New Element** is not selectable.
 - c. Select a node type from the **Node type** menu in the DETAILS box.
The **Node type** menu displays only the node types that are valid at that point in the map. **Element** or **Attribute**.
 - d. Specify a node name in the **Node name** field.
 - e. Click **Add** to add the new node.
 - f. **To map the contents of an element or attribute node to a relational table:**
 - 1) Specify a text node.
 - a) Click the parent node.
 - b) Click **New Element**.
 - c) In the **Node type** field, select **Text**.
 - d) Select **Add** to add the node.
 - 2) Add a table node.
 - a) Select the text node you just created and click **New Element**.
 - b) In the **Node type** field, select **Table** and specify a table name for the element.
 - c) Click **Add** to add the node.
 - 3) Add a column node.
 - a) Select the text node again and click **New Element**.
 - b) In the **Node type** field, select **Column** and specify a column name for the element.
 - c) Click **Add** to add the node.

Restriction: New columns cannot be created using the administration wizard. If you specify Column as the node type, you can only select a column that already exists in your DB2 database.
 - 4) Optionally add a condition for the column.
 - a) Select the text node again and click **New Element**.
 - b) In the **Node type** field, select **Condition** and the condition with the syntax:
column_name LIKE|<|>|= value
 - c) Click **Add** to add the node.
 - g. Continue editing the RDB map or click **Next** to open the Specify a DAD window.
12. **To remove a node:**
 - a. Click on a node in the field on the left.
 - b. Click **Remove**.
 - c. Continue editing the RDB_node map or click **Next** to open the Specify a DAD window.
13. Type in an output file name for the modified DAD in the **File name** field of the Specify a DAD window.
14. Click **Finish** to remove the node and return to the LaunchPad window.

Using the command line

The DAD file is an XML file that you can create using any text editor. The following steps show fragments from the samples appendix, "Document access definition files" on page 244. Please refer to these examples for more comprehensive information and context.

1. Open a text editor.

2. Create the DAD header:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "path/dad.dtd" --> the path and file name of the DTD
      for the DAD
```

3. Insert the <DAD></DAD> tags.

4. After the <DAD> tag, specify the DTD ID that associates the DAD file with the XML document DTD.

```
<dtdid>path/dtd_name.dtd --> the path and file name of the DTD
      for your application
```

5. Specify whether to validate (that is, to use a DTD to ensure that the XML document is a valid XML document). For example:

```
<validation>NO</validation> --> specify YES or NO
```

6. Use the <Xcollection> element to define the access and storage method as XML collection. The access and storage methods define that the XML data is stored in a collection of DB2 tables.

```
<Xcollection>
</Xcollection>
```

7. Add the following prolog information:

```
<prolog>?xml version="1.0"?</prolog>
```

This exact text is required.

8. Add the <doctype></doctype> tags. For example:

```
<doctype>! DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>
```

If you need to specify an encoding value for internationalization, add the ENCODING attribute and value. See "Appendix C. Code page considerations" on page 251 to learn about encoding issues in an client/server environment.

9. Define the root node using the <root_node>. Inside the root_node, you specify the elements and attributes that make up the XML document.

10. Map the elements and attributes in the XML document to element and attribute nodes that correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.

a. Define a root element_node. This element_node contains:

- An RDB_node which specifies table_nodes with a join condition to specify the collection
- Child elements
- Attributes

To specify the table nodes and condition:

1) Create an RDB_node element: For example:

```
<RDB_node>
</RDB_node>
```

2) Define a <table_node> for each table that contains data to be included in the XML document. For example, if you have three tables, ORDER_TAB, PART_TAB, and SHIP_TAB, that have column data to be in the document, create a table node for each. For example:

```

<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB"></RDB_node>

```

- 3) Optionally, specify a key column for each table when you plan to enable this collection. The key attribute is not normally required for composition; however, when you enable a collection, the DAD file used must support both composition and decomposition. For example:

```

<RDB_node>
<table name="ORDER_TAB" key="order_key">
<table name="PART_TAB" key="part_key">
<table name="SHIP_TAB" key="date mode">
</RDB_node>

```

- 4) Define a join condition for the tables in the collection. The syntax is

```

table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...

```

For example:

```

<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB">
<condition>
  order_tab.order_key = part_tab.order_key AND
  part_tab.part_key = ship_tab.part_key
</condition>
</RDB_node>

```

- b. Define an <element_node> tag for each element in your XML document that maps to a column in a DB2 table. For example:

```

<element_node name="name">
</element_node>

```

An element node can have one of the following types of elements:

- <text_node>: to specify that the element has content to a DB2 table; the element does not have child elements
- <attribute_node>: to specify an attribute. Attribute nodes are defined in the next step.

The text_node contains an <RDB_node> to map content to a DB2 table and column name.

RDB_nodes are used for bottom-level elements that have content to map to a DB2 table. An RDB_node has the following child elements:

- <table>: defines the table corresponding to the element
- <column>: defines the column containing the corresponding element and specifies the column type with the type attribute
- <condition>: optionally specifies a condition on the column

For example, you might have an XML element <Tax> that maps to a column called TAX:

XML document:

```

<Tax>0.02</Tax>

```

In this case, you want the value 0.02 to be a value in the column TAX.

```

<element_node name="Tax">
  <text_node>
    <RDB_node>
      <table name="part_tab"/>
      <column name="tax"/>
    </RDB_node>
  </text_node>
</element_node>

```

In this example, the <RDB_node> specifies that the value of the <Tax> element is a text value, the data is stored in the PART_TAB table in the TAX column.

See the example DAD files in “Document access definition files” on page 244 for RDB_node mapping, as well as the DTD for the DAD file in “Appendix A. DTD for the DAD file” on page 237, which provides the full syntax for the DAD file.

- c. Optionally, add a type attribute to each <column> element when you plan to enable this collection. The type attribute is not normally required for composition; however, when you enable a collection, the DAD file used must support both composition and decomposition. For example:

```

<column name="tax" type="real"/>

```

- d. Define an <attribute_node> for each attribute in your XML document that maps to a column in a DB2 table. For example:

```

<attribute_node name="key">
</attribute_node>

```

The attribute_node has an <RDB_node> to map the attribute value to a DB2 table and column. An <RDB_node> has the following child elements:

- <table>: defines the table corresponding to the element
- <column>: defines the column containing the corresponding element
- <condition>: optionally specifies a condition on the column

For example, you might want to have an attribute key for an element <Order>. The value of key needs to be stored in a column PART_KEY. In the DAD file, create an <attribute_node> for key and indicate the table where the value is to be stored.

DAD file:

```

<attribute_node name="key">
  <RDB_node>
    <table name="part_tab">
      <column name="part_key"/>
    </RDB_node>
</attribute_node>

```

Composed XML document:

```

<Order key="1">

```

11. Ensure that you have an ending </root_node> tag after the last </element_node> tag.
12. Ensure that you have an ending </Xcollection> tag after the </root_node> tag.
13. Ensure that you have an ending </DAD> tag after the </Xcollection> tag.

Specifying a stylesheet for the XML document

When composing documents, the XML Extender also supports processing instructions for stylesheets, using the <stylesheet> element. The processing instructions must be inside the <Xcollection> root element, located with the <doctype> and <prolog> defined for the XML document structure. For example:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dtd\dad.dtd">
<DAD>
<SQL_stmt>
    ...
</SQL_stmt>
<Xcollection>
    ...
<prolog>...</prolog>
<doctype>...</doctype>
<stylesheet?xml-stylesheet type="text/css" href="order.css"?</stylesheet>
<root_node>...</root_node>
    ...

</Xcollection>
    ...
</DAD>
```

Decomposing XML documents with RDB_node mapping

Use RDB_node mapping to decompose XML documents. This method uses the <RDB_node> to specify DB2 tables, column, and conditions for an element or attribute node. The <RDB_node> uses the following elements:

- <table>: defines the table corresponding to the element
- <column>: defines the column containing the corresponding element
- <condition>: optionally specifies a condition on the column

The child elements that are used in the <RDB_node> depend on the context of the node and use the following rules:

If the node type is:	RDB child element is used:		
	Table	Column	Condition ¹
Root element	Y	N	Y
Attribute	Y	Y	optional
Text	Y	Y	optional

(1) Required with multiple tables

Using the administration wizard

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with DAD files** from the LaunchPad window. The Specify a DAD windows opens.
3. Choose whether to edit an existing DAD file or to create a new DAD.

To edit an existing DAD:

- a. Type the DAD file name in the **File name** field or click ... to browse for an existing DAD.
- b. Verify that the wizard recognizes the specified DAD file.
 - If the wizard recognizes the specified DAD file, **Next** is selectable, and XML collection RDB node mapping is displayed in the **Type** field.

- If the wizard does not recognize the specified DAD file, **Next** is not selectable. Either retype the DAD file name in the **File name** field or click ... to browse again for an existing DAD file. Continue these steps until **Next** is selectable.
- c. Click **Next** to open the Select Validation window.

To create a new DAD:

- a. Leave the **File name** field blank.
 - b. Select XML collection RDB_node mapping from the **Type** menu.
 - c. Click **Next** to open the Select Validation window.
4. In the Select Validation window, choose whether to validate your XML documents with a DTD.
- To validate:
 - a. Click **Validate XML documents with the DTD**.
 - b. Select the DTD to be used for validation from the **DTD ID** menu.

If XML Extender does not find the specified DTD in the DTD reference table, it searches for the specified DTD on the file system and uses it to validate.

- Click **Do NOT validate XML documents with the DTD** to continue without validating your XML documents.
5. Click **Next** to open the Specify Text window.
6. If you are decomposing an XML document only, ignore the **Prolog** field. If you are using the DAD file for both composition and decomposition, type the prolog name in the **Prolog** field of the Specify Text window. The prolog is not required if you are decomposing XML documents into DB2 data.
- ```
<?xml version="1.0"?>
```

If you are editing an existing DAD, the prolog is automatically displayed in the **Prolog** field.

7. If you are decomposing an XML document only, ignore the **Doctype** field. If you are using the DAD file for both composition and decomposition, enter the document type of the XML document in the **Doctype** field.
- If you are editing an existing DAD, the document type is automatically displayed in the **Doctype** field.
8. Click **NEXT** to open the RDB Mapping window.
9. Select an element or attribute node to map from by clicking on it in the field on the left of the RDB Mapping window.
- Map the elements and attributes in the XML document to element and attribute nodes which correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.

10. **To add the root node:**

- a. Select the **Root** icon.
- b. Click **New Element** to define a new node.
- c. In the **Details** box, specify **Node type** as **Element**.
- d. Enter the name of the top level node in the **Node name** field.
- e. Click **Add** to create the new node.

You have created the root node or element, which is the parent to all the other element and attribute nodes in the map. The root node has table child elements and a join condition.

- f. Add table nodes for each table that is part of the collection.

- 1) Highlight the root node name and select **New Element**.
  - 2) In the **Details** box, specify **Node type** as **Table**.
  - 3) Select the name of the table from **Table name**. The table must already exist.
  - 4) Specify a key column for the table in the **Table key** field.
  - 5) Click **Add** to add the table node.
  - 6) Repeat these steps for each table.
- g. Add a join condition for the table nodes.
- 1) Highlight the root node name and select **New Element**.
  - 2) In the **Details** box, specify **Node type** as **Condition**.
  - 3) In the **Condition** field, enter the join condition using the following syntax:
 

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```
  - 4) Click **Add** to add the condition.

You can now add child elements and attributes to this node.

11. **To add an element or attribute node:**

- a. Click on a parent node in the field on the left to add a child element or attribute.
 

If you have not selected a parent node, **New** is not selectable.
- b. Click **New Element**.
- c. Select a node type from the **Node type** menu in the DETAILS box.
 

The **Node type** menu displays only the node types that are valid at that point in the map. **Element** or **Attribute**.
- d. Specify a node name in the **Node name** field.
- e. Click **Add** to add the new node.
- f. **To map the contents of an element or attribute node to a relational table:**
  - 1) Specify a text node.
    - a) Click the parent node.
    - b) Click **New Element**.
    - c) In the **Node type** field, select **Text**.
    - d) Select **Add** to add the node.
  - 2) Add a table node.
    - a) Select the text node you just created and click **New Element**.
    - b) In the **Node type** field, select **Table** and specify a table name for the element.
    - c) Click **Add** to add the node.
  - 3) Add a column node.
    - a) Select the text node again and click **New Element**.
    - b) In the **Node type** field, select **Column** and specify a column name for the element.
    - c) Specify a base data type for the column in the **Type** field, to specify what type the column must be to store the untagged data.
    - d) Click **Add** to add the node.

**Restriction:** New columns cannot be created using the administration wizard. If you specify Column as the node type, you can only select a column that already exists in your DB2 database.

- 4) Optionally add a condition for the column.
  - a) Select the text node again and click **New Element**.
  - b) In the **Node type** field, select **Condition** and the condition with the syntax:  
`column_name LIKE|<|>|= value`
  - c) Click **Add** to add the node.

You can modify these nodes by selecting the node, change the fields in the **Details** box, and clicking **Change**.

- g. Continue editing the RDB map or click **Next** to open the Specify a DAD window.
12. **To remove a node:**
  - a. Click on a node in the field on the left.
  - b. Click **Remove**.
  - c. Continue editing the RDB\_node map or click **Next** to open the Specify a DAD window.
13. Type in an output file name for the modified DAD in the **File name** field of the Specify a DAD window.
14. Click **Finish** to remove the node and return to the LaunchPad window.

### Using the command line

The DAD file is an XML file that you can create using any text editor. The following steps show fragments from the samples appendix, "Document access definition files" on page 244. Please refer to these examples for more comprehensive information and context.

1. Open a text editor.
2. Create the DAD header:  
`<?xml version="1.0"?>  
<!DOCTYPE DAD SYSTEM "path/dad.dtd" --> the path and file name of the DTD  
for the DAD`
3. Insert the `<DAD></DAD>` tags.
4. After the `<DAD>` tag, specify the DTD ID that associates the DAD file with the XML document DTD.

`<dtdid>path/dtd_name.dtd --> the path and file name of the DTD  
for your application`

5. Specify whether to validate (that is, to use a DTD to ensure that the XML document is a valid XML document). For example:  
`<validation>NO</validation> --> specify YES or NO`
6. Use the `<Xcollection>` element to define the access and storage method as XML collection. The access and storage methods define that the XML data is stored in a collection of DB2 tables.

`<Xcollection>  
</Xcollection>`

7. Add the following prolog information:  
`<prolog>?xml version="1.0"?</prolog>`

This exact text is required.

8. Add the `<doctype></doctype>` tags. For example:  
`<doctype>! DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>`

If you need to specify an encoding value for internationalization, add the ENCODING attribute and value. See “Appendix C. Code page considerations” on page 251 to learn about encoding issues in an client/server environment.

9. Define the root\_node using the <root\_node></root\_node> tags. Inside the root\_node, you specify the elements and attributes that make up the XML document.
10. After the <root\_node> tag, map the elements and attributes in the XML document to element and attribute nodes that correspond to DB2 data. These nodes provide a path from the XML data to the DB2 data.
  - a. Define a top level, root element\_node. This element\_node contains:
    - Table nodes with a join condition to specify the collection.
    - Child elements
    - Attributes

To specify the table nodes and condition:

- 1) Create an RDB\_node element: For example:

```
<RDB_node>
</RDB_node>
```

- 2) Define a <table\_node> for each table that contains data to be included in the XML document. For example, if you have three tables, ORDER\_TAB, PART\_TAB, and SHIP\_TAB, that have column data to be in the document, create a table node for each. For example:

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB"></RDB_node>
```

- 3) Define a join condition for the tables in the collection. The syntax is:

```
table_name.table_column = table_name.table_column AND
table_name.table_column = table_name.table_column ...
```

For example:

```
<RDB_node>
<table name="ORDER_TAB">
<table name="PART_TAB">
<table name="SHIP_TAB">
<condition>
 order_tab.order_key = part_tab.order_key AND
 part_tab.part_key = ship_tab.part_key
</condition>
</RDB_node>
```

- 4) Specify a primary key for each table. The primary key consists of a single column or multiple columns, called a composite key. To specify the primary key, add an attribute key to the table element of the RDB\_node. The following example defines a primary key for each of the tables in the RDB\_node of the root element\_node Order:

```
<element_node name="Order">
 <RDB_node>
 <table name="order_tab" key="order_key"/>
 <table name="part_tab" key="part_key price"/>
 <table name="ship_tab" key="date mode"/>
 <condition>
 order_tab.order_key = part_tab.order_key AND
 part_tab.part_key = ship_tab.part_key
 </condition>
 </RDB_node>
```

The information specified for decomposition is ignored when composing an XML document.

The key attribute is required for decomposition, and when you enable a collection because the DAD file used must support both composition and decomposition.

- b. Define an `<element_node>` tag for each element in your XML document that maps to a column in a DB2 table. For example:

```
<element_node name="name">
</element_node>
```

An element node can have one of the following types of elements:

- `<text_node>`: to specify that the element has content to a DB2 table; in this case it does not have child elements.
- `<attribute_node>`: to specify an attribute; attribute nodes are defined in the next step
- child elements

The `text_node` contains an `RDB_node` to map content to a DB2 table and column name.

`RDB_nodes` are used for bottom-level elements that have content to map to a DB2 table. An `RDB_node` has the following child elements:

- `<table>`: defines the table corresponding to the element
- `<column>`: defines the column containing the corresponding element
- `<condition>`: optionally specifies a condition on the column

For example, you might have an XML element `<Tax>` for which you want to store the untagged content in a column called `TAX`:

**XML document:**

```
<Tax>0.02</Tax>
```

In this case, you want the value `0.02` to be stored in the column `TAX`.

In the DAD file, you specify an `<RDB_node>` to map the XML element to the DB2 table and column.

**DAD file:**

```
<element_node name="Tax">
 <text_node>
 <RDB_node>
 <table name="part_tab"/>
 <column name="tax"/>
 </RDB_node>
 </text_node>
</element_node>
```

The `<RDB_node>` specifies that the value of the `<Tax>` element is a text value, the data is stored in the `PART_TAB` table in the `TAX` column.

- c. Define an `<attribute_node>` for each attribute in your XML document that maps to a column in a DB2 table. For example:

```
<attribute_node name="key">
</attribute_node>
```

The `attribute_node` has an `RDB_node` to map the attribute value to a DB2 table and column. An `RDB_node` has the following child elements:

- `<table>`: defines the table corresponding to the element
- `<column>`: defines the column containing the corresponding element
- `<condition>`: optionally specifies a condition on the column

For example, you might have an attribute key for an element `<Order>`. The value of key needs to be stored in a column `PART_KEY`.

**XML document:**

```
<Order key="1">
```

In the DAD file, create an `attribute_node` for key and indicate the table where the value of 1 is to be stored.

**DAD file:**

```
<attribute_node name="key">
 <RDB_node>
 <table name="part_tab">
 <column name="part_key"/>
 </RDB_node>
 </attribute_node>
```

11. Specify the column type for the `RDB_node` for each `attribute_node` and `text_node`. This ensures the correct data type for each column where the untagged data will be stored. To specify the column types, add the attribute type to the column element. The following example defines the column type as an `INTEGER`:

```
<attribute_node name="key">
 <RDB_node>
 <table name="order_tab"/>
 <column name="order_key" type="integer"/>
 </RDB_node>
</attribute_node>
```

12. Ensure that you have an ending `</root_node>` tag after the last `</element_node>` tag.
13. Ensure that you have an ending `</Xcollection>` tag after the `</root_node>` tag.
14. Ensure that you have an ending `</DAD>` tag after the `</Xcollection>` tag.

---

## Enabling XML collections

Enabling an XML collection parses the DAD file to identify the tables and columns related to the XML document, and records control information in the `XML_USAGE` table. Enabling an XML collection is optional for:

- Decomposing an XML document and storing the data in new DB2 tables
- Composing an XML document from existing data in multiple DB2 tables

If the same DAD file is used for composing and decomposing, you can enable the collection for both composition and decomposition.

You can enable an XML collection through the XML Extender administration wizard, using the `DXXADM` command with the `enable_collection` option, or you can use the XML Extender stored procedure `dxxEnableCollection()`.

## Using the administration wizard

Use the following steps to enable an XML collection.

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with XML Collections** from the LaunchPad window. The Select a Task window opens.
3. Click **Enable a Collection** and then **Next**. The Enable a Collection window opens.
4. Select the name of the collection you want to enable in the **Collection name** field from the pull-down menu.
5. Type the DAD file name in the **DAD file name** field or click ... to browse for an existing DAD file.
6. Optionally, type the name of a previously created table space in the **Table space** field.  
The table space will contain new DB2 tables generated for decomposition.
7. Click FINISH to enable the collection and return to the LaunchPad window.
  - If the collection is successfully enabled, an Enabled collection is successful message is displayed.
  - If the collection is not successfully enabled, an error message is displayed. Continue the preceding steps until the collection is successfully enabled.

## Using the command line

To enable an XML collection, enter the DXXADM command:

### Syntax:

```
►►—enable_collection—-a—subsystem_name—collection—DAD_file—————►

►—————
└─t—tablespace—┘
```

### Parameters:

**-a** *subsystem\_name*

The name of the DB2 subsystem.

*collection*

The name of the XML collection. This value is used as a parameter for the XML collection stored procedures.

*DAD\_file*

The name of the file that contains the document access definition (DAD).

*tablespace*

An existing table space that contains new DB2 tables that were generated for decomposition. If not specified, the default table space is used.

**Example:** The following example enables a collection called sales\_ord in the database SALES\_DB using the command line. The DAD file uses SQL mapping and can be found in “DAD file: XML collection - SQL mapping” on page 245.

```
dxxadm enable_collection -a SUBSYS1 using ORDRPSC SALES_ORD
'ORDPRJ.WORK.DAD(GETSTART_XCOLLECTION)'
```



After you enable the XML collection, you can compose or decompose XML documents using the XML Extender stored procedures.

---

## Disabling XML collections

Disabling an XML collection removes the record in the XML\_USAGE table that identify tables and columns as part of a collection. It does not drop any data tables. You disable a collection when you want to update the DAD and need to re-enable a collection, or to drop a collection.

You can disable an XML collection through the XML Extender administration wizard, using the DXXADM command with the `disable_collection` option, or using the XML Extender stored procedure `dxxDisableCollection()`.

## Using the administration wizard

Use the following steps to disable an XML collection.

1. Set up and start the administration wizard. See “Starting the administration wizard” on page 65 for details.
2. Click **Work with XML Collections** from the LaunchPad window to view the XML Extender collection related tasks. The Select a Task window opens.
3. Click **Disable an XML Collection** and then **Next** to disable an XML collection. The Disable a Collection window opens.
4. Type the name of the collection you want to disable in the **Collection name** field.
5. Click **Finish** to disable the collection and return to the LaunchPad window.
  - If the collection is successfully disabled, an Disabled collection is successful message is displayed.
  - If the collection is not successfully disabled, an error box is displayed. Continue the preceding steps until the collection is successfully disabled.

## Using the command line

To disable an XML collection, enter the DXXADM command:

### Syntax:

```
►►—dxxadm—disable_collection—-a—subsystem_name—collection—►►
```

### Parameters:

- a *subsystem\_name*

The name of the DB2 subsystem.

*collection*

The name of the XML collection. This value is used as a parameter for the XML collection stored procedures.

### Example:

```
dxxadm disable_collection -a SUBSYS1 SALES_ORD
```



---

## **Part 3. Programming**

This part describes programming techniques for managing your XML data.



---

## Chapter 8. Managing XML column data

When using XML columns, you store an entire XML document as column data. This access and storage method allows you to keep the XML document intact, while giving you the ability to index and search the document, retrieve data from the document, and update the document. An XML column contains XML documents in their native format in DB2 as column data. After you enable a database for XML, the following user-defined types (UDTs) are available for your use:

### **XMLCLOB**

XML document content that is stored as a character large object (CLOB) in DB2

### **XMLVARCHAR**

XML document content that is stored as a VARCHAR in DB2

### **XMLFile**

XML document that is stored in a file on a local file system

You can create or alter application tables using XML UDTs as column data types. These tables are known as XML tables. To learn how to create or alter a table for XML, see “Creating or altering an XML table” on page 76.

After you enable a column for XML, you can begin managing the contents of the XML column. After the XML column is created, you can perform the following management tasks:

- Store XML documents in DB2
- Retrieve XML data or documents from DB2
- Update XML documents
- Delete XML data or documents

To perform these tasks, you can use two methods:

- *Default casting functions*, which cast the SQL base type to the XML Extender user-defined types. A casting function converts instances of a data type (origin) into instances of a different data type (target).
- XML Extender-provided user-defined functions (UDFs)

This book describes both methods for each task.

---

## User-defined types and user-defined function names

The full name of a DB2 function is: *schema-name.function-name*, where *schema-name* is an identifier that provides a logical grouping for the SQL objects. The schema name for XML Extender UDFs is DB2XML. The DB2XML schema name is also the qualifier for the XML Extender UDTs. In this book, references are made only to the function name.

You can specify UDTs and UDFs without the schema name if you add the schema to the function path. The function path is an ordered list of schema names. DB2 uses the order of schema names in the list to resolve references to functions and UDTs. You can specify the function path by specifying the SQL statement SET CURRENT FUNCTION PATH. This sets the function path in the CURRENT FUNCTION PATH special register.

For the XML Extender, it is a good idea to add the DB2XML schema to the function path. This allows you to enter XML Extender UDF and UDT names without having to qualify them with DB2XML. The following example shows how to add the DB2XML schema to the function path:

```
SET CURRENT FUNCTION PATH = DB2XML, CURRENT FUNCTION PATH
```

**Important:** Do not add DB2XML as the first schema in the function path if you log on as DB2XML; DB2XML is automatically set as the first schema when you log on as DB2XML. This generates an error condition because your function path will begin with two DB2XML schemas.

## Storing data

Using the XML Extender, you can insert intact XML documents into an XML column. If you define side tables, the XML Extender automatically updates these tables. When you store an XML document directly, the XML Extender stores the base type as an XML type.

### Task overview:

1. Ensure that you have created or updated the DAD file.
2. Determine what data type to use when you store the document.
3. Choose a method for storing the data in the DB2 table (casting functions or UDFs).
4. Specify an SQL INSERT statement that specifies the XML table and column to contain the XML document.

The XML Extender provides two methods for storing XML documents: default casting functions and storage UDFs.

Table 13 shows when to use each method.

Table 13. The XML Extender storage functions

Base type	Store in DB2 as...		
	XMLVARCHAR	XMLCLOB	XMLFILE
VARCHAR	XMLVARCHAR()	N/A	XMLFileFromVarchar()
CLOB	N/A	XMLCLOB()	XMLFileFromCLOB()
FILE	XMLVarcharFromFile()	XMLCLOBFromFile()	XMLFILE

### Use a default casting function

For each UDT, a default casting function exists to cast the SQL base type to the UDT. You can use the XML Extender-provided casting functions in your VALUES clause to insert data. Table 14 shows the provided casting functions:

Table 14. The XML Extender default cast functions

Casting used in SELECT clause	Return type	Description
XMLVARCHAR(VARCHAR)	XMLVARCHAR	Input from memory buffer of VARCHAR
XMLCLOB(CLOB)	XMLCLOB	Input from memory buffer of CLOB or a CLOB locator
XMLFILE(VARCHAR)	XMLFILE	Only store file name

**Example:** The following statement inserts a cast VARCHAR type into the XMLVARCHAR type:

```
INSERT INTO sales_tab
VALUES('123456', 'Sriram Srinivasan', DB2XML.XMLVarchar(:xml_buff))
```

**Use a storage UDF:**

For each XML Extender UDT, a storage UDF exists to import data into DB2 from a resource other than its base type. For example, if you want to import an XML file document to DB2 as a XMLCLOB, you can use the function XMLCLOBFromFile().

Table 15 shows the storage functions provided by the XML Extender.

*Table 15. The XML Extender storage UDFs*

Storage user-defined function	Return type	Description
XMLVarcharFromFile()	XMLVARCHAR	Reads an XML document from a file on the server and returns the value of the XMLVARCHAR type.
XMLCLOBFromFile()	XMLCLOB	Reads an XML document from a file on the server and returns the value of the XMLCLOB type.
XMLFileFromVarchar()	XMLFILE	Reads an XML document from memory as VARCHAR, writes it to an external file, and returns the value of the XMLFILE type, which is the file name.
XMLFileFromCLOB()	XMLFILE	Reads an XML document from memory as CLOB or a CLOB locator, writes it to an external file, and returns the value of the XMLFILE type, which is the file name.

**Example:** The following statement stores a record in an XML table using the XMLCLOBFromFile() function as an XMLCLOB.

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
VALUES('1234', 'Sriram Srinivasan',
XMLCLOBFromFile('dxx_install/samples/cmd/getstart.xml'))
```

The preceding example imports the XML object from the file named *dxx\_install/cmd/getstart.xml* to the column ORDER in the table SALES\_TAB.

---

## Retrieving data

Using the XML Extender, you can retrieve either an entire document or the contents of elements and attributes. When you retrieve an XML column directly, the XML Extender returns the UDT as the column type. For details on retrieving data, see the following sections:

- “Retrieving an entire document” on page 110
- “Retrieving element contents and attribute values” on page 111

The XML Extender provides two methods for retrieving data: default casting functions and the Content() overloaded UDF. Table 16 shows when to use each method.

Table 16. The XML Extender retrieval functions

XML type	Retrieve from DB2 as...		
	VARCHAR	CLOB	FILE
XMLVARCHAR	VARCHAR	N/A	Content()
XMLCLOB	N/A	XMLCLOB	Content()
XMLFILE	N/A	Content()	FILE

## Retrieving an entire document

### Task overview:

1. Ensure that you have stored the XML document in an XML table and determine what data you want to retrieve.
2. Choose a method for retrieving the data in the DB2 table (casting functions or UDFs).
3. If using the overloaded Content() UDF, determine which data type is associated with the data that is being retrieved, and which data type is to be exported.
4. Specify an SQL query that specifies the XML table and column from which to retrieve the XML document.

The XML Extender provides two methods for retrieving data:

### Use a default casting function

Use the default casting function provided by DB2 for UDTs to convert an XML UDT to an SQL base type, and then operate on it. You can use the XML Extender-provided casting functions in your SELECT statement to retrieve data. Table 17 shows the provided casting functions:

Table 17. The XML Extender default cast functions

Casting used in select clause	Return type	Description
varchar(XMLVARCHAR)	VARCHAR	XML document in VARCHAR
clob(XMLCLOB)	CLOB	XML document in CLOB
varchar(XMLFile)	VARCHAR	XML file name in VARCHAR

**Example:** The following example retrieves the XMLVARCHAR and stores it in memory as a VARCHAR data type:

```
EXEC SQL SELECT DB2XML.Varchar(ORDER) from SALES_TAB
```

### Use the Content() overloaded UDF

Use the Content() UDF to retrieve the document content from external storage to memory, or export the document from internal storage to an *external file*, a file external to DB2, on the DB2 server.

For example, you might have your XML document stored as XMLFILE and you want to operate on it in memory, you can use the Content() UDF, which can take an XMLFILE data type as input and return a CLOB.

The Content() UDF performs two different retrieval functions, depending on the specified data type. It:



### Retrieves a document from external storage and puts it in memory

You can use Content() to retrieve the XML document to a memory buffer or a CLOB *locator* (a host variable with a value that represents a single LOB value in the database server) when the document is stored as the external file . Use the following function syntax, where *xmlobj* is the XML column being queried:

**XMLFILE to CLOB:** Retrieves data from a file and exports to a CLOB locator.

```
Content(xmlobj XMLFile)
```

### Retrieves a document from internal storage and exports it to an external file

You can also use Content() to retrieve an XML document that is stored inside DB2 as an XMLCLOB data type and export it to a file on the database server file system. It returns the name of the file of VARCHAR type. Use the following function syntax, where *xmlobj* is the XML column that is being queried and *filename* is the external file. *XML type* can be of XMLVARCHAR or XMLCLOB data type.

**XML type to external file:** Retrieves the XML content that is stored as an XML data type and exports it to an external file.

```
Content(xmlobj XML type, filename varchar(512))
```

Where:

*xmlobj* Is the name of the XML column from which the XML content is to be retrieved; *xmlobj* can be of type XMLVARCHAR or XMLCLOB.

*filename*

Is the name of the file in which the XML data is to be stored.

In the example below, a small C program segment with *embedded SQL* (SQL statements coded within an application program) illustrates how an XML document is retrieved from a file to memory. This example assumes that the column ORDER is of the XMLFILE type.

```
EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;
EXEC SQL CONNECT TO SALES_DB
EXEC SQL DECLARE c1 CURSOR FOR
 SELECT Content(order) from sales_tab
 EXEC SQL OPEN c1;
do {
 EXEC SQL FETCH c1 INTO :xml_buff;
 if (SQLCODE != 0) {
 break;
 }
 else {
 /* do whatever you need to do with the XML doc in buffer */
 }
}
EXEC SQL CLOSE c1;
EXEC SQL CONNECT RESET;
```

## Retrieving element contents and attribute values

You can retrieve (extract) the content of an element or an attribute value from one or more XML documents (single document or collection document search). The

XML Extender provides user-defined extracting functions that you can specify in the SQL SELECT clause for each of the SQL data types.

Retrieving the content and values of elements and attributes is useful in developing your applications, because you can access XML data as relational data. For example, you might have 1000 XML documents that are stored in the column ORDER in the table SALES\_TAB. You can retrieve the names of all customers who have ordered items using the following SQL statement with the extracting UDF in the SELECT clause to retrieve this information:

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
 WHERE price > 2500.00
```

In this example, the extracting UDF retrieves the element <customer> from the column ORDER as a VARCHAR data type. The location path is /Order/Customer/Name (see “Location path” on page 61 for location path syntax). Additionally, the number of returned values is reduced by using a WHERE clause, which specifies that only the contents of the <customer> element with a subelement <ExtendedPrice> has a value greater than 2500.00.

**To extract element content or attribute values:** Use the extracting UDFs listed in Table 18 on page 113 by using the following syntax as either table or scalar functions:

```
extractretrieved_datatype(xmlobj, path)
```

Where:

*retrieved\_datatype*

Is the data type that is returned from the extracting function; it can be one of the following types:

- INTEGER
- SMALLINT
- DOUBLE
- REAL
- CHAR
- VARCHAR
- CLOB
- DATE
- TIME
- TIMESTAMP
- FILE

*xmlobj* Is the name of the XML column from which the element or attribute is to be extracted. This column must be defined as one of the following XML user-defined types:

- XMLVARCHAR
- XMLCLOB as LOCATOR
- XMLFILE

*path* Is the location path of the element or attribute in the XML document (such as /Order/Customer/Name). See “Location path” on page 61 for location path syntax.

**Important:** Note that the extracting UDFs support location paths that have predicates with attributes, but not elements. For example, the following predicate is supported:

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

The following predicate is not supported:

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

Table 18 shows the extracting functions, both in scalar and table format:

Table 18. The XML Extender extracting functions

Scalar function	Table function	Returned column name (table function)	Return type
extractInteger()	extractIntegers()	returnedInteger	INTEGER
extractSmallint()	extractSmallints()	returnedSmallint	SMALLINT
extractDouble()	extractDoubles()	returnedDouble	DOUBLE
extractReal()	extractReals()	returnedReal	REAL
extractChar()	extractChars()	returnedChar	CHAR
extractVarchar()	extractVarchars()	returnedVarchar	VARCHAR
extractCLOB()	extractCLOBs()	returnedCLOB	CLOB
extractDate()	extractDates()	returnedDate	DATE
extractTime()	extractTimes()	returnedTime	TIME
extractTimestamp()	extractTimestamps()	returnedTimestamp	TIMESTAMP

**Scalar function example:**

In the following example, one value is inserted with the attribute value of key = "1". The value is extracted as an integer and automatically converted to a DECIMAL type.

```
CREATE TABLE t1(key decimal(3,2));
INSERT into t1 values
SELECT * from table(DB2XML.extractInteger(DB2XML.XMLFile
('c:\dxx\samples\xml\getstart.xml'), '/Order/@key="1"'));
SELECT * from t1;
```

In the following example, each key value for the sales order is extracted as an INTEGER

```
SELECT * from table(DB2XML.extractIntegers(DB2XML.XMLFile
('c:\dxx\samples\xml\getstart.xml'), '/Order/@key')) as x;
```

## Updating XML data

With the XML Extender, you can update the entire XML document by replacing the XML column data, or you can update the values of specified elements or attributes.

**Task overview:**

1. Ensure that you have stored the XML document in an XML table and determine what data you want to retrieve.
2. Choose a method for updating the data in the DB2 table (casting functions or UDFs).
3. Specify an SQL query that specifies the XML table and column to update.

**Important:** When updating a column that is enabled for XML, the XML Extender automatically updates the side tables to reflect the changes. Do not update these tables directly without updating the original XML document that is stored in the XML column by changing the corresponding XML element or attribute value. Such updates can cause data inconsistency problems.

#### To update an XML document:

Use one of the following methods:

##### Use a default casting function

For each user-defined type (UDT), a default casting function exists to cast the SQL base type to the UDT. You can use the XML Extender-provided casting functions to update the XML document. Table 14 on page 108 shows the provided casting functions and assumes the column ORDER is created of a different UDT provided by the XML Extender.

**Example:** Updates the XMLVARCHAR type from the cast VARCHAR type, assuming that `xml_buf` is a host variable that is defined as a VARCHAR type.

```
UPDATE sales_tab VALUES('123456', 'Sriram Srinivasan',
 DB2XML.XMLVarchar(:xml_buff))
```

##### Use a storage UDF

For each of the XML Extender UDTs, a storage UDF exists to import data into DB2 from a resource other than its base type. You can use a storage UDF to update the entire XML document by replacing it.

**Example:** The following example updates an XML document using the `XMLVarcharFromFile()` function:

```
UPDATE sales_tab
 set order = XMLVarcharFromFile('dxx_install/samples/cmd/getstart.xml')
WHERE sales_person = 'Sriram Srinivasan'
```

The preceding example updates the XML object from the file named `dxx_install/samples/cmd/getstart.xml` to the column ORDER in the table SALES\_TAB.

See Table 15 on page 109 for a list of the storage functions that the XML Extender provides.

#### To update specific elements and attributes of an XML document:

Use the `Update()` UDF to specify specific changes, rather than updating the entire document. Using the UDF, you specify a location path and the value of the element or attribute represented by the location path to be replaced. (See “Location path” on page 61 for location path syntax.) You do not need to edit the XML document: the XML Extender makes the change for you.

The `Update` UDF updates the entire XML file, and reconstructs the file based on information from the XML parser. See “How the Update function processes the XML document” on page 184 to learn how the `Update` UDF processes the document and for examples documents before and after they are updated.

##### Syntax:

```
Update(xmlobj, path, value)
```

Where:

*xmlobj* Is the name of the XML column for which the value of the element or attribute is to be updated.

*path* Is the location path of the element or attribute that is to be updated. See “Location path” on page 61 for location path syntax. See “Multiple occurrence” on page 186 to learn about considerations for multiple occurrence.

*value* Is the value that is to be updated.

**Example:** The following statement updates the value of the <Customer> element to the character string IBM, using the Update() UDF:

```
UPDATE sales_tab
 set order = Update(order, '/Order/Customer/Name', 'IBM')
WHERE sales_person = 'Sriram Srinivasan'
```

#### **Multiple occurrence:**

When a location path is provided in the Update() UDF, the content of every element or attribute with a matching path is updated with the supplied value. This means that if a document has multiple occurring locations paths, the Update function replaces the existing values with the value provided in the *value* parameter.

---

## **Searching XML documents**

Searching XML data is similar to retrieving XML data: both techniques retrieve data for further manipulation but they search by using the WHERE clause to define predicates as the criteria of retrieval.

The XML Extender provides several methods for searching XML documents in an XML column, depending on your application’s needs. It provides the ability to search document structure and return results based on element content and attribute values. You can search a view of the XML column and its side tables, directly search the side tables for better performance, or use extracting UDFs with WHERE clauses. Additionally, you can use the DB2 Text Extender and search column data within the structural content for a text string.

With the XML Extender you can use indexes on side-table columns, which contain XML element content or attribute values that are extracted from XML documents, for high-speed searching. By specifying the data type of an element or attribute, you can search on SQL general data type or do range searches. For example, in our purchase order example, you could search for all orders that have an extended price of over 2500.00.

Additionally, you can use the DB2 UDB Text Extender to do structural text search or full text search. For example, you could have a column RESUME that contains resumes in XML format. You might want the name of all applicants who have Java skills. You could use the DB2 Text Extender to search on the XML documents for all resumes where the <skill> element contains the character string “JAVA”.

The following sections describe search methods:

- “Searching the XML document by structure” on page 116
- “Using the Text Extender for structural text search” on page 117

## Searching the XML document by structure

Using the XML Extender search features, you can search XML data in a column based on the document structure, that is on elements and attributes. To search the column data you use a SELECT statement in several ways and return a *result set* based on the matches to the document elements and attributes. You can search column data using the following methods:

- Searching with direct query on side tables
- Searching from a *joined view*
- Searching with extracting UDFs
- Searching on elements or attributes with multiple occurrence

These methods are described in the following sections and use examples with the following scenario. The application table SALES\_TAB has an XML column named ORDER. This column has three side tables, ORDER\_SIDE\_TAB, PART\_SIDE\_TAB, and SHIP\_SIDE\_TAB. A default view, sales\_order\_view, was specified when the ORDER column was enabled and joins these tables using the following CREATE VIEW statement:

```
CREATE VIEW sales_order_view(invoice_num, sales_person, order,
 order_key, customer, part_key, price, date)
AS
SELECT sales_tab.invoice_num, sales_tab.sales_person, sales_tab.order,
 order_side_tab.order_key, order_side_tab.customer,
 part_side_tab.part_key, ship_side_tab.date
FROM sales_tab, order_side_tab, part_side_tab, ship_side_tab
WHERE sales_tab.invoice_num = order_side_tab.invoice_num
 AND sales_tab.invoice_num = part_side_tab.invoice_num
 AND sales_tab.invoice_num = ship_side_tab.invoice_num
```

### Searching with direct query on side tables

Direct query with subquery search provides the best performance for structural search when the side tables are indexed. You can use a query or subquery to search side tables correctly.

**Example:** The following statement uses a query and subquery to directly search a side table:

```
SELECT sales_person from sales_tab
WHERE invoice_num in
 (SELECT invoice_num from part_side_tab
 WHERE price > 2500.00)
```

In this example, invoice\_num is the primary key in the SALES\_TAB table.

### Searching from a joined view

You can have the XML Extender create a default view that joins the application table and the side tables using a unique ID. You can use this default view, or any view which joins application table and side tables, to search column data and query the side tables. This method provides a single virtual view of the application table and its side tables. However, the more side tables that are created, the more expensive the query.

**Tip:** You can use the root ID, or DXXROOT\_ID (created by the XML Extender), to join the tables when creating your own view.

**Example:** The following statement searches a view

```
SELECT sales_person from sales_order_view
WHERE price > 2500.00
```

The SQL statement returns the values of sales\_person from the joined view sales\_order\_view table which have line item orders with a price greater than 2500.00.

### Searching with extracting UDFs

You can also use the XML Extender's extracting UDFs to search on elements and attributes, when you have not created indexes or side tables for the application table. Using the extracting UDFs to scan the XML data is very expensive and should only be used with WHERE clauses that restrict the number of XML documents that are included in the search.

**Example:** The following statement searches with an extracting XML Extender UDF:

```
SELECT sales_person from sales_tab
 WHERE extractVarchar(order, '/Order/Customer/Name')
 like '%IBM%'
AND invoice_num > 100
```

In this example, the extracting UDF extracts </Order/Customer/Name> elements with the value of IBM.

### Searching on elements or attributes with multiple occurrence

When searching on elements or attributes that have multiple occurrence, use the DISTINCT clause to prevent duplicate values.

**Example:** The following statement searches with the DISTINCT clause:

```
SELECT sales_person from sales_tab
 WHERE invoice_num in
 (SELECT DISTINCT invoice_num from part_side_tab
 WHERE price > 2500.00)
```

In this example, the DAD file specifies that /Order/Part/Price has multiple occurrence and creates a side table, PART\_SIDE\_TAB, for it. The PART\_SIDE\_TAB table might have more than one row with the same invoice\_num. Using DISTINCT returns only unique values.

## Using the Text Extender for structural text search

When searching the XML document structure, the XML Extender searches element that are converted to general data types, but it does not search text. You can use the DB2 UDB Text Extender for structural or full text search on a column that is enabled for XML. The Text Extender supports XML document search in DB2 UDB version 6.1 or higher. Text Extender is available on Windows operating systems, AIX, and Sun Solaris.

### Structural text search

Searches text strings that are based on the tree structure of the XML document. For example, if you have the document structure of /Order/Customer/Name and you want to search for the character string "IBM" within the <Customer> subelement, you can use a structural text search. The document might also have the string IBM in a <Comment> subelement or as the name of part of a product. A structural text searches only in the specified elements for the string. In this example, only the documents which have IBM in the </Order/Customer/Name> subelement are found; the documents that have IBM in other elements but not in the </Order/Customer/Name> subelement are not returned.

### Full text search

Searches text strings anywhere in the document structure, without regard to

elements or attributes. Using the previous example, all documents that have the string IBM would be returned regardless of where the character string IBM occurs.

To use the Text Extender search, you must install the DB2 Text Extender and enable your database and tables as described below. To learn how to use the Text Extender search, see the chapter on searching with the Text Extender's UDFs in *DB2 Universal Database for OS/390 and z/OS Text Extender Administration and Programming, Version 7*.

### Enabling an XML column for the Text Extender

Assuming that you have an XML-enabled database, use the following steps to enable the Text Extender to search the content of an XML-enabled column. For example purposes, the database is named SALES\_DB, the table is named ORDER, and the XML column names are XVARCHAR and XCLOB:

1. See the `install.txt` file on the Extenders CD to learn how to install the Text Extender.
2. Enter the `txstart` command from one of the following locations:
  - On UNIX operating systems, enter the command from the instance owner's command prompt.
  - On Windows NT, enter the command from the command window where DB2INSTANCE is specified.
3. Open the Text Extender command line window. This step assumes that you have database named SALES\_DB and a table named ORDER, which has two XML columns named XVARCHAR and XCLOB. You might need to run the sample programs in `dxx/samples/c`.
4. Connect to the database. At the DB2TX command prompt, type:  
`'connect to SALES_DB'`
5. Enable the database for the Text Extender.  
From the DB2TX command prompt, type:  
`'enable database'`
6. Enable the columns in the XML table for the Text Extender, defining the data types of the XML document, the language, code pages, and other information about the column.
  - For the VARCHAR column XVARCHAR, type:  
`'enable text column order xvarchar function db2xml.varchartovarchar handle varcharhandle ccsid 850 language us_english format xml indextype precise indexproperty sections_enabled documentmodel (Order) updateindex update'`
  - For the CLOB column XCLOB, type:  
`'enable text column order xclob function db2xml.clob handle clobhandle ccsid 850 language us_english indextype precise updateindex update'`
7. Check the status of the index.
  - For column XVARCHAR, type: `get index status order handle varcharhandle`
  - For column XCLOB, type: `get index status order handle clobhandle`
8. Define the XML document model in a document model INI file called `desmodel.ini`. This file is in: `/db2tx/txins000` for UNIX and `/instance//db2tx/txins000` for Windows NT and sections in an initialization file. For example, for the `textmodel.ini`:

```
;list of document models
[MODELS]
modelname=Order
```



```

; an 'Order' document model definition
; left side = section name identifier
; right side = section name tag

[Order]
Order = /Order
Order/Customer/Name = /Order/Customer/Name
Order/Customer/Email = /Order/Customer/Email
Order/Part/Shipment/ShipMode = /Order/Part/Shipment/ShipMode

```

### Searching for text using the Text Extender

The Text Extender's search capability works well with the XML Extender document structural search. The recommended method is to create a query that searches on the document element or attribute and uses the Text Extender to search the element content or attribute values.

**Example:** The following statements search an XML document text with the Text Extender. At the DB2 command window, type:

```

'connect to SALES_DB'
'select xvarchar from order where db2tx.contains(vvarcharhandle,
'model Order section(Order/Customer/Name) "Motors")=1'
'select xclob from order where db2tx.contains(clobhandle,
'model Order section(Order/Customer/Name) "Motors")=1'

```

The Text Extender Contains() UDF searches.

This example does not contain all of the steps that required to use the Text Extender to search column data. To learn about the Text Extender search concepts and capability, see the chapter on searching with the Text Extender's UDFs in *DB2 Universal Database for OS/390 and z/OS Text Extender Administration and Programming, Version 7*.

## Deleting XML documents

Use the SQL DELETE statement to delete an XML document row from an XML column. You can specify WHERE clauses to refine which documents are to be deleted.

**Example:** The following statements delete all documents that have a value for <ExtendedPrice> greater than 2500.00.

```

DELETE from sales_tab
WHERE invoice_num in
(SELECT invoice_num from part_side_tab
WHERE price > 2500.00)

```

## Limitations when invoking functions from Java database (JDBC)

When using parameter markers in functions, a JDBC restriction requires that the parameter marker for the function must be cast to the data type of the column into which the returned data will be inserted. The function selection logic does not know what data type the argument might turn out to be, and it cannot resolve the reference.

As a result, JDBC cannot resolve the following code:

```

DB2XML.XMLdefault_casting_function(length)

```

You can use the CAST specification to provide a type for the parameter marker, such as VARCHAR, and then the function selection logic can proceed:

```
DB2XML.XMLdefault_casting_function(CAST(? AS cast_type(length))
```

**Example 1:** In the following example, the parameter marker is cast as VARCHAR. The parameter being passed is an XML document, which is cast as VARCHAR(1000) and inserted into the column ORDER.

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
(?,?,DB2XML.XMLVarchar(cast (? as varchar(1000))))";
```

**Example 2:** In the following example, the parameter marker is cast as VARCHAR. The parameter being passed is a file name and its contents are converted to VARCHAR and inserted into the column ORDER.

```
String query = "insert into sales_tab(invoice_num, sales_person, order) values
(?,?,DB2XML.XMLVarcharfromFILE(cast (? as varchar(1000))))";
```

---

## Chapter 9. Managing XML collection data

An XML collection is a set of relational tables that contain data that is mapped to XML documents. This access and storage method lets you compose an XML document from existing data, decompose an XML document, and use XML as an interchange method.

The relational tables can be new tables that the XML Extender generates when decomposing XML documents, or existing tables that have data that is to be used with the XML Extender to generate XML documents for your applications. Column data in these tables does not contain XML tags; it contains the content and values that are associated with elements and attributes, respectively. Stored procedures act as the access and storage method for storing, retrieving, updating, searching, and deleting XML collection data.

The parameter limits used by the XML collection stored procedures are documented in “Appendix D. The XML Extender limits” on page 263.

You can increase the CLOB sizes for the stored procedures results as documented in “Increasing the CLOB limit” on page 190.

See the following sections for information on managing your XML collection:

- “Composing XML documents from DB2 data”
- “Decomposing XML documents into DB2 data” on page 130

---

### Composing XML documents from DB2 data

Composition is the generation of a set of XML documents from relational data in an XML collection. You can compose XML documents using stored procedures. To use these stored procedures, you must create a DAD file, which specifies the mapping between the XML document and the DB2 table structure. The stored procedures use the DAD file to compose the XML document. See “Planning for XML collections” on page 52 to learn how to create a DAD file.

#### Before you begin

- Map the structure of the XML document to the relational tables that contain the contents of the element and attribute values.
- Select a mapping method: SQL mapping or RDB\_node mapping.
- Prepare the DAD file. See “Planning for XML collections” on page 52 for complete details.
- Optionally, enable the XML collection.

#### Composing the XML document

The XML Extender provides two stored procedures, `dxxGenXML()` and `dxxRetrieveXML()`, to compose XML documents.

##### **dxxGenXML()**

This stored procedure is used for applications that do occasional updates or for applications that do not want the overhead of administering the XML data. The stored procedure `dxxGenXML()` does not require an enabled collection; it uses a DAD file instead.

The stored procedure `dxxGenXML()` constructs XML documents using data that is stored in XML collection tables, which are specified by the `<Xcollection>` element in the DAD file. This stored procedure inserts each XML document as a row into a *result table*. You can also open a cursor on the result table and fetch the result set. The result table should be created by the application and always has at least one column of VARCHAR, CLOB, XMLVARCHAR, or XMLCLOB type.

Additionally, if you specify the validation element in the DAD as YES, the application must also create a validation column of type INTEGER in the result table. You can specify any name for the validate column as long as its data type is integer. The default column value for integer is 0. You do not have to set the value. XML Extender will set the value to 1 if the document is valid, otherwise it is 0

The stored procedure `dxxGenXML()` also allows you to specify the maximum number of rows that are to be generated in the result table. This shortens processing time. The stored procedure returns the actual number of rows in the table, along with any return codes and messages.

The corresponding stored procedure for decomposition is `dxxShredXML()`; it also takes the DAD as the input parameter and does not require that the XML collection be enabled.

## To compose an XML collection: dxxGenXML()

Embed a stored procedure call in your application using the following stored procedure declaration:

```
dxxGenXML(CLOB(100K) DAD, /* input */
 char(32) resultTabName, /* input */
 char(30) result_column, /* input */
 char(30) valid_column, /* input */
 integer overrideType, /* input */
 varchar(1024) override, /* input */
 integer maxRows, /* input */
 integer numRows, /* output */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

See “dxxGenXML()” on page 199 for the full syntax and examples.

**Example:** The following example composes an XML document:

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE is CLOB(100K) dad; /* DAD */
EXEC SQL DECLARE :dad VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running from USS */
/* to ensure that DB2 converts the code page correctly*/

char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
char valid_colname[32]; /* name of the valid column, will set to NULL */
char override[2]; /* override, will set to NULL*/
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short dad_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &dad.data,
 1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf("Error reading dad file
 /dxx/samples/dad/getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
}
```

```

 dad.length = file_length;
 }
 else {
 printf("Error opening dad file \n",);
 rc = -1;
 goto exit;
 }
 /* initialize host variable and indicators */
 strcpy(result_tab,"xml_order_tab");
 strcpy(result_colname, "xmlorder")
 valid_colname = '\0';
 override[0] = '\0';
 overrideType = NO_OVERRIDE;
 max_row = 500;
 num_row = 0;
 returnCode = 0;
 msg_txt[0] = '\0';
 dad_ind = 0;
 rtab_ind = 0;
 rcol_ind = 0;
 vcol_ind = -1;
 ov_ind = -1;
 ovtype_ind = 0;
 maxrow_ind = 0;
 numrow_ind = -1;
 returnCode_ind = -1;
 returnMsg_ind = -1;

 /* Call the store procedure */
 EXEC SQL CALL "DB2XML.DXXGENXML" (:dad:dad_ind;
 :result_tab:rtab_ind,
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind,:override:ov_ind,
 :max_row:maxrow_ind,:num_row:numrow_ind,
 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

 if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
 }
 else
 EXEC SQL COMMIT;
}

exit:
 return rc;

```

The result table after the stored procedure is called contains 250 rows because the SQL query specified in the DAD file generated 250 XML documents.

### **dxxRetrieveXML()**

This stored procedure is used for applications that make regular updates. Because the same tasks are repeated, improved performance is important. Enabling an XML collection and using the collection name in the stored procedure improves performance.

The stored procedure `dxxRetrieveXML()` works the same as the stored procedure `dxxGenXML()`, except that it takes the name of an enabled XML collection instead of a DAD file. When an XML collection is enabled, a DAD file is stored in the `XML_USAGE` table. Therefore, the XML Extender retrieves the DAD file and, from this point forward, `dxxRetrieveXML()` is the same as `dxxGenXML()`.

dxxRetrieveXML() allows the same DAD file to be used for both composition and decomposition. This stored procedure also can be used for retrieving decomposed XML documents.

The corresponding stored procedure for decomposition is dxxInsertXML(); it also takes the name of an enabled XML collection.

### To compose an XML collection: dxxRetrieveXML()

Embed a stored procedure call in your application using the following stored procedure declaration:

```
dxxRetrieveXML(char(32) collectionName, /* input */
 char(32) resultTabName, /* input */
 char(30) result_column, /* input */
 char(30) valid_column, /* input */
 integer overrideType, /* input */
 varchar(1024) override, /* input */
 integer maxRows, /* input */
 integer numRows, /* output */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

See “dxxRetrieveXML()” on page 203 for full syntax and examples.

**Example:** The following example is of a call to dxxRetrieveXML(). It assumes that a result table is created with the name of XML\_ORDER\_TAB and it has one column of XMLVARCHAR type.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
char valid_colname[32]; /* name of the valid column, will set to NULL*/
char override[2]; /* override, will set to NULL*/
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab, "xml_order_tab");
strcpy(result_col, "xmlorder");
valid_colname[0] = '\0';
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
```

```

num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
 :result_tab:rtab_ind,
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind,:override:ov_ind,
 :max_row:maxrow_ind,:num_row:numrow_ind,
 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
else
 EXEC SQL COMMIT;
}

```

## Dynamically overriding values in the DAD file

For dynamic queries you can use two optional parameters to override conditions in the DAD file: *override* and *overrideType*. Based on the input from *overrideType*, the application can override the <SQL\_stmt> tag values for SQL mapping or the conditions in RDB\_nodes for RDB\_node mapping in the DAD.

These parameters have the following values and rules:

### *overrideType*

This parameter is a required input parameter (IN) that flags the type of the *override* parameter. *overrideType* has the following values:

#### **NO\_OVERRIDE**

Specifies not to override a condition in the DAD file.

#### **SQL\_OVERRIDE**

Specifies to override a condition in DAD file with an SQL statement.

#### **XML\_OVERRIDE**

Specifies to override a condition in the DAD file with an XPath-based condition.

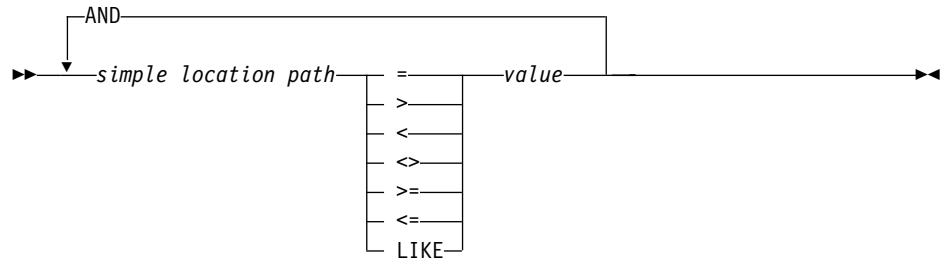
### *override*

This parameter is an optional input parameter (IN) that specifies the override condition for the DAD file. The input value syntax corresponds to the value specified on the *overrideType*.

- If you specify NO\_OVERRIDE, the input value is a NULL string.
- If you specify SQL\_OVERRIDE, the input value is a valid SQL statement.  
**Required:** If you use SQL\_OVERRIDE and an SQL statement, you must use the SQL mapping scheme in the DAD file. The input SQL statement overrides the SQL statement specified by the <SQL\_stmt> element in the DAD file.



- If you use XML\_OVERRIDE, the input value is a string which contains one or more expressions. **Required:** If you use XML\_OVERRIDE and an expression, you must use the RDB\_node mapping scheme in the DAD file. The input XML expression overrides the RDB\_node condition specified in the DAD file. The expression uses the following syntax:



Where:

*simple location path*

A simple location path using syntax defined by XPath; see Table 10 on page 62 for syntax.

#### operators

=, >, <, <>, >=, <=, and LIKE. Can have a space to separate the operator from the other parts of the expression.

*value*

A numeric value or a single quoted string.

You can have optional spaces around the operations; spaces are mandatory around the LIKE operator.

When the XML\_OVERRIDE value is specified, the condition for the RDB\_node in the text\_node or attribute\_node that matches the simple location path is overridden by the specified expression.

XML\_OVERRIDE is not completely XPath compliant. The simple location path is only used to identify the element or attribute that is mapped to a column.

#### Examples:

The following examples show dynamic override using SQL\_OVERRIDE and XML\_OVERRIDE. Most stored procedure examples in this book use NO\_OVERRIDE.

#### Example 1: A stored procedure using SQL\_OVERRIDE.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
char valid_colname[32]; /* name of the valid column, will set to NULL*/
char override[512]; /* override */
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
```

```

short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;
float price_value;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab, "xml_order_tab");
strcpy(result_col, "xmlorder");
valid_colname[0] = '\0';

/* get the price_value from some place, such as from data */
price_value = 1000.00 /* for example */

/* specify the override */
sprintf(override,
 " SELECT o.order_key, customer, p.part_key, quantity, price,
 tax, ship_id, date, mode
 FROM order_tab o, part_tab p,
 table(select db2xml.generate_unique()
 as ship_id, date, mode from ship_tab) s
 WHERE p.price > %d and s.date >'1996-06_01' AND
 p.order_key = o.order_key and s.part_key = p.part_key",
 price_value);

overrideType = SQL_OVERRIDE;max_row = 0;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
ov_ind = 0;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
 :result_tab:rtab_ind,
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind, :override:ov_ind,
 :returnCode:returnCode_ind, :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
 else
 EXEC SQL COMMIT;
}

```

In this example, the <xcollection> element in the DAD file must have an <SQL\_stmt> element. The *override* parameter overrides the value of <SQL\_stmt>, by changing the price to be greater than 50.00, and the date is changed to be greater than 1998-12-01.

**Example 2:** A stored procedure using XML\_OVERRIDE.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
char valid_colname[32]; /* name of the valid column, will set to NULL*/
char override[256]; /* override, SQL_stmt*/
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab, "xml_order_tab");
strcpy(result_col, "xmlorder");
valid_colname[0] = '\0';
sprintf(override, "%s %s",
 "/Order/Part Price > 50.00 AND ",
 "/Order/Part/Shipment/ShipDate > '1998-12-01'");
overrideType = XML_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
ov_ind = 0;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
 :result_tab:rtab_ind,
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind, :override:ov_ind,
```

```

: max_row: maxrow_ind, : num_row: numrow_ind,
: returnCode: returnCode_ind, : returnMsg: returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
} else
 EXEC SQL COMMIT;
}

```

In this example, the <collection> element in the DAD file has an RDB\_node for the root element\_node. The *override* value is XML-content based. The XML Extender converts the simple location path to the mapped DB2 column.

---

## Decomposing XML documents into DB2 data

To decompose an XML document is to break down the data inside of an XML document and store it in relational tables. The XML Extender provides stored procedures to decompose XML data from source XML documents into relational tables. To use these stored procedures, you must create a DAD file, which specifies the mapping between the XML document and DB2 table structure. The stored procedures use the DAD file to decompose the XML document. See “Planning for XML collections” on page 52 to learn how to create a DAD file.

### Enabling an XML collection for decomposition

In most cases, you need to enable an XML collection before using the stored procedures. In the following cases, you are required to enable an XML collection:

- When decomposing XML documents into new tables, an XML collection must be enabled because all tables in the XML collection are created by the XML Extender when the collection is enabled.
- When keeping the sequence of elements and attributes that have multiple occurrence is important. The XML Extender only preserves the sequence order of elements or attributes of multiple occurrence for tables that are created during enablement of a collection. When decomposing XML documents into existing relational tables, the sequence order is not guaranteed to be preserved.

If you want to pass the DAD file spontaneously when the tables already exist in your database, you do not need to enable an XML collection.

### Decomposition table size limits

Decomposition uses RDB\_node mapping to specify how an XML document is decomposed into DB2 tables by extracting the element and attribute values and storing them in table rows. The values from each XML document are stored in one or more DB2 tables. Each table can have a maximum of 1024 rows decomposed from each document.

For example, if an XML document is decomposed into five tables, each of the five tables can have up to 1024 rows for that particular document. If the table has rows for multiple documents, it can have up to 1024 rows for each document. If the table has 20 documents, it can have 20,480 rows, 1024 for each document.

Using multiple-occurring elements (elements with location paths that can occur more than once in the XML structure) affects the number of rows. For example, a document that contains an element <Part> that occurs 20 times, might be decomposed as 20 rows in a table. When using multiple occurring elements, consider that a maximum of 1024 rows can be decomposed into one table from a single document.

## Before you begin

- Map the structure of the XML document to the relational tables that contain the contents of the elements and attributes values.
- Prepare the DAD file, using RDB\_node mapping. See “Planning for XML collections” on page 52 for details.
- Optionally, enable the XML collection.

## Decomposing the XML document

The XML Extender provides two stored procedures, `dxxShredXML()` and `dxxInsertXML`, to decompose XML documents.

### `dxxShredXML()`

This stored procedure is used for applications that do occasional updates or for applications that do not want the overhead of administering the XML data. The stored procedure `dxxShredXML()` does not require an enabled collection; it uses a DAD file instead.

The stored procedure `dxxShredXML()` takes two input parameters, a DAD file and the XML document that is to be decomposed; it returns two output parameters: a return code and a return message.

The stored procedure `dxxShredXML()` inserts an XML document into an XML collection according to the `<Xcollection>` specification in the input DAD file. The tables that are used in the `<Xcollection>` of the DAD file are assumed to exist, and the columns are assumed to meet the data types specified in the DAD mapping. If this is not true, an error message is returned. The stored procedure `dxxShredXML()` then decomposes the XML document, and it inserts untagged XML data into the tables specified in the DAD file.

The corresponding stored procedure for composition is `dxxGenXML()`; it also takes the DAD as the input parameter and does not require that the XML collection be enabled.

### To decompose an XML collection: `dxxShredXML()`

Embed a stored procedure call in your application using the following stored procedure declaration:

```
dxxShredXML(CLOB(100K) DAD, /* input */
 CLOB(1M) xmlobj, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

See “`dxxShredXML()`” on page 207 for the full syntax and examples.

**Example:** The following is an example of a call to `dxxShredXML()`:

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS CLOB(100K) dad; /* DAD */
EXEC SQL DECLARE :dad VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running */
/* from USS to ensure that DB2 converts the */
/* code page correctly */
SQL TYPE IS CLOB(100K) xmlDoc; /* input xml document */
EXEC SQL DECLARE :xmlDoc VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running */
/* from USS to ensure that DB2 converts the */
```

```

/* code page correctly */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short dad_ind;
short xmlDoc_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &dad.data;, 1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf("Error reading dad file getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
 dad.length = file_length;
}
else {
 printf("Error opening dad file \n");
 rc = -1;
 goto exit;
}

/* Initialize the XML CLOB object. */
file_handle = fopen("/dxx/samples/xml/getstart_xcollection.xml", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &xmlDoc.data;, 1, FILE_SIZE,
 file_handle);

 if (file_length == 0) {
 printf("Error reading xml file getstart_xcollection.xml \n");
 rc = -1;
 goto exit;
 } else
 xmlDoc.length = file_length;
}
else {
 printf("Error opening xml file \n");
 rc = -1;
 goto exit;
}

/* initialize host variable and indicators */
returnCode = 0;
msg_txt[0] = '\0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED" (:dad:dad_ind;
 :xmlDoc:xmlDoc_ind,
 :returnCode:returnCode_ind,
 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
}
else
 EXEC SQL COMMIT;

```

```

}
exit:
 return rc;

```

### **dxxInsertXML()**

This stored procedure is used for applications that make regular updates. Because the same tasks are repeated, improved performance is important. Enabling an XML collection and using the collection name in the stored procedure improves performance. The stored procedure `dxxInsertXML()` works the same as `dxxShredXML()`, except that `dxxInsertXML()` takes an enabled XML collection as its first input parameter.

The stored procedure `dxxInsertXML()` inserts an XML document into an enabled XML collection, which is associated with a DAD file. The DAD file contains specifications for the collection tables and the mapping. The collection tables are checked or created according to the specifications in the `<Xcollection>`. The stored procedure `dxxInsertXML()` then decomposes the XML document according to the mapping, and it inserts untagged XML data into the tables of the named XML collection.

The corresponding stored procedure for composition is `dxxRetrieveXML()`; it also takes the name of an enabled XML collection.

### **To decompose an XML collection: dxxInsertXML()**

Embed a stored procedure call in your application using the following stored procedure declaration:

```

dxxInsertXML(char(collectionName32) collectionName, /* input */
 CLOB(1M) xmlObj, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */

```

See “`dxxInsertXML()`” on page 210 for the full syntax and examples.

**Example:** The following is an example of a call to `dxxInsertXML()`:

```

#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short xmlDoc_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &dad.data;, 1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf ("Error reading dad file getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
 dad.length = file_length;
}

```

```

}
else {
 printf("Error opening dad file \n");
 rc = -1;
 goto exit;
}

/* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXINSERTXML" (:collection_name:collection_name_ind,
 :xmlDoc:xmlDoc_ind,
 :returnCode:returnCode_ind,
 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
}
else
 EXEC SQL COMMIT;
}

exit:
return rc;

```

---

## Accessing an XML collection

You can update, delete, search, and retrieve XML collections. Remember, however, that the purpose of using an XML collection is to store or retrieve untagged, pure data in database tables. The data in existing database tables has nothing to do with any incoming XML documents; update, delete, and search operations literally consist of normal SQL access to these tables. If the data is decomposed from incoming XML documents, no original XML documents continue to exist.

The XML Extender provides the ability to perform operations on the data from an XML collection view. Using UPDATE and DELETE SQL statements, you can modify the data that is used for composing XML documents, and therefore, update the XML collection.

### Considerations:

- To update a document, do not delete a row containing the primary key of the table, which is the foreign key row of the other collection tables. When the primary key and foreign key row is deleted, the document is deleted.
- To replace or delete elements and attribute values, you can delete and insert rows in lower-level tables without deleting the document.
- To delete a document, delete the row which composes the top element\_node specified in the DAD.

## Updating data in an XML collection

The XML Extender allows you to update untagged data that is stored in XML collection tables. By updating XML collection table values, you are updating the text of an XML element, or the value of an XML attribute. Additionally, updates can delete an instance of data from multiple-occurring elements or attributes.



From an SQL point of view, changing the value of the element or attribute is an update operation, and deleting an instance of an element or attribute is a delete operation. From an XML point of view, as long as the element text or attribute value of the root element\_node exists, the XML document still exists and is, therefore, an update operation.

**Requirements:** To update data in an XML collection, observe the following rules.

- Specify the primary-foreign key relationship among the collection tables when the existing tables have this relationship. If they do not, ensure that there are columns that can be joined.
- Include the join condition that is specified in the DAD file:
  - For SQL mapping, in the <SQL\_stmt> element
  - For RDB\_node mapping, in the RDB\_node of the root element node

### Updating element and attribute values

In an XML collection, element text and attribute value are all mapped to columns in database tables. Regardless of whether the column data previously exists or is decomposed from incoming XML documents, you replace the data using the normal SQL update technique.

To update an element or attribute value, specify a WHERE clause in the SQL UPDATE statement that contains the join condition that is specified in the DAD file.

For example:

```
UPDATE SHIP_TAB
 set MODE = 'BOAT'
WHERE MODE='AIR' AND PART_KEY in
 (SELECT PART_KEY from PART_TAB WHERE ORDER_KEY=68)
```

The <ShipMode> element value is updated from AIR to BOAT in the SHIP\_TAB table, where the key is 68.

### Deleting element and attribute instances

To update composed XML documents by eliminating multiple-occurring elements or attributes, delete a row containing the field value that corresponds to the element or attribute value, using the WHERE clause. As long as you do not delete the row that contains the values for the top element\_node, deleting element values is considered an update of the XML document.

For example, in the following DELETE statement, you are deleting a <shipment> element by specifying a unique value of one of its subelements.

```
DELETE from SHIP_TAB
 WHERE DATE='1999-04-12'
```

Specifying a DATE value deletes the row that matches this value. The composed document originally contained two <shipment> elements, but now contains one.

## Deleting an XML document from an XML collection

You can delete an XML document that is composed from a collection. This means that if you have an XML collection that composes multiple XML documents, you can delete one of these composed documents.

To delete the document, you delete a row in the table that composes the top element\_node that is specified in the DAD file. This table contains the primary key for the top-level collection table and the foreign key for the lower-level tables.

For example, the following DELETE statement specifies the value of the primary key column.

```
DELETE from order_tab
WHERE order_key=1
```

ORDER\_KEY is the primary key in the table ORDER\_TAB and is the top element\_node when the XML document is composed. Deleting this row deletes one XML document that is generated during composition. Therefore, from the XML point of view, one XML document is deleted from the XML collection.

## Retrieving XML documents from an XML collection

Retrieving XML documents from an XML collection is similar to composing documents from the collection.

To retrieve XML documents, use the stored procedure, dxxRetrieveXML(). See “dxxRetrieveXML()” on page 203 for syntax and examples.

**DAD file consideration:** When you decompose XML documents in an XML collection, you can lose the order of multiple-occurring elements and attribute values, unless you specify the order in the DAD file. To preserve this order, you should use the RDB\_node mapping scheme. This mapping scheme allows you to specify an orderBy attribute for the table containing the root element in its RDB\_node.

---

## Searching an XML collection

This section describes searching an XML collection in terms of the following goals:

- **Generating XML documents using search criteria:**

This task is actually composition using a condition. You can specify the search criteria using the following search criteria:

- Specify the condition in the text\_node and attribute\_node of the DAD file
- Specify the *overwrite* parameter when using the dxxGenXML() and dxxRetrieveXML() stored procedures.

For example, if you enabled an XML collection, sales\_ord, using the DAD file, order.dad, but you now want to override the price using form data derived from the Web, you can override the value of the <SQL\_stmt> DAD element, as follows:

```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL END DECLARE SECTION;

float price_value;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(result_tab,"xml_order_tab");
overrideType = SQL_OVERRIDE;
max_row = 20;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
override_ind = 0;
```

```

overrideType_ind = 0;
rtab_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* get the price_value from some place, such as form data */
price_value = 1000.00 /* for example*/

/* specify the overwrite */
sprintf(overwrite,
 "SELECT o.order_key, customer, p.part_key, quantity, price,
 tax, ship_id, date, mode
 FROM order_tab o, part_tab p,
 table
(select db2xml.generate_unique()
 as ship_id, date, mode from ship_tab) s
 WHERE p.price > %d and s.date >'1996-06-01' AND
 p.order_key = o.order_key and s.part_key = p.part_key",
 price_value);

/* Call the store procedure */
EXEC SQL CALL DB2XML.dxxRetrieve(:collection:collection_ind,
 :result_tab:rtab_ind,
 :overrideType:overrideType_ind,:overwrite:overwrite_ind,
 :max_row:maxrow_ind,:num_row:numrow_ind,
 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

```

The condition of price > 2500.00 in order.dad is overridden by price > ?, where ? is based on the input variable *price\_value*.

- **Searching for decomposed XML data:**

You can use normal SQL query operations to search collection tables. You can join collection tables, or use subqueries, and then do structural-text search on text columns. With the results of the structural search, you can apply that data to retrieve or generate the specified XML document.



---

## Part 4. Reference

This part provides syntax information for the XML Extender UDTs, UDFs, and stored procedures. Message text is also provided for problem determination activities.



---

## Chapter 10. XML Extender administration command: DXXADM

You perform the following XML Extender administration tasks by calling DXXADM, using various subcommands:

- Enabling or disabling the database server for the XML Extender
- Enabling or disabling an XML column
- Enabling or disabling an XML collection

You can also use the XML Extender administration wizard or stored procedures to perform each of the administration tasks.

---

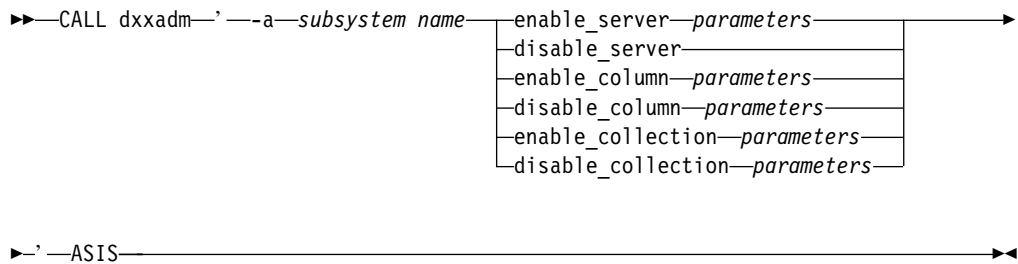
## High-level syntax

### Purpose

To perform XML Extender administration tasks you call DXXADM with the subcommand and parameters for each task.

The following syntax diagram provides the high-level syntax for calling DXXADM. Descriptions of each subcommand and their parameters are provided in the following sections.

### Format



### Parameters

Table 19. *dxxadm* parameters

Parameter	Description
<i>subsystem name</i>	The name of the DB2 subsystem to which the application attaches.

The call assumes you have the XML Extender load module library activated. If you do not, use the fully qualified for DXXADM.

---

## Administration subcommands

The following sections describe each of the DXXADM subcommands that are available to system programmers:

- “enable\_server” on page 143
- “disable\_server” on page 145
- “enable\_column” on page 146
- “disable\_column” on page 147
- “enable\_collection” on page 148
- “disable\_collection” on page 149



## enable\_server

### Purpose

Connects to and enables a database server so that it can be used with the XML Extender. When the database server is enabled, the XML Extender creates the following objects:

- The XML Extender user-defined types (UDTs).
- The XML Extender user-defined functions (UDFs).
- The XML Extender stored procedures
- The XML Extender DTD reference table, DTD\_REF, which stores DTDs and information about each DTD. For a complete description of the DTD\_REF table, see “DTD reference table” on page 213.
- The XML Extender usage table, XML\_USAGE, which stores common information for each column that is enabled for XML and for each collection. For a complete description of the XML\_USAGE table, see “XML usage table” on page 213.

### Format

```
▶—dxxadm—enable_server—-a—subsystem_name—┐
└──┘
└──────────security──security_level──┘

▶—using—tablespace_DTD_REF, tablespace_XML_USAGE—▶

▶—WLM environment—WLM_name1—┐
└────────────────────────────────┘
└──────────WLM_name2──┘
```

### Parameters

Table 20. enable\_server parameters

Parameter	Description
-a <i>subsystem_name</i>	The name of the DB2 subsystem.
<i>security_level</i>	Determines the user ID that is authorized to access external resources when running stored procedures. Choices are DB2, USER, DEFINER. DB2 is the default. See <i>DB2 Universal Database for OS/390 and z/OS SQL Reference, Version 7</i> for more information.
<i>tablespace_DTD_REF</i>	The name of the table space in which the CLOB column, CONTENT, of the DTD_REF table, is stored. This table space is either created by the DB2 administrator manually or when the DXXGPREP job file is run during initialization. The default is XMLLOBTS.
<i>tablespace_XML_USAGE</i>	The name of table space in which the CLOB column, DAD, of the XML_USAGE table, is stored. This table space is either created by the DB2 administrator manually or when the DXXGPREP job file is run during initialization. The default is XMLLOBT2.

Table 20. *enable\_server* parameters (continued)

Parameter	Description
<i>WLM name</i>	The names of the WLM environments. At least one name is required. If one is specified, the name is for all stored procedures and UDFs. If two are specified, the first name is for the stored procedures, the second name is for the UDFs.

### Examples

The following example enables the database server for XML Extender using the SUBSYS1 subsystem, the table spaces TBSPC1 and TBSPC2, and the WLM environment ENVIR233:

```
dxxadm 'enable_server -a SUBSYS1 using tbspc1,tbspc2 wlm environment envir233' ASIS
```

## disable\_server

### Purpose

Connects to and disables the XML-enabled database server. When the database server is disabled, it can no longer be used by the XML Extender. When the XML Extender disables the database server, it drops the following objects:

- The XML Extender user-defined types (UDTs).
- The XML Extender user-defined functions (UDFs).
- The XML Extender DTD reference table, DTD\_REF, which stores DTDs and information about each DTD. For a complete description of the DTD\_REF table, see “DTD reference table” on page 213.
- The XML Extender usage table, XML\_USAGE, which stores common information for each column that is enabled for XML and for each collection. For a complete description of the XML\_USAGE table, see “XML usage table” on page 213.

**Important:** You must disable all XML columns before attempting to disable a database server. The XML Extender cannot disable a database server that contains columns or collections that are enabled for XML.

### Format

►►—disable\_server—-a—*subsystem\_name*—◄◄

### Parameters

Table 21. *disable\_server* parameters

Parameter	Description
-a <i>subsystem_name</i>	The name of the DB2 subsystem.

### Examples

The following example disables the database server:

```
dxxadm disable_server -a SUBSYS1
```

## enable\_column

### Purpose

Connects to a database server and enables an XML column so that it can contain the XML Extender UDTs. When enabling a column, the XML Extender completes the following tasks:

- Determines whether the XML table has a primary key; if not, the XML Extender alters the XML table and adds a column called DXXROOT\_ID.
- Creates side tables that are specified in the DAD file with a column containing a unique identifier for each row in the XML table. This column is either the root ID that the user specified or the DXXROOT\_ID that was named by the XML Extender.
- Creates a default view for the XML table and its side tables, optionally using a name you specify.

### Format

```
▶▶ dxxadm enable_column -- a subsystem_name →
▶ tab_name column_name DAD_file →
 └─t tablespace┘ └─v default_view┘
▶ └─r root_id┘ →▶
```

### Parameters

Table 22. enable\_column parameters

Parameter	Description
- a subsystem_name	The name of the DB2 subsystem.
tab_name	The name of the table in which the XML column resides.
column_name	The name of the XML column.
DAD_file	The name of the DAD file that maps the XML document to the XML column and side tables.
-t tablespace	The table space, which is optional and contains the side tables associated with the XML column. If not specified, the default table space is used.
-v default_view	The name of the default view, which is optional, that joins the XML column and side tables.
-r root_id	The name of the primary key in the XML column table that is to be used as the root_id for side tables. The root_id is optional.

### Examples

The following example enables an XML column.

```
dxxadm enable_column -a SUBSYS1 SALES_TAB ORDER -v SALODVW -r INVOICE_NUMBER
```

## disable\_column

### Purpose

Connects to a database and disables the XML-enabled column. When the column is disabled, it can no longer contain XML data types. When an XML-enabled column is disabled, the following actions are performed:

- The XML column usage entry is deleted from the XML\_USAGE table.
- The USAGE\_COUNT is decremented in the DTD\_REF table.
- All triggers that are associated with this column are dropped.
- All side tables that are associated with this column are dropped.

**Important:** You must disable an XML column before dropping an XML table. If an XML table is dropped but its XML column is not disabled, the XML Extender keeps both the side tables it created and the XML column entry in the XML\_USAGE table.

### Format

```
►►—dxxadm—disable_column—-a—subsystem_name—tab_name—column_name—◄◄
```

### Parameters

Table 23. *disable\_column* parameters

Parameter	Description
-a <i>subsystem_name</i>	The name of the DB2 subsystem.
<i>tab_name</i>	The name of the table in which the XML column resides.
<i>column_name</i>	The name of the XML column.

### Examples

The following example disables an XML-enabled column.

```
dxxadm disable_column -a SUBSYS1 SALES_TAB ORDER
```

## enable\_collection

### Purpose

Connects to a database server and enables an XML collection according to the specified DAD. When enabling a collection, the XML Extender does the following tasks:

- Creates an XML collection usage entry in the XML\_USAGE table.
- For RDB\_node mapping, creates collection tables specified in the DAD if the tables do not exist in the database.

### Format

```
▶▶ enable_collection -a subsystem_name collection_name DAD_file ▶▶

▶▶ ┌-t tablespace ─┐ ▶▶
```

### Parameters

Table 24. enable\_collection parameters

Parameter	Description
-a <i>subsystem_name</i>	The name of the DB2 subsystem.
-t <i>tablespace</i>	The name of the table space, which is optional and associated with the collection. If not specified, the default table space is used.
<i>collection_name</i>	The name of the XML collection.
<i>DAD_file</i>	The name of the DAD file that maps the XML document to the relational tables in the collection.

### Examples

The following example enables an XML collection named SALES\_ORD with the GETSTART\_XCOLLECTION.DAD:

```
dxadm enable_collection -a SUBSYS1 using ORDRPSC SALES_ORD
 'ORDPRJ.WORK.DAD(GETSTART_XCOLLECTION)'
```

## disable\_collection

### Purpose

Connects to a database and disables an XML-enabled collection. The collection name can no longer be used in the composition (dxxRetrieveXML) and decomposition (dxxInsertXML) stored procedures. When an XML collection is disabled, the associated collection entry is deleted from the XML\_USAGE table. Note that disabling the collection does not drop the collection tables that are created during the enable\_collection step.

### Format

```
►►—dxxadm—disable_collection—-a—subsystem_name—collection_name—◀◀
```

### Parameters

Table 25. *disable\_collection* parameters

Parameter	Description
-a <i>subsystem_name</i>	The name of the DB2 subsystem.
<i>collection_name</i>	The name of the XML collection.

### Examples

The following example disables an XML collection named SALES\_ORD.

```
dxxadm disable_collection -a SUBSYS1 SALES_ORD
```





---

## Chapter 11. XML Extender user-defined types

The XML Extender user-defined types (UDTs) are data types that are used for XML columns and XML collections. All the UDTs have the schema name DB2XML. The XML Extender creates UDTs for storing and retrieving XML documents. Table 26 contains an overview of the UDTs.

Table 26. The XML Extender UDTs

User-defined type column	Source data type	Usage description
XMLVARCHAR	VARCHAR( <i>varchar_len</i> )	Stores an entire XML document as VARCHAR inside DB2.
XMLCLOB	CLOB( <i>clob_len</i> )	Stores an entire XML document as character large object (CLOB) inside DB2.
XMLFILE	VARCHAR(512)	Specifies the file name of the local file server. If XMLFILE is specified for the XML column, then the XML Extender stores the XML document in an external server file. The Text Extender cannot be enabled with XMLFILE. It is your responsibility to ensure integrity between the file content and DB2, as well as the side table created for indexing.

Where *varchar\_len* and *clob\_len* are specific to the operating system.

For DB2 UDB, *varchar\_len* = 3K and *clob\_len* = 2G.

These UDTs are used only to specify the types of application columns; they do not apply to the side tables that the XML Extender creates.



## Chapter 12. XML Extender user-defined functions

The XML Extender provides functions for storing, retrieving, searching, and updating XML documents, and for extracting XML elements or attributes. Use XML user-defined functions (UDFs) for XML columns, but not for XML collections. All the UDFs have the schema name DB2XML, which can be omitted in front of UDFs.

The four types of XML Extender functions are: storage functions, retrieval functions, extracting functions, and an update function.

### storage functions

Storage functions insert XML documents into a DB2 database. For syntax and examples, see “Storage functions” on page 154.

### retrieval functions

Retrieval functions retrieve XML documents from XML columns in a DB2 database. For syntax and examples, see “Retrieval functions” on page 158.

### extracting functions

Extracting functions extract and convert the element content or attribute value from an XML document to the data type that is specified by the function name. The XML Extender provides a set of extracting functions for various SQL data types. For syntax and examples, see “Extracting functions” on page 163.

### update function

The Update() function modifies the element content or attribute value and returns a copy of an XML document with an updated value that is specified by the location path. The Update() function allows the application programmer to specify the element or attribute that is to be updated. For syntax and examples, see “Update function” on page 183.

### generate\_unique function

The generate\_unique() function returns a unique key. For syntax and examples, see “Generate unique function” on page 188.

Table 27 provides a summary of the XML Extender functions.

Table 27. The XML Extender user-defined functions

Type	Function
Storage functions	“XMLVarcharFromFile()” on page 155
	“XMLCLOBFromFile()” on page 156
	“XMLFileFromVarchar()” on page 157
	“XMLFileFromCLOB()” on page 158
Retrieval functions	“Content(): retrieve from XMLFILE to a CLOB” on page 160
	“Content(): retrieve from XMLVARCHAR to an external server file” on page 161
	“Content(): retrieval from XMLCLOB to an external server file” on page 162

Table 27. The XML Extender user-defined functions (continued)

Type	Function
Extracting functions	"extractInteger() and extractIntegers()" on page 164
	"extractSmallint() and extractSmallints()" on page 165
	"extractDouble() and extractDoubles()" on page 167
	"extractReal() and extractReals()" on page 169
	"extractChar()and extractChars()" on page 171
	"extractVarchar() and extractVarchars()" on page 173
	"extractCLOB() and extractCLOBs()" on page 175
	"extractDate() and extractDates()" on page 177
	"extractTime() and extractTimes()" on page 179
	"extractTimestamp() and extractTimestamps()" on page 181
Update function	"Update function" on page 183
Generate unique function	"Generate unique function" on page 188

When using parameter markers in UDFs, a Java database (JDBC) restriction requires that the parameter marker for the UDF must be casted to the data type of the column into which the returned data will be inserted. See "Limitations when invoking functions from Java database (JDBC)" on page 119 to learn how to cast the parameter markers.

## Storage functions

Use storage functions to insert XML documents into a DB2 database. You can use the default casting functions of a UDT directly in INSERT or SELECT statements as described in "Storing data" on page 108. Additionally, the XML Extender provides UDFs to take XML documents from sources other than the UDT base data type and convert them to the specified UDT.

The XML Extender provides the following storage functions:

- "XMLVarcharFromFile()" on page 155
- "XMLCLOBFromFile()" on page 156
- "XMLFileFromVarchar()" on page 157
- "XMLFileFromCLOB()" on page 158

## XMLVarcharFromFile()

### Purpose

Reads an XML document from a server file and returns the document as an XMLVARCHAR type.

### Syntax

►► XMLVarcharFromFile(—*fileName*—) ◀◀

### Parameters

Table 28. XMLVarcharFromFile parameter

Parameter	Data type	Description
<i>fileName</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

XMLVARCHAR

The following example reads an XML document from a server file and inserts it into an XML column as an XMLVARCHAR type.

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
VALUES('1234', 'Sriram Srinivasan',
XMLVarcharFromFile('dxx_install/samples/cmd/getstart.xml'))
```

In this example, a record is inserted into the SALES\_TAB table. The function XMLVarcharFromFile() imports the XML document from a file into DB2 and stores it as a XMLVARCHAR.

## XMLCLOBFromFile()

### Purpose

Reads an XML document from a server file and returns the document as an XMLCLOB type.

### Syntax

►► XMLCLOBFromFile(—*fileName*—)◄◄

### Parameters

Table 29. XMLCLOBFromFile parameter

Parameter	Data type	Description
<i>fileName</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

XMLCLOB as LOCATOR

The following example reads an XML document from a server file and inserts it into an XML column as an XMLCLOB type.

```
EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
VALUES('1234', 'Sriram Srinivasan',
XMLCLOBFromFile('dxx_install/samples/cmd/getstart.xml'))
```

The column ORDER in the SALES\_TAB table is defined as an XMLCLOB type. The preceding example shows how the column ORDER is inserted into the SALES\_TAB table.

## XMLFileFromVarchar()

### Purpose

Reads an XML document from memory as VARCHAR, writes it to an external server file, and returns the file name and path as an XMLFILE type.

### Syntax

►► XMLFileFromVarchar(—buffer—, —fileName—) ◀◀

### Parameters

Table 30. XMLFileFromVarchar parameters

Parameter	Data type	Description
<i>buffer</i>	VARCHAR(3K)	The memory buffer.
<i>fileName</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

XMLFILE

The following examples reads an XML document from memory as VARCHAR, writes it to an external server file, and inserts the file name and path as an XMLFILE type in an XML column.

```
EXEC SQL BEGIN DECLARE SECTION;
 struct { short len; char data[3000]; } xml_buf;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
 VALUES('1234', 'Sriram Srinivasan',
 XMLFileFromVarchar(:xml_buf, 'dxx_install/samples/cmd/getstart.xml'))
```

The column ORDER in the SALES\_TAB table is defined as an XMLFILE type. The preceding example shows that if you have an XML document in your buffer, you can store it in a server file.

## XMLFileFromCLOB()

### Purpose

Reads an XML document as CLOB locator, writes it to an external server file, and returns the file name and path as an XMLFILE type.

### Syntax

►► XMLFileFromCLOB(—buffer—, —fileName—) ◀◀

### Parameters

Table 31. XMLFileFromCLOB() parameters

Parameters	Data type	Description
<i>buffer</i>	CLOB as LOCATOR	The buffer containing the XML document.
<i>fileName</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

XMLFILE

The following example reads an XML document as CLOB locator (a host variable with a value that represents a single LOB value in the database server), writes it to an external server file, and inserts the file name and path as an XMLFILE type in an XML column.

```
EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
 VALUES('1234', 'Sriram Srinivasan',
 XMLFileFromCLOB(:xml_buf, 'dxx_install/samples/cmd/getstart.xml'))
```

The column ORDER in the SALES\_TAB table is defined as an XMLFILE type. If you have an XML document in your buffer, you can store it in a server file.

---

## Retrieval functions

The XML Extender provides an *overloaded function* Content(), which is used for retrieval. This overloaded function refers to a set of retrieval functions that have the same name, but behave differently based on where the data is being retrieved. You can also use the default casting functions to convert an XML UDT to the base data type as described in “Retrieving an entire document” on page 110.

The Content() functions provide the following types of retrieval:

- **Retrieval from external storage at the server to a host variable at the client.**

You can use Content() to retrieve an XML document to a memory buffer when it is stored as an external server file. You can use “Content(): retrieve from XMLFILE to a CLOB” on page 160 for this purpose.

- **Retrieval from internal storage to an external server file**

You can also use Content() to retrieve an XML document that is stored inside DB2 and store it to a server file on the DB2 server’s file system. The following Content() functions are used to store information on external server files:



- “Content(): retrieve from XMLVARCHAR to an external server file” on page 161
- “Content(): retrieval from XMLCLOB to an external server file” on page 162

The examples in the following section assume you are using the DB2 command shell, in which you do not need to type “DB2” at the beginning of each command.

## Content(): retrieve from XMLFILE to a CLOB

### Purpose

Retrieves data from a server file and stores it in a CLOB LOCATOR.

### Syntax

►►—Content—(—xmlobj—)—————►►

### Parameters

Table 32. XMLFILE to a CLOB parameter

Parameter	Data type	Description
<i>xmlobj</i>	XMLFILE	The XML document.

### Return type

CLOB (*clob\_len*) as LOCATOR

*clob\_len* for DB2 UDB is 2G.

The following example retrieves data from a server file and stores it in a CLOB locator.

```
EXEC SQL BEGIN DECLARE SECTION;
 SQL TYPE IS CLOB_LOCATOR xml_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO SALES_DB

EXEC SQL DECLARE c1 CURSOR FOR

 SELECT Content(order) from sales_tab
 WHERE sales_person = 'Sriram Srinivasan'

EXEC SQL OPEN c1;

do {
 EXEC SQL FETCH c1 INTO :xml_buff;
 if (SQLCODE != 0) {
 break;
 }
 else {
 /* do with the XML doc in buffer */
 }
}

EXEC SQL CLOSE c1;

EXEC SQL CONNECT RESET;
```

The column ORDER in the SALES\_TAB table is of an XMLFILE type, so the Content() UDF retrieves data from a server file and stores it in a CLOB locator.

## Content(): retrieve from XMLVARCHAR to an external server file

### Purpose

Retrieves the XML content that is stored as an XMLVARCHAR type and stores it in an external server file.

### Syntax

►► Content(—xmlobj—, —filename—) ◀◀

**Important:** If a file with the specified name already exists, the content function overrides its content.

### Parameters

Table 33. XMLVarchar to external server file parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR	The XML document.
<i>filename</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

VARCHAR(512)

The following example retrieves the XML content that is stored as XMLVARCHAR type and stores it in an external server file.

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLVarchar);
INSERT into app1 values (1, '<?xml version="1.0"?>
<!DOCTYPE SYSTEM dxx_install/samples/dtd/getstart.dtd"->
<Order key="1">
 <Customer>
 <Name>American Motors</Name>
 <Email>parts@am.com</Email>
 </Customer>
 <Part color="black">
 <key>68</key>
 <Quantity>36</Quantity>
 <ExtendedPrice>34850.16</ExtendedPrice>
 <Tax>6.000000e-02</Tax>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>AIR </ShipMode>
 </Shipment>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>BOAT </ShipMode>
 </Shipment>
 </Part>
</Order>');

SELECT DB2XML.Content(order, 'dxx_install/samples/dad/getstart_column.dad')
from app1 where ID=1;
```

## Content(): retrieval from XMLCLOB to an external server file

### Purpose

Retrieves the XML content that is stored as an XMLCLOB type and stores it in an external server file.

### Syntax

►► Content(—xmlobj—,—filename—)◄◄

**Important:** If a file with the specified name already exists, the content function overrides its content.

### Parameters

Table 34. XMLCLOB to external server file parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLCLOB as LOCATOR	The XML document.
<i>filename</i>	VARCHAR(512)	The fully qualified server file name.

### Return type

VARCHAR(512)

The following example retrieves the XML content that is stored as an XMLCLOB type and stores it in an external server file.

```
CREATE table app1 (id int NOT NULL, order DB2XML.XMLCLOB not logged);
```

```
INSERT into app1 values (1, '<?xml version="1.0"?>
<!DOCTYPE SYSTEM dxx_install/samples/dtd/getstart.dtd"->
<Order key="1">
 <Customer>
 <Name>American Motors</Name>
 <Email>parts@am.com</Email>
 </Customer>
 <Part color="black">
 <key>68</key>
 <Quantity>36</Quantity>
 <ExtendedPrice>34850.16</ExtendedPrice>
 <Tax>6.000000e-02</Tax>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>AIR </ShipMode>
 </Shipment>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>BOAT </ShipMode>
 </Shipment>
 </Part>
</Order>');
```

```
SELECT DB2XML.Content(order, 'dxx_install/samples/cmd/getstart.xml')
from app1 where ID=1;
```

---

## Extracting functions

The extracting functions extract the element content or attribute value from an XML document and return the requested SQL data types. The XML Extender provides a set of extracting functions for various SQL data types. The extracting functions take two input parameters. The first parameter is the XML Extender UDT, which can be one of the XML UDTs. The second parameter is the location path that specifies the XML element or attribute. Each extracting function returns the value or content that is specified by the location path.

Because some element or attribute values have multiple occurrence, the extracting functions return either a scalar or a table value; the former is called a scalar function, the latter is called a table function.

The XML Extender provides the following extracting functions:

- “extractInteger() and extractIntegers()” on page 164
- “extractSmallint() and extractSmallints()” on page 165
- “extractDouble() and extractDoubles()” on page 167
- “extractReal() and extractReals()” on page 169
- “extractChar()and extractChars()” on page 171
- “extractVarchar() and extractVarchars()” on page 173
- “extractCLOB() and extractCLOBs()” on page 175
- “extractDate() and extractDates()” on page 177
- “extractTime() and extractTimes()” on page 179
- “extractTimestamp() and extractTimestamps()” on page 181

The examples in the following section assume you are using the DB2 command shell, in which you do not need to type “DB2” at the beginning of each command.

## extractInteger() and extractIntegers()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as INTEGER type.

### Syntax

#### Scalar function:

►► extractInteger(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractIntegers(—xmlobj—, —path—) ◀◀

### Parameters

Table 35. extractInteger and extractIntegers function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

INTEGER

### Returned column name (table function)

returnedInteger

#### Scalar function example:

In the following example, one value is returned when the attribute value of key = "1". The value is extracted as an INTEGER.

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
 DB2XML.extractInteger(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

#### Table function example:

In the following example, each order key for the sales orders extracted as INTEGER.

```
SELECT *
FROM TABLE(
 DB2XML.extractIntegers(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/key')) AS X;
```

## extractSmallint() and extractSmallints()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as SMALLINT type.

### Syntax

#### Scalar function:

►► extractSmallint(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractSmallints(—xmlobj—, —path—) ◀◀

### Parameters

Table 36. extractSmallint and extractSmallints function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

SMALLINT

### Returned column name (table function)

returnedSmallint

#### Scalar function example:

In the following example, the value of key in all sales orders is extracted as SMALLINT

```
CREATE TABLE t1(key INT);
INSERT INTO t1 values (
 DB2XML.extractSmallint(b2xml.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="black "]/key'));
SELECT * from t1;
```

### Table function example:

In the following example, the value of key in all sales orders is extracted as SMALLINT

```
SELECT *
FROM TABLE(
 DB2XML.extractSmallints(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/key')) AS X;
```



## extractDouble() and extractDoubles()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as DOUBLE type.

### Syntax

#### Scalar function:

►► extractDouble(*--xmlobj--*, *--path--*) ◀◀

#### Table function:

►► extractDoubles(*--xmlobj--*, *--path--*) ◀◀

### Parameters

Table 37. *extractDouble* and *extractDoubles* function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

DOUBLE

### Returned column name (table function)

returnedDouble

#### Scalar function example:

The following example automatically converts the price in an order from a DOUBLE type to a DECIMAL.

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
 DB2XML.extractDouble(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="black "]/ExtendedPrice');
SELECT * from t1;
```

**Table function example:**

In the following example, the value of ExtendedPrice in each part of the sales order is extracted as DOUBLE.

```
SELECT CAST(RETURNEDDOUBLE AS DOUBLE)
FROM TABLE(
 DB2XML.extractDoubles(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/ExtendedPrice')) AS X;
```

## extractReal() and extractReals()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as REAL type.

### Syntax

#### Scalar function:

►► extractReal(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractReals(—xmlobj—, —path—) ◀◀

### Parameters

Table 38. extractReal and extractReals function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

REAL

### Returned column name (table function)

returnedReal

#### Scalar function example:

In the following example, the value of ExtendedPrice is extracted as a REAL.

```
CREATE TABLE t1(price DECIMAL(9,2));
INSERT INTO t1 values (
 DB2XML.extractReal(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="black"]/ExtendedPrice');
SELECT * from t1;
```

**Table function example:**

In the following example, the value of ExtendedPrice is extracted as a REAL.

```
SELECT CAST(RETURNEDREAL AS REAL)
FROM TABLE(
 DB2XML.extractReals(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/ExtendedPrice')) AS X;
```

## extractChar()and extractChars()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as CHAR type.

### Syntax

#### Scalar function:

►► extractChar(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractChars(—xmlobj—, —path—) ◀◀

### Parameters

Table 39. extractChar and extractCharsfunction parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

CHAR

### Returned column name (table function)

returnedChar

#### Scalar function example:

In the following example, the value of Name is extracted as CHAR.

```
CREATE TABLE t1(name char(30));
INSERT INTO t1 values (
 DB2XML.extractChar(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Customer/Name'));
SELECT * from t1;
```

**Table function example:**

In the following example, the value of Color is extracted as CHAR.

```
SELECT *
FROM TABLE(
 DB2XML.extractChars(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/@color')) AS X;
```

## extractVarchar() and extractVarchars()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as VARCHAR type.

### Syntax

#### Scalar function:

►► extractVarchar(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractVarchars(—xmlobj—, —path—) ◀◀

### Parameters

Table 40. extractVarchar and extractVarchars function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

VARCHAR(4K)

### Returned column name (table function)

returnedVarchar

#### Scalar function example:

In the following example, the value of Name is extracted as VARCHAR.

```
CREATE TABLE t1(name varchar(30));
INSERT INTO t1 values (
 DB2XML.extractVarchar(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Customer/Name'));
SELECT * from t1;
```

**Table function example:**

In the following example, the value of Color is extracted as VARCHAR.

```
SELECT*
FROM TABLE(
 DB2XML.extractVarchars(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/@color')) AS X;
```



## extractCLOB() and extractCLOBs()

### Purpose

Extracts a fragment of XML documents, with element and attribute markup, content of elements and attributes, including sub-elements. This function differs from the other extract functions; they return only the content of elements and attributes. The extractClob(s) functions should be used to extract document fragments, whereas extractVarchar(s) and extractChar(s) should be used to extract simple values.

### Syntax

#### Scalar function:

►► extractCLOB(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractCLOBs(—xmlobj—, —path—) ◀◀

### Parameters

Table 41. extractCLOB and extractCLOBs function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

CLOB(10K)

### Returned column name (table function)

returnedCLOB

#### Scalar function example:

In this example, all name element content and tags are extracted from a purchase order.

```
CREATE TABLE t1(name DB2XML.xmlclob);
INSERT INTO t1 values (
 DB2XML.extractClob(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Customer/Name'));
SELECT * from t1;
```

**Table function example:**

In this example, all of the color attributes are extracted from a purchase order.

```
SELECT *
FROM TABLE(
 DB2XML.extractCLOBs(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part/@color')) AS X;
```

## extractDate() and extractDates()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as DATE type.

### Syntax

#### Scalar function:

►► extractDate(—xmlobj—, —path—) ◀◀

#### Table function:

►► extractDates(—xmlobj—, —path—) ◀◀

### Parameters

Table 42. extractDate and extractDates function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

DATE

### Returned column name (table function)

returnedDate

#### Scalar function example:

In the following example, the value of ShipDate is extracted as DATE.

```
CREATE TABLE t1(shipdate DATE);
INSERT INTO t1 values (
 DB2XML.extractDate(DB2XML.xmlfile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="red "]/Shipment/ShipDate');
SELECT * from t1;
```

### Table function example:

In the following example, the value of ShipDate is extracted as DATE.

```
SELECT *
FROM TABLE(
 DB2XML.extractDates(DB2XML.XMLFile('dxx_install/samples/xml/getstart.xml'),
 '/Order/Part[@color="black "]/Shipment/ShipDate')) AS X;
```

## extractTime() and extractTimes()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as TIME type.

### Syntax

#### Scalar function:

►► extractTime(—xmlObj—, —path—) ◀◀

#### Table function:

►► extractTimes(—xmlObj—, —path—) ◀◀

### Parameters

Table 43. extractTime and extractTimes function parameters

Parameter	Data type	Description
<i>xmlObj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

TIME

### Returned column name (table function)

returnedTime

#### Scalar function example:

```
CREATE TABLE t1(testtime TIME);
INSERT INTO t1 values (
 DB2XML.extractTime(DB2XML.XMLCLOB(
 '<stuff><data>11.12.13</data></stuff>'), '//data'));
SELECT * from t1;
```

**Table function example:**

```
select *
from table(
 DB2XML.extractTimes(DB2XML.XMLCLOB(
 '<stuff><data>01.02.03</data><data>11.12.13</data></stuff>'),
 '//data')) as x;
```

## extractTimestamp() and extractTimestamps()

### Purpose

Extracts the element content or attribute value from an XML document and returns the data as `TIMESTAMP` type.

### Syntax

#### Scalar function:

```
►► extractTimestamp(—xmlobj—, —path—) ◀◀
```

#### Table function:

```
►► extractTimestamps(—xmlobj—, —path—) ◀◀
```

### Parameters

Table 44. *extractTimestamp* and *extractTimestamps* function parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.

### Return type

`TIMESTAMP`

### Returned column name (table function)

`returnedTimestamp`

#### Scalar function example:

```
CREATE TABLE t1(testtimestamp TIMESTAMP);
INSERT INTO t1 values (
 DB2XML.extractTimestamp(DB2XML.XMLCLOB(
 '<stuff><data>1998-11-11-11.12.13.888888</data></stuff>'),
 '//data');
SELECT * from t1;
```

**Table function example:**

```
select * from
table(DB2XML.extractTimestamps(DB2XML.XMLClob(
 '<stuff><data>1998-11-11-11.12.13.888888
 </data><data>1998-12-22-11.12.13.888888</data></stuff>'),
 '//data')) as x;
```



---

## Update function

The Update() function updates a specified element or attribute value in one or more XML documents stored in the XML column. You can also use the default casting functions to convert an SQL base type to the XML UDT, as described in “Updating XML data” on page 113.

### Purpose

Takes the column name of an XML UDT, a location path, and a string of the update value and returns an XML UDT that is the same as the first input parameter. With the Update() function, you can specify the element or attribute that is to be updated.

### Syntax

►► Update(—xmlobj—, —path—, —value—) ◀◀

### Parameters

Table 45. The UDF Update parameters

Parameter	Data type	Description
<i>xmlobj</i>	XMLVARCHAR, XMLCLOB as LOCATOR	The column name.
<i>path</i>	VARCHAR	The location path of the element or attribute.
<i>value</i>	VARCHAR	The update string.  <b>Restriction:</b> The Update function does not have an option to disable output escaping; the output of an extractClob (which is a tagged fragment) cannot be inserted using this function. Use textual values, only.

**Important:** Note that the Update UDF supports location paths that have predicates with attributes, but not elements. For example, the following predicate is supported:

```
'/Order/Part[@color="black "]/ExtendedPrice'
```

The following predicate is not supported:

```
'/Order/Part/Shipment/[Shipdate < "11/25/00"]'
```

### Return type

Data type	Return type
XMLVARCHAR	XMLVARCHAR
XMLCLOB as LOCATOR	XMLCLOB

## Example

The following example updates the purchase order handled by the salesperson Sriram Srinivasan.

```
UPDATE sales_tab
 set order = db2xml.update(order, '/Order/Customer/Name', 'IBM')
 WHERE sales_person = 'Sriram Srinivasan'
```

In this example, the content of /Order/Customer/Name is updated to IBM.

## Usage

When you use the Update function to change a value in one or more XML documents, it replaces the XML documents within the XML column. Based on output from the XML parser, some parts of the original document are preserved, while others are lost or changed. The following sections describe how the document is processed and provide examples of how the documents look before and after updates.

### How the Update function processes the XML document

When the Update function replaces XML documents, it must reconstruct the document based on the XML parser output. Table 46 describes how the parts of the document are handled, with examples. For examples that compare XML documents before and after an update, see “Examples” on page 186.

Table 46. Update function rules

Item or Node type	XML document code example	Status after update
XML Declaration	<pre>&lt;?xml version='1.0' encoding='utf-8' standalone='yes' &gt;</pre>	<p>The XML declaration is preserved:</p> <ul style="list-style-type: none"><li>• Version information is preserved.</li><li>• Encoding declaration is preserved and appears when specified in the original document.</li><li>• Standalone declaration is preserved and appears when specified in the original document.</li><li>• After update, single quotes are used to delineate values.</li></ul>

Table 46. Update function rules (continued)

Item or Node type	XML document code example	Status after update
DOCTYPE Declaration	<pre>&lt;!DOCTYPE books SYSTEM "http://dtds.org/books.dtd" &gt; &lt;!DOCTYPE books PUBLIC "local.books.dtd" "http://dtds.org/books.dtd" &gt; &lt;!DOCTYPE books&gt; -Any of &lt;!DOCTYPE books ( S ExternalID ) ? [ internal-dtd-subset ] &gt; -Such as &lt;!DOCTYPE books [ &lt;!ENTITY mydog "Spot"&gt; ] &gt;? [ internal-dtd-subset ] &gt;</pre>	<p>The document type declaration is preserved:</p> <ul style="list-style-type: none"> <li>• Root element name is supported.</li> <li>• Public and system ExternalIDs are preserved and appear when specified in the original document.</li> <li>• Internal DTD subset is NOT preserved. Entities are replaced; defaults for attributes are processed and appear in the output documents.</li> <li>• After update, double quotes are used to delineate public and system URI values.</li> <li>• The current XML4c parser does not report an XML declaration that does not contain an ExternalID or internal DTD subset. After update, the DOCTYPE declaration would be missing in this case.</li> </ul>
Processing Instructions	<pre>&lt;?xml-stylesheet title="compact" href="datatypes1.xsl" type="text/xsl"?&gt;</pre>	<p>Processing instructions are preserved.</p>
Comments	<pre>&lt;!-- comment --&gt;</pre>	<p>Comments are preserved when inside the root element.</p> <p>Comments outside the root element are discarded.</p>
Elements	<pre>&lt;books&gt; content &lt;/books&gt;</pre>	<p>Elements are preserved.</p>
Attributes	<pre>id='1' date="01/02/1997"</pre>	<p>Attributes of elements are preserved.</p> <ul style="list-style-type: none"> <li>• After update, double quotes are used to delineate values.</li> <li>• Data within attributes is escaped.</li> <li>• Entities are replaced.</li> </ul>
Text Nodes	<pre>This chapter is about my dog &amp;mydoc;.</pre>	<p>Text nodes (element content) are preserved.</p> <ul style="list-style-type: none"> <li>• Data within text nodes is escaped.</li> <li>• Entities are replaced.</li> </ul>

## Multiple occurrence

When a location path is provided in the Update() UDF, the content of every element or attribute with a matching path is updated with the supplied value. This means that if a document has multiple occurring locations paths, the Update function replaces the existing values with the value provided in the *value* parameter.

You can use specify a predicate in the *path* parameter to provide distinct locations paths to prevent unintentional updates. Note, that the Update UDF supports location paths that have predicates with attributes, but not elements. See “Parameters” on page 183 for more information.

## Examples

The following examples show instances of an XML document before and after an update.

Table 47. XML documents before and after an update

Example 1:

### Before:

```
<?xml version='1.0' encoding='utf-8' standalone="yes"?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?>
<!-- comment -->
<book>
 <chapter id="1" date='07/01/1997'>
 <!-- first section -->
 <section>This is a section in Chapter One.</section>
 </chapter>
 <chapter id="2" date="01/02/1997">
 <section>This is a section in Chapter Two.</section>
 <footnote>A footnote in Chapter Two is here.</footnote>
 </chapter>
 <price date="12/22/1998" time="11.12.13"
 timestamp="1998-12-22-11.12.13.888888">38.281</price>
</book>
```

- Contains white space in the XML declaration
- Specifies a processing instruction
- Contains a comment outside of the root node
- Specifies PUBLIC ExternalID
- Contains a comment inside of root note

### After:

```
<?xml version='1.0' encoding='utf-8' standalone='yes'?>
<!DOCTYPE book PUBLIC "public.dtd" "system.dtd">
<?pitarget option1='value1' option2='value2'?><book>
 <chapter id="1" date="07/01/1997">
 <!-- first section -->
 <section>This is a section in Chapter One.</section>
 </chapter>
 <chapter id="2" date="01/02/1997">
 <section>This is a section in Chapter Two.</section>
 <footnote>A footnote in Chapter Two is here.</footnote>
 </chapter>
 <price date="12/22/1998" time="11.12.13"
 timestamp="1998-12-22-11.12.13.888888">60.02</price>
</book>
```

- White space inside of markup is eliminated
- Processing instruction is preserved
- Comment outside of the root node is not preserved
- PUBLIC ExternalID is preserved
- Comment inside of root node is preserved
- Changed value is the value of the <price> element

Table 47. XML documents before and after an update (continued)

Example 2:

**Before:**

```
<?xml version='1.0' ?>
<!DOCTYPE book>
<!-- comment -->
<book>
 ...
</book>
```

Contains DOCTYPE declaration without an ExternalID or an internal DTD subset. Not supported.

**After:**

```
<?xml version='1.0'?>
<book>
 ...
</book>
```

DOCTYPE declaration is not reported by the XML parser and not preserved.

Example 3:

**Before:**

```
<?xml version='1.0' ?>
<!DOCTYPE book [<!ENTITY myDog "Spot">]>
<!-- comment -->
<book>
 <chapter id="1" date='07/01/1997'>
 <!-- first section -->
 <section>This is a section in Chapter
 One about my dog &myDog;.</section>
 ...
 </chapter>
 ...
</book>
```

- Contains white space in markup
- Specifies internal DTD subset
- Specifies entity in text node

**After:**

```
<?xml version='1.0'?>
<!DOCTYPE book>
<book>
 <chapter id="1" date="07/01/1997">
 <!-- first section -->
 <section>This is a section in Chapter
 One about my dog Spot.</section>
 ...
 </chapter>
 ...
</book>
```

- White space in markup is eliminated
- Internal DTD subset is not preserved
- Entity in text node is resolved and replaced

---

## Generate unique function

### Purpose

The generate unique function returns a character string that is unique compared to any other execution of the same function. There are no arguments to this function (the empty parentheses must be specified). The result of the function is a unique value. The result cannot be null.

### Syntax

►►—db2xml.generate\_unique()—◄◄

### Return value

VARCHAR(13)

### Example

The following example uses db2xml.generate\_unique() to generate a unique key for a column to be indexed.

```
<SQL_stmt>
SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
price, tax, ship_id, date, mode from order_tab o, part_tab p,
table (select db2xml.generate_unique()
as ship_id, date, mode, part_key from ship_tab) s
WHERE o.order_key = 1 and
p.price > 20000 and
p.order_key = o.order_key and
s.part_key = p.part_key
ORDER BY order_key, part_key, ship_id
</SQL_stmt>
```

---

## Chapter 13. XML Extender stored procedures

The XML Extender provides stored procedures for administration and management of XML columns and collections. These stored procedures can be called from the DB2 client. The client interface can be embedded in SQL, ODBC, or JDBC. Refer to the section on stored procedures in the *DB2 UDB for OS/390 Administration Guide* for details on how to call stored procedures.

The stored procedures use the schema DB2XML, which is the schema name of the XML Extender.

The XML Extender provides three types of stored procedures:

- “Administration stored procedures” on page 191, which assist users in completing administrative tasks
- “Composition stored procedures” on page 198, which generate XML documents using data in existing database tables
- “Decomposition stored procedures” on page 206, which break down or shred incoming XML documents and store data in new or existing database tables

The parameter limits used by the XML collection stored procedures are documented in “Appendix D. The XML Extender limits” on page 263.

---

### Specifying include files

Ensure that you include the XML Extender external header files in the program that calls stored procedures. The header files are located in the *dxx\_install/include* directory. *dxx\_install* is the installation directory for the XML Extender. It is operating system dependent. The header files are:

**dxx.h**            The XML Extender defined constant and data types

**dxxrc.h**         The XML Extender return code

The syntax for including these header files is:

```
#include "dxx.h"
#include "dxxrc.h"
```

Make sure that the path of the include files is specified in your makefile with the compilation option.

---

### Calling XML Extenders stored procedures

In general, call the XML Extender using the following syntax:

```
CALL DB2XML.function_entry_point
```

Where:

*function\_entry\_point*

Specifies the arguments passed to the stored procedure.

In the CALL statement, the arguments that are passed to the stored procedure must be host variables, not constants or expressions. The host variables can have null indicators.

See samples for calling stored procedures in the *dxx\_install/samples/c* and *dxx\_install/samples/cli* directories, and in the following sections of this book: “Composing the XML document” on page 31 and “Chapter 9. Managing XML collection data” on page 121. In the *dxx\_install/samples/c* directory, SQC code files are provided to call XML collection stored procedures using embedded SQL. In the *dxx\_install/samples/cli* directory, the sample files show how to call stored procedures using the Call Level Interface (CLI).

---

## Increasing the CLOB limit

The default limit for CLOB parameter when passed to a stored procedure is 1 MB. You can increase the limit by completing the following steps:

1. Drop each stored procedure. For example:

```
db2 "drop procedure DB2XML.dxxShredXML"
```

2. Create a new procedure with the increased CLOB limit. For example:

```
db2 "create procedure DB2XML.dxxShredXML(in dadBuf clob(100K),
 in XMLObj clob(10M),
 out returnCode integer,
 out returnMsg varchar(1024)
)
 external name 'DB2XML.dxxShredXML'
 language C
 parameter style DB2DARI
 not deterministic
 fenced
 null call;
```



---

## Before you begin

Run the DXXGPREP JCL job before working with stored procedures. See “Initializing the XML Extender environment using DXXGPREP” on page 39 to learn more about initializing the XML Extender before running stored procedures.

---

## Administration stored procedures

These stored procedures are used for administration tasks, such as enabling or disabling an XML column or collection. They are called by the XML Extender administration wizard and the administration command DXXADM. These stored procedures are:

- dxxEnableDB()
- dxxDisableDB()
- dxxEnableColumn()
- dxxDisableColumn()
- dxxEnableCollection()
- dxxDisableCollection()

## dxxEnableSRV()

### Purpose

Enables the database server. When the database server is enabled, the XML Extender creates the following objects:

- The XML Extender user-defined types (UDTs)
- The XML Extender user-defined functions (UDFs)
- The XML Extender stored procedures
- The XML Extender DTD reference table, DTD\_REF, which stores DTDs and information about each DTD. For a complete description of the DTD\_REF table, see “DTD reference table” on page 213.
- The XML Extender usage table, XML\_USAGE, which stores common information for each column that is enabled for XML and for each collection. For a complete description of the XML\_USAGE table, see “XML usage table” on page 213.

### Format

```
dxxEnableSRV(long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
 varchar (254) tspaceNames /* input */
 varchar (64) wlmNames /* input */
 varchar (18) extSecurity /* input */
```

### Parameters

Table 48. *dxxEnableSRV()* parameters

Parameter	Description	IN/OUT parameter
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT
<i>tspaceNames</i>	The name of the table spaces used for administration tables, DTD_REF and XML_USAGE	IN
<i>wlmNames</i>	The names of the WLM environments.	IN
<i>extSecurity</i>	The external security option; values can be DB2 (default), USER, or DEFINER	IN

## dxxDisableSRV()

### Purpose

Disables the database server. When the XML Extender disables the database server, it drops the following objects:

- The XML Extender user-defined types (UDTs).
- The XML Extender user-defined functions (UDFs).
- The XML Extender DTD reference table, DTD\_REF, which stores DTDs and information about each DTD. For a complete description of the DTD\_REF table, see “DTD reference table” on page 213.
- The XML Extender usage table, XML\_USAGE, which stores common information for each column that is enabled for XML and for each collection. For a complete description of the XML\_USAGE table, see “XML usage table” on page 213.

**Important:** You must disable all XML columns before attempting to disable a database server. The XML Extender cannot disable a database server that contains tables with columns or collections that are enabled for XML.

### Format

```
dxxDisableSRV(char(
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

### Parameters

Table 49. *dxxDisableDB()* parameters

Parameter	Description	IN/OUT parameter
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## dxxEnableColumn()

### Purpose

Enables an XML column. When enabling a column, the XML Extender completes the following tasks:

- Determines whether the XML table has a primary key; if not, the XML Extender alters the XML table and adds a column called DXXROOT\_ID.
- Creates side tables that are specified in the DAD file with a column containing a unique identifier for each row in the XML table. This column is either the `root_id` that is specified by the user, or it is the DXXROOT\_ID that was named by the XML Extender.
- Creates a default view for the XML table and its side tables, optionally using a name you specify.

### Format

```
dxxEnableColumn(char(tbName) tbName, /* input */
 char(colName) colName, /* input */
 CLOB(100K) DAD, /* input */
 char(tablespace) tablespace, /* input */
 char(defaultView) defaultView, /* input */
 char(rootID) rootID, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

### Parameters

Table 50. *dxxEnableColumn()* parameters

Parameter	Description	IN/OUT parameter
<i>tbName</i>	The name of the table containing the XML column.	IN
<i>colName</i>	The name of the XML column.	IN
<i>DAD</i>	A CLOB containing the DAD file.	IN
<i>tablespace</i>	The table space that contains the side tables other than the default table space. If not specified, the default table space is used.	IN
<i>defaultView</i>	The name of the default view joining the application table and side tables.	IN
<i>rootID</i>	The name of the single primary key in the application table that is to be used as the root ID for the side table.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## dxxDisableColumn()

### Purpose

Disables the XML-enabled column. When an XML column is disabled, it can no longer contain XML data types.

### Format

```
dxxDisableColumn(char(tbName) tbName, /* input */
 char(colName) colName, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

### Parameters

Table 51. *dxxDisableColumn()* parameters

Parameter	Description	IN/OUT parameter
<i>tbName</i>	The name of the table containing the XML column.	IN
<i>colName</i>	The name of the XML column.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## dxxEnableCollection()

### Purpose

Enables an XML collection that is associated with an application table.

### Format

```
dxxEnableCollection(char() dbName, /* input */
 char(colName) colName, /* input */
 CLOB(100K) DAD, /* input */
 char(tableSpace) tableSpace, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

### Parameters

Table 52. *dxxEnableCollection()* parameters

Parameter	Description	IN/OUT parameter
<i>colName</i>	The name of the XML collection.	IN
<i>DAD</i>	A CLOB containing the DAD file.	IN
<i>tableSpace</i>	The table space that contains the side tables other than the default table space. If not specified, the default table space is used.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## dxxDisableCollection()

### Purpose

Disables an XML-enabled collection, removing markers that identify tables and columns as part of a collection.

### Format

```
dxxDisableCollection(char(colName) colName, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

### Parameters

Table 53. *dxxDisableCollection()* parameters

Parameter	Description	IN/OUT parameter
<i>colName</i>	The name of the XML collection.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

---

## Composition stored procedures

The composition stored procedures `dxxGenXML()` and `dxxRetrieveXML()` are used to generate XML documents using data in existing database tables. The `dxxGenXML()` stored procedure takes a DAD file as input; it does not require an enabled XML collection. The `dxxRetrieveXML()` stored procedure takes an enabled XML collection name as input.



## dxxGenXML()

### Purpose

Constructs XML documents using data that is stored in the XML collection tables that are specified by the <Xcollection> in the DAD file and inserts each XML document as a row into the result table. You can also open a cursor on the result table and fetch the result set.

To provide flexibility, dxxGenXML() also lets the user specify the maximum number of rows to be generated in the result table. This decreases the amount of time the application must wait for the results during any trial process. The stored procedure returns the number of actual rows in the table and any error information, including error codes and error messages.

To support dynamic query, dxxGenXML() takes an input parameter, *override*. Based on the input *overrideType*, the application can override the SQL\_stmt for SQL mapping or the conditions in RDB\_node for RDB\_node mapping in the DAD file. The input parameter *overrideType* is used to clarify the type of the *override*. For details about the *override* parameter, see “Dynamically overriding values in the DAD file” on page 126.

### Format

```
dxxGenXML(CLOB(100K) DAD, /* input */
 char(32) resultTabName, /* input */
 char(30) result_column, /* input */
 char(30) valid_column, /* input */
 integer overrideType /* input */
 varchar(1024) override, /* input */
 integer maxRows, /* input */
 integer numRows, /* output */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

## Parameters

Table 54. *dxxGenXML()* parameters

Parameter	Description	IN/OUT parameter
<i>DAD</i>	A CLOB containing the DAD file.	IN
<i>resultTabName</i>	The name of the result table, which should exist before the call. The table contains only one column of either XMLVARCHAR or XMLCLOB type.	IN
<i>overrideType</i>	A flag to indicate the type of the following <i>override</i> parameter: <ul style="list-style-type: none"> <li>• <b>NO_OVERRIDE</b>: No override.</li> <li>• <b>SQL_OVERRIDE</b>: Override by an SQL_stmt.</li> <li>• <b>XML_OVERRIDE</b>: Override by an XPath-based condition.</li> </ul>	IN
<i>override</i>	Overrides the condition in the DAD file. The input value is based on the <i>overrideType</i> . <ul style="list-style-type: none"> <li>• <b>NO_OVERRIDE</b>: A NULL string.</li> <li>• <b>SQL_OVERRIDE</b>: A valid SQL statement. Using this <i>overrideType</i> requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file.</li> <li>• <b>XML_OVERRIDE</b>: A string that contains one or more expressions in double quotation marks separated by "AND". Using this <i>overrideType</i> requires that RDB_node mapping is used in the DAD file.</li> </ul>	IN
<i>maxRows</i>	The maximum number of rows in the result table.	IN
<i>numRows</i>	The actual number generated rows in the result table.	OUT
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## Examples

The following example fragment assumes that a result table is created with the name of XML\_ORDER\_TAB, and that the table has one column of XMLVARCHAR type. A complete, working sample is located in DXXSAMPLES/QCSRC(GENX).

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE is CLOB(100K) dad; /* DAD */
EXEC SQL DECLARE :dad VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running from USS */
/* to ensure that DB2 converts the code page correctly*/

char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
```

```

char valid_colname[32]; /* name of the valid column, will set to NULL */
char override[2]; /* override, will set to NULL*/
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short dad_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &dad.data;
, 1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf("Error reading dad file
/dxx/samples/dad/getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
 dad.length = file_length;
}
else {
 printf("Error opening dad file \n",);
 rc = -1;
 goto exit;
}
/* initialize host variable and indicators */
strcpy(result_tab,"xml_order_tab");
strcpy(result_colname, "xmlorder")
valid_colname = '\0';
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
dad_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXGENXML" (:dad:dad_ind;
 :result_tab:rtab_ind,

```

```
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind,:override:ov_ind,
 :max_row:maxrow_ind,:num_row:numrow_ind,
 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
 else
 EXEC SQL COMMIT;
}

exit:
 return rc;
```

## dxxRetrieveXML()

### Purpose

Enables the same DAD file to be used for both composition and decomposition. The stored procedure `dxxRetrieveXML()` also serves as a means for retrieving decomposed XML documents. As input, `dxxRetrieveXML()` takes a buffer containing the DAD file, the name of the created result table, and the maximum number of rows to be returned. It returns a result set of the result table, the actual number of rows in the result set, an error code, and message text.

To support dynamic query, `dxxRetrieveXML()` takes an input parameter, *override*. Based on the input *overrideType*, the application can override the `SQL_stmt` for SQL mapping or the conditions in `RDB_node` for `RDB_node` mapping in the DAD file. The input parameter *overrideType* is used to clarify the type of the *override*. For details about the *override* parameter, see “Dynamically overriding values in the DAD file” on page 126.

The requirements of the DAD file for `dxxRetrieveXML()` are the same as the requirements for `dxxGenXML()`. The only difference is that the DAD is not an input parameter for `dxxRetrieveXML()`, but it is the name of an enabled XML collection.

### Format

```
dxxRetrieveXML(char(32) collectionName, /* input */
 char(32) resultTabName, /* input */
 char(30) result_column, /* input */
 char(30) valid_column, /* input */
 integer overrideType, /* input */
 varchar(1024) override, /* input */
 integer maxRows, /* input */
 integer numRows, /* output */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

## Parameters

Table 55. *dxxRetrieveXML()* parameters

Parameter	Description	IN/OUT parameter
<i>collectionName</i>	The name of an enabled XML collection.	IN
<i>resultTabName</i>	The name of the result table, which should exist before the call. The table contains only one column of either XMLVARCHAR or XMLCLOB type.	IN
<i>overrideType</i>	A flag to indicate the type of the following <i>override</i> parameter: <ul style="list-style-type: none"> <li>• <b>NO_OVERRIDE</b>: No override.</li> <li>• <b>SQL_OVERRIDE</b>: Override by an SQL_stmt.</li> <li>• <b>XML_OVERRIDE</b>: Override by an XPath-based condition.</li> </ul>	IN
<i>override</i>	Overrides the condition in the DAD file. The input value is based on the <i>overrideType</i> . <ul style="list-style-type: none"> <li>• <b>NO_OVERRIDE</b>: A NULL string.</li> <li>• <b>SQL_OVERRIDE</b>: A valid SQL statement. Using this <i>overrideType</i> requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file.</li> <li>• <b>XML_OVERRIDE</b>: A string that contains one or more expressions in double quotation marks, separated by "AND". Using this <i>overrideType</i> requires that RDB_node mapping is used in the DAD file.</li> </ul>	IN
<i>maxRows</i>	The maximum number of rows in the result table.	IN
<i>numRows</i>	The actual number of generated rows in the result table.	OUT
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## Examples

The following fragment is an example of a call to *dxxRetrieveXML()*. In this example, a result table is created with the name of XML\_ORDER\_TAB, and it has one column of XMLVARCHAR type. A complete, working sample is located in DXXSAMPLES/QCSRC (RTRX).

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
char result_tab[32]; /* name of the result table */
char result_colname[32]; /* name of the result column */
char valid_colname[32]; /* name of the valid column, will set to NULL*/
```

```

char override[2]; /* override, will set to NULL*/
short overrideType; /* defined in dxx.h */
short max_row; /* maximum number of rows */
short num_row; /* actual number of rows */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short rtab_ind;
short rcol_ind;
short vcol_ind;
short ovtype_ind;
short ov_ind;
short maxrow_ind;
short numrow_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

/* create table */
EXEC SQL CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initial host variable and indicators */
strcpy(collection, "sales_ord");
strcpy(result_tab, "xml_order_tab");
strcpy(result_col, "xmlorder");
valid_colname[0] = '\0';
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
rtab_ind = 0;
rcol_ind = 0;
vcol_ind = -1;
ov_ind = -1;
ovtype_ind = 0;
maxrow_ind = 0;
numrow_ind = -1;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXRETRIEVE" (:collectionName:collectionName_ind,
 :result_tab:rtab_ind,
 :result_colname:rcol_ind,
 :valid_colname:vcol_ind,
 :overrideType:ovtype_ind,:override:ov_ind,
 :max_row:maxrow_ind,:num_row:numrow_ind,
 :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
 else
 EXEC SQL COMMIT;
}

```

---

## Decomposition stored procedures

The decomposition stored procedures `dxxInsertXML()` and `dxxShredXML()` are used to break down or shred incoming XML documents and to store data in new or existing database tables. The `dxxInsertXML()` stored procedure takes an enabled XML collection name as input. The `dxxShredXML()` stored procedure takes a DAD file as input; it does not require an enabled XML collection.



## dxxShredXML()

### Purpose

Decomposes XML documents, based on a DAD file mapping, storing the content of the XML elements and attributes in specified DB2 tables. In order for dxxShredXML() to work, all tables specified in the DAD file must exist, and all columns and their data types that are specified in the DAD must be consistent with the existing tables. The stored procedure requires that the columns specified in the join condition, in the DAD, correspond to primary- foreign key relationships in the existing tables. The join condition columns that are specified in the RDB\_node of the root element\_node must exist in the tables.

The stored procedure fragment in this section is a sample for explanation purposes. A complete, working sample is located in DXXSAMPLES/QCSRC(X).

### Format

```
dxxShredXML(CLOB(100K) DAD, /* input */
 CLOB(1M) xmlobj, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

## Parameters

Table 56. *dxxShredXML()* parameters

Parameter	Description	IN/OUT parameter
<i>DAD</i>	A CLOB containing the DAD file.	IN
<i>xmlobj</i>	An XML document object in XMLCLOB type.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## Examples

The following fragment is an example of a call to *dxxShredXML()*. A complete, working sample is located in *DXXSAMPLES/QCSRC(SHDX)*.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
SQL TYPE IS CLOB(100K) dad; /* DAD */
EXEC SQL DECLARE :dad VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running from USS */
/* to ensure that DB2 converts the code page correctly*/
SQL TYPE IS CLOB(100K) xmlDoc; /* input xml document */
EXEC SQL DECLARE :xmlDoc VARIABLE CCSID 1047;
/* specifies the CCSID for DAD when running */
/* from USS to ensure that DB2 converts the */
/* code page correctly */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short dad_ind;
short xmlDoc_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) &dad.data;
, 1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf("Error reading dad file getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
 dad.length = file_length;
}
else {
 printf("Error opening dad file \n");
 rc = -1;
 goto exit;
}

/* Initialize the XML CLOB object. */
file_handle = fopen("/dxx/samples/xml/getstart_xcollection.xml", "r");
if (file_handle != NULL) {
```

```

 file_length = fread ((void *) &xmlDoc.data;
, 1, FILE_SIZE,
 file_handle);
 if (file_length == 0) {
 printf ("Error reading xml file getstart_xcollection.xml \n");
 rc = -1;
 goto exit;
 } else
 xmlDoc.length = file_length;
}
else {
 printf("Error opening xml file \n");
 rc = -1;
 goto exit;
}

/* initialize host variable and indicators */
returnCode = 0;
msg_txt[0] = '\0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXSHRED" (:dad:dad_ind;
 :xmlDoc:xmlDoc_ind,
 :returnCode:returnCode_ind,
 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
 else
 EXEC SQL COMMIT;
}

exit:
 return rc;

```

## **dxxInsertXML()**

### **Purpose**

Takes two input parameters, the name of an enabled XML collection and the XML document that are to be decomposed, and returns two output parameters, a return code and a return message.

### **Format**

```
dxxInsertXML(char(32) collectionName, /* input */
 CLOB(1M) xmlobj, /* input */
 long returnCode, /* output */
 varchar(1024) returnMsg) /* output */
```

## Parameters

Table 57. *dxxInsertXML()* parameters

Parameter	Description	IN/OUT parameter
<i>collectionName</i>	The name of an enabled XML collection.	IN
<i>xmlobj</i>	An XML document object in CLOB type.	IN
<i>returnCode</i>	The return code from the stored procedure.	OUT
<i>returnMsg</i>	The message text that is returned in case of error.	OUT

## Examples

In the following fragment example, the *dxxInsertXML()* call decomposes the input XML document *dxxinstall/xml/order1.xml* and inserts data into the SALES\_ORDER collection tables according to the mapping that is specified in the DAD file with which it was enabled with. A complete, working sample is located in DXXSAMPLES/QCSRC(INSX).

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
char collectionName[32]; /* name of an XML collection */
SQL TYPE is CLOB(100K) xmlDoc; /* input xml document */
long returnCode; /* return error code */
char returnMsg[1024]; /* error message text */
short collectionName_ind;
short xmlDoc_ind;
short returnCode_ind;
short returnMsg_ind;
EXEC SQL END DECLARE SECTION;

FILE *file_handle;
long file_length=0;

/* initialize the DAD CLOB object. */
file_handle = fopen("/dxx/samples/dad/getstart_xcollection.dad", "r");
if (file_handle != NULL) {
 file_length = fread ((void *) , &dad.data;
1, FILE_SIZE, file_handle);
 if (file_length == 0) {
 printf("Error reading dad file getstart_xcollection.dad\n");
 rc = -1;
 goto exit;
 } else
 dad.length = file_length;
}
else {
 printf("Error opening dad file \n");
 rc = -1;
 goto exit;
}

/* initialize host variable and indicators */
strcpy(collectionName, "sales_ord");
returnCode = 0;
msg_txt[0] = '\0';
collectionName_ind = 0;
xmlDoc_ind = 0;
```

```
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL "DB2XML.DXXINSERTXML" (:collection_name:collection_name_ind,
 :xmlDoc:xmlDoc_ind,
 :returnCode:returnCode_ind,
 :returnMsg:returnMsg_ind);

if (SQLCODE < 0) {
 EXEC SQL ROLLBACK;
else
 EXEC SQL COMMIT;
}

exit:
 return rc;
```

---

## Chapter 14. Administration support tables

When a database is enabled, a DTD reference table, DTD\_REF, and an XML\_USAGE table are created. The DTD\_REF table contains information about all of the DTDs. The XML\_USAGE table stores common information for each XML-enabled column. Each is created with specific PUBLIC privileges.

The parameter limits listed in the support tables are also documented in “Appendix D. The XML Extender limits” on page 263.

---

### DTD reference table

The XML Extender also serves as an XML DTD repository. When a database is XML-enabled, a DTD reference table, DTD\_REF, is created. Each row of this table represents a DTD with additional metadata information. Users can access this table, and insert their own DTDs. The DTDs in the DTD\_REF table are used to validate XML documents and to help applications to define a DAD file. It has the schema name of DB2XML. A DTD\_REF table can have the columns shown in Table 58.

Table 58. DTD\_REF table

Column name	Data type	Description
DTDID	VARCHAR(128)	The primary key (unique and not NULL). It is used to identify the DTD. When it is specified in the DAD file, the DAD file must follow the schema that is defined by the DTD.
CONTENT	XMLCLOB	The content of the DTD.
ROW_ID	ROWID	An identifier of the row.
USAGE_COUNT	INTEGER	The number of XML columns and XML collections in the database that use the DTD to define their DAD files.
AUTHOR	VARCHAR(128)	The author of the DTD, optional information for the user to input.
CREATOR	VARCHAR(128)	The user ID that does the first insertion. The CREATOR column is optional.
UPDATOR	VARCHAR(128)	The user ID that does the last update. The UPDATOR column is optional.

**Restriction:** The DTD can be modified by the application only when the USAGE\_COUNT is zero.

#### Privileges granted to PUBLIC

Privileges of INSERT, UPDATE, DELETE, and SELECT are granted for PUBLIC.

---

### XML usage table

Stores common information for each XML-enabled column. The XML\_USAGE table's schema name is DB2XML, and its primary key is (*table\_name*, *col\_name*). Only read privileges of this table are granted to PUBLIC. An XML\_USAGE table is created at the time the database is enabled with the columns listed in Table 59 on page 214.

Table 59. XML\_USAGE table

Column name	Description
table_schema	For XML column, the schema name of the user table that contains an XML column. For XML collection, a value of "DXX_COLL" as the default schema name.
table_name	For XML column, the name of the user table that contains an XML column. For XML collection, a value "DXX_COLLECTION," which identifies the entity as a collection.
col_name	The name of the XML column or XML collection. It is part of the composite key along with the table_name.
DTDID	A string associating a DTD inserted into DTD_REF with a DTD specifid in a DAD file; this value must match the value of the DTDID element in the DAD. This column is a foreign key.
DAD	The content of the DAD file that is associated with the column or collection.
row_id	An identifier of the row.
access_mode	Specifies which access mode is used: 1 for XML collection, 0 for XML column
default_view	Stores the default view name if there is one.
trigger_suffix	Not NULL. For unique trigger names.
validation	1 for yes, 0 for no.

Do not add, modify or delete entries from the XML\_USAGE table; it is for XML Extender internal use only.

#### Privileges granted to PUBLIC

For XML\_USAGE, the privilege of SELECT is granted for PUBLIC. INSERT, DELETE, and UPDATE are granted to DB2XML.



---

## Chapter 15. Troubleshooting

All embedded SQL statements in your program and DB2 command line interface (CLI) calls in your program, including those that invoke the DB2 XML Extender user-defined functions (UDFs), generate codes that indicate whether the embedded SQL statement or DB2 CLI call executed successfully.

Your program can retrieve information that supplements these codes. This includes SQLSTATE information and error messages. You can use this diagnostic information to isolate and fix problems in your program.

Occasionally the source of a problem cannot be easily diagnosed. In these cases, you might need to provide information to your Software Support Provider to isolate and fix the problem. The XML Extender includes a trace facility that records the XML Extender activity. The trace information can be valuable input to Software Service Provider. You should use the trace facility only under instruction from Software Service Provider.

This chapter describes how to access this diagnostic information. It describes:

- How to handle XML Extender UDF return codes.
- How to control tracing

It also lists and describes the SQLSTATE codes and error messages that the XML Extender might return.

---

### Handling UDF return codes

Embedded SQL statements return codes in the SQLCODE, SQLWARN, and SQLSTATE fields of the SQLCA structure. This structure is defined in an SQLCA INCLUDE file. (For more information about the SQLCA structure and SQLCA INCLUDE file, see the *DB2 Application Development Guide* *DB2 Application Programming and SQL Guide*.)

DB2 CLI calls return SQLCODE and SQLSTATE values that you can retrieve using the SQLERROR function. (For more information about retrieving error information with the SQLERROR function, see the *CLI Guide and Reference* *ODBC Guide and Reference*.)

An SQLCODE value of 0 means that the statement ran successfully (with possible warning conditions). A positive SQLCODE value means that the statement ran successfully but with a warning. (Embedded SQL statements return information about the warning that is associated with 0 or positive SQLCODE values in the SQLWARN field.) A negative SQLCODE value means that an error occurred.

DB2 associates a message with each SQLCODE value. If an XML Extender UDF encounters a warning or error condition, it passes associated information to DB2 for inclusion in the SQLCODE message.

Embedded SQL statements and DB2 CLI calls that invoke the DB2 XML Extender UDFs might return SQLCODE messages and SQLSTATE values that are unique to these UDFs, but DB2 returns these values in the same way as it does for other embedded SQL statements or other DB2 CLI calls. Thus, the way you access these values is the same as for embedded SQL statements or DB2 CLI calls that do not start the DB2 XML Extender UDFs.

See “SQLSTATE codes” on page 217 for the SQLSTATE values and the message number of associated messages that can be returned by the XML Extender. See “Messages” on page 221 for information about each message.

---

## Handling stored procedure return codes

The XML Extender provides return codes to help resolve problems with stored procedures. When you receive a return code from a stored procedure, check the following file, which matches the return code with an XML Extender error message number and the symbolic constant.

`DXX_INSTALL/include/dxxrc.h`

You can reference the error message number in “Messages” on page 221 and use the diagnostic information in the explanation.

---

## SQLSTATE codes

Table 60 lists and describes the SQLSTATE values that the XML Extender returns. The description of each SQLSTATE value includes its symbolic representation. The table also lists the message number that is associated with each SQLSTATE value. See “Messages” on page 221 for information about each message.

*Table 60. SQLSTATE codes and associated message numbers*

SQLSTATE	Message No.	Description
00000	DXXnnnnl	No error has occurred.
01HX0	DXXD003W	The element or attribute specified in the path expression is missing from the XML document.
38X00	DXXC000E	The XML Extender is unable to open the specified file.
38X01	DXXA072E	XML Extender tried to automatically bind the database before enabling it, but could not find the bind files
	DXXC001E	The XML Extender could not find the file specified.
38X02	DXXC002E	The XML Extender is unable to read data from the specified file.
38X03	DXXC003E	The XML Extender is unable to write data to the file.
	DXXC011E	The XML Extender is unable to write data to the trace control file.
38X04	DXXC004E	The XML Extender was unable to operate the specified locator.
38X05	DXXC005E	The file size is greater than the XMLVarchar size and the XML Extender is unable to import all the data from the file.
38X06	DXXC006E	The file size is greater than the size of the XMLCLOB and the XML Extender is unable to import all the data from the file.
38X07	DXXC007E	The number of bytes in the LOB Locator does not equal the file size.
38X08	DXXD001E	A scalar extraction function used a location path that occurs multiple times. A scalar function can only use a location path that does not have multiple occurrence.
38X09	DXXD002E	The path expression is syntactically incorrect.
38X10	DXXG002E	The XML Extender was unable to allocate memory from the operating system.
38X11	DXXA009E	This stored procedure is for XML Column only.

Table 60. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38X12	DXXA010E	While attempting to enable the column, the XML Extender could not find the DTD ID, which is the identifier specified for the DTD in the document access definition (DAD) file.
38X14	DXXD000E	There was an attempt to store an invalid document into a table. Validation has failed.
38X15	DXXA056E	The validation element in document access definition (DAD) file is wrong or missing.
	DXXA057E	The name attribute of a side table in the document access definition (DAD) file is wrong or missing.
	DXXA058E	The name attribute of a column in the document access definition (DAD) file is wrong or missing.
	DXXA059E	The type attribute of a column in the document access definition (DAD) file is wrong or missing.
	DXXA060E	The path attribute of a column in the document access definition (DAD) file is wrong or missing.
	DXXA061E	The multi_occurrence attribute of a column in the document access definition (DAD) file is wrong or missing.
	DXXQ000E	A mandatory element is missing from the document access definition (DAD) file.
38X16	DXXG004E	A null value for a required parameter was passed to an XML stored procedure.
38X17	DXXQ001E	The SQL statement in the document access definition (DAD) or the one that overrides it is not valid. A SELECT statement is required for generating XML documents.
38X18	DXXG001E	XML Extender encountered an internal error.
	DXXG006E	XML Extender encountered an internal error while using CLI.
38X19	DXXQ002E	The system is running out of space in memory or disk. There is no space to contain the resulting XML documents.

Table 60. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38X20	DXXQ003W	The user-defined SQL query generates more XML documents than the specified maximum. Only the specified number of documents are returned.
38X21	DXXQ004E	The specified column is not one of the columns in the result of the SQL query.
38X22	DXXQ005E	The mapping of the SQL query to XML is incorrect.
38X23	DXXQ006E	An attribute_node element in the document access definition(DAD) file does not have a name attribute.
38X24	DXXQ007E	The attribute_node element in the document access definition (DAD) does not have a column element or RDB_node.
38X25	DXXQ008E	A text_node element in the document access definition (DAD) file does not have a column element.
38X26	DXXQ009E	The specified result table could not be found in the system catalog.
38X27	DXXQ010E	The RDB_node of the attribute_node or text_node must have a table.
	DXXQ011E	The RDB_node of the attribute_node or text_node must have a column.
	DXXQ017E	An XML document generated by the XML Extender is too large to fit into the column of the result table.
38X28	DXXQ012E	XML Extender could not find the expected element while processing the DAD.
	DXXQ016E	All tables must be defined in the RDB_node of the top element in the document access definition (DAD) file. Sub-element tables must match the tables defined in the top element. The table name in this RDB_node is not in the top element.
38X29	DXXQ013E	The element table or column must have a name in the document access definition (DAD) file.
	DXXQ015E	The condition in the condition element in the document access definition (DAD) has an invalid format.

Table 60. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38X30	DXXQ014E	An element_node element in the document access definition (DAD) file does not have a name attribute.
	DXXQ018E	The ORDER BY clause is missing from the SQL statement in a document access definition (DAD) file that maps SQL to XML.
38X31	DXXQ019E	The objids element does not have a column element in the document access definition (DAD) file that maps SQL to XML.
38X36	DXXA073E	The database was not bound when user tried to enable it.
38X37	DXXG007E	The server operating system locale is inconsistent with DB2 code page.
38X38	DXXG008E	The server operating system locale can not be found in the code page table.
38x33	DXXG005E	This parameter is not supported in this release, will be supported in the future release.
38x34	DXXG000E	An invalid file name was specified.

---

## Messages

The XML Extender provides error messages to help with problem determination.

### Error messages

The XML Extender generates the following messages when it completes an operation or detects an error.

---

**DXXA000I**    **Enabling column** *<column\_name>*.  
**Please Wait.**

**Explanation:** This is an informational messages.

**User Response:** No action required.

---

**DXXA001S**    **An unexpected error occurred in build**  
*<build\_ID>*, **file** *<file\_name>*, **and line**  
*<line\_number>*.

**Explanation:** An unexpected error occurred.

**User Response:** If this error persists, contact your Software Service Provider. When reporting the error, be sure to include all the message text, the trace file, and an explanation of how to reproduce the problem.

---

**DXXA002I**    **Connecting to database** *<database>*.

**Explanation:** This is an informational message.

**User Response:** No action required.

---

**DXXA003E**    **Cannot connect to database**  
*<database>*.

**Explanation:** The database specified might not exist or could be corrupted.

**User Response:**

1. Ensure the database is specified correctly.
2. Ensure the database exists and is accessible.
3. Determine if the database is corrupted. If it is, ask your database administrator to recover it from a backup.

---

**DXXA004E**    **Cannot enable database** *<database>*.

**Explanation:** The database might already be enabled or might be corrupted.

**User Response:**

1. Determine if the database is enabled.
2. Determine if the database is corrupted. If it is, ask your database administrator to recover it from a backup.

---

**DXXA005I**    **Enabling database** *<database>*. **Please wait.**

**Explanation:** This is an informational message.

**User Response:** No action required.

---

**DXXA006I**    **The database** *<database>* **was enabled successfully.**

**Explanation:** This is an informational message.

**User Response:** No action required.

---

**DXXA007E**    **Cannot disable database** *<database>*.

**Explanation:** The database cannot be disabled by XML Extender if it contains any XML columns or collections.

**User Response:** Backup any important data, disable any XML columns or collections, and update or drop any tables until there are no XML data types left in the database.

---

**DXXA008I**    **Disabling column** *<column\_name>*.  
**Please Wait.**

**Explanation:** This is an information message.

**User Response:** No action required.

---

**DXXA009E**    **Xcolumn tag is not specified in the DAD file.**

**Explanation:** This stored procedure is for XML Column only.

**User Response:** Ensure the Xcolumn tag is specified correctly in the DAD file.

---

**DXXA010E**    **Attempt to find DTD ID** *<dtid>* **failed.**

**Explanation:** While attempting to enable the column, the XML Extender could not find the DTD ID, which is the identifier specified for the DTD in the document access definition (DAD) file.

**User Response:** Ensure the correct value for the DTD ID is specified in the DAD file.

---

**DXXA011E Inserting a record into DB2XML.XML\_USAGE table failed.**

**Explanation:** While attempting to enable the column, the XML Extender could not insert a record into the DB2XML.XML\_USAGE table.

**User Response:** Ensure the DB2XML.XML\_USAGE table exists and that a record by the same name does not already exist in the table.

---

**DXXA012E Attempt to update DB2XML.DTD\_REF table failed.**

**Explanation:** While attempting to enable the column, the XML Extender could not update the DB2XML.DTD\_REF table.

**User Response:** Ensure the DB2XML.DTD\_REF table exists. Determine whether the table is corrupted or if the administration user ID has the correct authority to update the table.

---

**DXXA013E Attempt to alter table <table\_name> failed.**

**Explanation:** While attempting to enable the column, the XML Extender could not alter the specified table.

**User Response:** Check the privileges required to alter the table.

---

**DXXA014E The specified root ID column: <root\_id> is not a single primary key of table <table\_name>.**

**Explanation:** The root ID specified is either not a key, or it is not a single key of table *table\_name*.

**User Response:** Ensure the specified root ID is the single primary key of the table.

---

**DXXA015E The column DXXROOT\_ID already exists in table <table\_name>.**

**Explanation:** The column DXXROOT\_ID exists, but was not created by XML Extender.

**User Response:** Specify a primary column for the root ID option when enabling a column, using a different different column name.

---

**DXXA016E The input table <table\_name> does not exist.**

**Explanation:** The XML Extender was unable to find the specified table in the system catalog.

**User Response:** Ensure that the table exists in the database, and is specified correctly.

---

**DXXA017E The input column <column\_name> does not exist in the specified table <table\_name>.**

**Explanation:** The XML Extender was unable to find the column in the system catalog.

**User Response:** Ensure the column exists in a user table.

---

**DXXA018E The specified column is not enabled for XML data.**

**Explanation:** While attempting to disable the column, XML Extender could not find the column in the DB2XML.XML\_USAGE table, indicating that the column is not enabled. If the column is not XML-enabled, you do not need to disable it.

**User Response:** No action required.

---

**DXXA019E A input parameter required to enable the column is null.**

**Explanation:** A required input parameter for the enable\_column() stored procedure is null.

**User Response:** Check all the input parameters for the enable\_column() stored procedure.

---

**DXXA020E Columns cannot be found in the table <table\_name>.**

**Explanation:** While attempting to create the default view, the XML Extender could not find columns in the specified table.

**User Response:** Ensure the column and table name are specified correctly.

---

**DXXA021E Cannot create the default view <default\_view>.**

**Explanation:** While attempting to enable a column, the XML Extender could not create the specified view.

**User Response:** Ensure that the default view name is unique. If a view with the name already exists, specify a unique name for the default view.

---

**DXXA022I Column <column\_name> enabled.**

**Explanation:** This is an informational message.

**User Response:** No response required.

---

**DXXA023E Cannot find the DAD file.**

**Explanation:** While attempting to disable a column, the XML Extender was unable to find the document access definition (DAD) file.

**User Response:** Ensure you specified the correct database name, table name, or column name.

---



---

**DXXA024E** The XML Extender encountered an internal error while accessing the system catalog tables.

**Explanation:** The XML Extender was unable to access system catalog table.

**User Response:** Ensure the database is in a stable state.

---

**DXXA025E** Cannot drop the default view  
<default\_view>.

**Explanation:** While attempting to disable a column, the XML Extender could not drop the default view.

**User Response:** Ensure the administration user ID for XML Extender has the privileges necessary to drop the default view.

---

**DXXA026E** Unable to drop the side table  
<side\_table>.

**Explanation:** While attempting to disable a column, the XML Extender was unable to drop the specified table.

**User Response:** Ensure that the administrator user ID for XML Extender has the privileges necessary to drop the table.

---

**DXXA027E** Could not disable the column.

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA028E** Could not disable the column.

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA029E** Could not disable the column.

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

---

**DXXA030E** Could not disable the column.

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA031E** Unable to reset the DXXROOT\_ID column value in the application table to NULL.

**Explanation:** While attempting to disable a column, the XML Extender was unable to set the value of DXXROOT\_ID in the application table to NULL.

**User Response:** Ensure that the administrator user ID for XML Extender has the privileges necessary to alter the application table.

---

**DXXA032E** Decrement of USAGE\_COUNT in DB2XML.XML\_USAGE table failed.

**Explanation:** While attempting to disable the column, the XML Extender was unable to reduce the value of the USAGE\_COUNT column by one.

**User Response:** Ensure that the DB2XML.XML\_USAGE table exists and that the administrator user ID for XML Extender has the necessary privileges to update the table.

---

**DXXA033E** Attempt to delete a row from the DB2XML.XML\_USAGE table failed.

**Explanation:** While attempting to disable a column, the XML Extender was unable to delete its associate row in the DB2XML.XML\_USAGE table.

**User Response:** Ensure that the DB2XML.XML\_USAGE table exists and that the administration user ID for XML Extender has the privileges necessary to update this table.

---

**DXXA034I** XML Extender has successfully disabled column <column\_name>.

**Explanation:** This is an informational message

**User Response:** No action required.

---

**DXXA035I** XML Extender is disabling database <database>. Please wait.

**Explanation:** This is an informational message.

**User Response:** No action is required.

---

---

**DXXA036I XML Extender has successfully disabled database <database>.**

**Explanation:** This is an informational message.

**User Response:** No action is required.

---

**DXXA037E The specified table space name is longer than 18 characters.**

**Explanation:** The table space name cannot be longer than 18 alphanumeric characters.

**User Response:** Specify a name less than 18 characters.

---

**DXXA038E The specified default view name is longer than 18 characters.**

**Explanation:** The default view name cannot be longer than 18 alphanumeric characters.

**User Response:** Specify a name less than 18 characters.

---

**DXXA039E The specified ROOT\_ID name is longer than 18 characters.**

**Explanation:** The ROOT\_ID name cannot be longer than 18 alphanumeric characters.

**User Response:** Specify a name less than 18 characters.

---

**DXXA046E Unable to create the side table <side\_table>.**

**Explanation:** While attempting to enable a column, the XML Extender was unable to create the specified side table.

**User Response:** Ensure that the administrator user ID for XML Extender has the privileges necessary to create the side table.

---

**DXXA047E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA048E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem

persists, contact your Software Service Provider and provide the trace file.

---

**DXXA049E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA050E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA051E Could not disable the column.**

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA052E Could not disable the column.**

**Explanation:** XML Extender could not disable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA053E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

---

**DXXA054E Could not enable the column.**

**Explanation:** XML Extender could not enable a column because an internal trigger failed. Possible causes:

**User Response:** Use the trace facility to create a trace file and try to correct the problem. If the problem persists, contact your Software Service Provider and provide the trace file.

---

**DXXA056E The validation value *<validation\_value>* in the DAD file is invalid.**

**Explanation:** The validation element in document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the validation element is specified correctly in the DAD file.

---

**DXXA057E A side table name *<side\_table\_name>* in DAD is invalid.**

**Explanation:** The name attribute of a side table in the document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the name attribute of a side table is specified correctly in the DAD file.

---

**DXXA058E A column name *<column\_name>* in the DAD file is invalid.**

**Explanation:** The name attribute of a column in the document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the name attribute of a column is specified correctly in the DAD file.

---

**DXXA059E The type *<column\_type>* of column *<column\_name>* in the DAD file is invalid.**

**Explanation:** The type attribute of a column in the document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the type attribute of a column is specified correctly in the DAD file.

---

**DXXA060E The path attribute *<location\_path>* of *<column\_name>* in the DAD file is invalid.**

**Explanation:** The path attribute of a column in the document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the path attribute of a column is specified correctly in the DAD file.

---

**DXXA061E The multi\_occurrence attribute *<multi\_occurrence>* of *<column\_name>* in the DAD file is invalid.**

**Explanation:** The multi\_occurrence attribute of a column in the document access definition (DAD) file is wrong or missing.

**User Response:** Ensure that the multi\_occurrence attribute of a column is specified correctly in the DAD file.

---

**DXXA062E Unable to retrieve the column number for *<column\_name>* in table *<table\_name>*.**

**Explanation:** XML Extender could not retrieve the column number for *column\_name* in table *table\_name* from the system catalog.

**User Response:** Make sure the application table is well defined.

---

**DXXA063I Enabling collection *<collection\_name>*. Please Wait.**

**Explanation:** This is an information message.

**User Response:** No action required.

---

**DXXA064I Disabling collection *<collection\_name>*. Please Wait.**

**Explanation:** This is an information message.

**User Response:** No action required.

---

**DXXA065E Calling stored procedure *<procedure\_name>* failed.**

**Explanation:** Check the shared library db2xml and see if the permission is correct.

**User Response:** Make sure the client has permission to run the stored procedure.

---

**DXXA066I XML Extender has successfully disabled collection *<collection\_name>*.**

**Explanation:** This is an informational message.

**User Response:** No response required.

---

**DXXA067I XML Extender has successfully enabled collection *<collection\_name>*.**

**Explanation:** This is an informational message.

**User Response:** No response required.

---

---

**DXXA068I XML Extender has successfully turned the trace on.**

**Explanation:** This is an informational message.

**User Response:** No response required.

---

**DXXA069I XML Extender has successfully turned the trace off.**

**Explanation:** This is an informational message.

**User Response:** No response required.

---

**DXXA070W The database has already been enabled.**

**Explanation:** The enable database command was executed on the enabled database

**User Response:** No action is required.

---

**DXXA071W The database has already been disabled.**

**Explanation:** The disable database command was executed on the disabled database

**User Response:** No action is required.

---

**DXXA072E XML Extender couldn't find the bind files. Bind the database before enabling it.**

**Explanation:** XML Extender tried to automatically bind the database before enabling it, but could not find the bind files

**User Response:** Bind the database before enabling it.

---

**DXXA073E The database is not bound. Please bind the database before enabling it.**

**Explanation:** The database was not bound when user tried to enable it.

**User Response:** Bind the database before enabling it.

---

**DXXA074E Wrong parameter type. The stored procedure expects a STRING parameter.**

**Explanation:** The stored procedure expects a STRING parameter.

**User Response:** Declare the input parameter to be STRING type.

---

**DXXA075E Wrong parameter type. The input parameter should be a LONG type.**

**Explanation:** The stored procedure expects the input parameter to be a LONG type.

---

**User Response:** Declare the input parameter to be a LONG type.

---

**DXXA076E XML Extender trace instance ID invalid.**

**Explanation:** Cannot start trace with the instance ID provided.

**User Response:** Ensure that the instance ID is a valid AS/400 user ID.

---

**DXXC000E Unable to open the specified file.**

**Explanation:** The XML Extender is unable to open the specified file.

**User Response:** Ensure that the application user ID has read and write permission for the file.

---

**DXXC001E The specified file is not found.**

**Explanation:** The XML Extender could not find the file specified.

**User Response:** Ensure that the file exists and the path is specified correctly.

---

**DXXC002E Unable to read file.**

**Explanation:** The XML Extender is unable to read data from the specified file.

**User Response:** Ensure that the application user ID has read permission for the file.

---

**DXXC003E Unable to write to the specified file.**

**Explanation:** The XML Extender is unable to write data to the file.

**User Response:** Ensure that the application user ID has write permission for the file or that the file system has sufficient space.

---

**DXXC004E Unable to operate the LOB Locator:  
rc=<locator\_rc>.**

**Explanation:** The XML Extender was unable to operate the specified locator.

**User Response:** Ensure the LOB Locator is set correctly.

---

**DXXC005E Input file size is greater than XMLVarchar size.**

**Explanation:** The file size is greater than the XMLVarchar size and the XML Extender is unable to import all the data from the file.

**User Response:** Use the XMLCLOB column type.

---

---

**DXXC006E The input file exceeds the DB2 LOB limit.**

**Explanation:** The file size is greater than the size of the XMLCLOB and the XML Extender is unable to import all the data from the file.

**User Response:** Decompose the file into smaller objects or use an XML collection.

---

**DXXC007E Unable to retrieve data from the file to the LOB Locator.**

**Explanation:** The number of bytes in the LOB Locator does not equal the file size.

**User Response:** Ensure the LOB Locator is set correctly.

---

**DXXC008E Can not remove the file <file\_name>.**

**Explanation:** The file has a sharing access violation or is still open.

**User Response:** Close the file or stop any processes that are holding the file. You might have to stop and restart DB2.

---

**DXXC009E Unable to create file to <directory> directory.**

**Explanation:** The XML Extender is unable to create a file in directory *directory*.

**User Response:** Ensure that the directory exists, that the application user ID has write permission for the directory, and that the file system has sufficient space for the file.

---

**DXXC010E Error while writing to file <file\_name>.**

**Explanation:** There was an error while writing to the file *file\_name*.

**User Response:** Ensure that the file system has sufficient space for the file.

---

**DXXC011E Unable to write to the trace control file.**

**Explanation:** The XML Extender is unable to write data to the trace control file.

**User Response:** Ensure that the application user ID has write permission for the file or that the file system has sufficient space.

---

**DXXC012E Cannot create temporary file.**

**Explanation:** Cannot create file in system temp directory.

**User Response:** Ensure that the application user ID has write permission for the file system temp directory or that the file system has sufficient space for the file.

---

---

**DXXD000E An invalid XML document is rejected.**

**Explanation:** There was an attempt to store an invalid document into a table. Validation has failed.

**User Response:** Check the document with its DTD using an editor that can view invisible invalid characters. To suppress this error, turn off validation in the DAD file.

---

**DXXD001E <location\_path> occurs multiple times.**

**Explanation:** A scalar extraction function used a location path that occurs multiple times. A scalar function can only use a location path that does not have multiple occurrence.

**User Response:** Use a table function (add an 's' to the end of the scalar function name).

---

**DXXD002E A syntax error occurred near position <position> in the search path.**

**Explanation:** The path expression is syntactically incorrect.

**User Response:** Correct the search path argument of the query. Refer to the documentation for the syntax of path expressions.

---

**DXXD003W Path not found. Null is returned.**

**Explanation:** The element or attribute specified in the path expression is missing from the XML document.

**User Response:** Verify that the specified path is correct.

---

**DXXG000E The file name <file\_name> is invalid.**

**Explanation:** An invalid file name was specified.

**User Response:** Specify a correct file name and try again.

---

**DXXG001E An internal error occurred in build <build\_ID>, file <file\_name>, and line <line\_number>.**

**Explanation:** XML Extender encountered an internal error.

**User Response:** Contact your Software Service Provider. When reporting the error, be sure to include all the messages, the trace file and how to reproduce the error.

---

**DXXG002E The system is out of memory.**

**Explanation:** The XML Extender was unable to allocate memory from the operating system.

**User Response:** Close some applications and try again. If the problem persists, refer to your operating system documentation for assistance. Some operating

---

systems might require that you reboot the system to correct the problem.

---

**DXXG004E Invalid null parameter.**

**Explanation:** A null value for a required parameter was passed to an XML stored procedure.

**User Response:** Check all required parameters in the argument list for the stored procedure call.

---

**DXXG005E Parameter not supported.**

**Explanation:** This parameter is not supported in this release, will be supported in the future release.

**User Response:** Set this parameter to NULL.

---

**DXXG006E Internal Error CLISTATE=<clistate>, RC=<cli\_rc>, build <build\_ID>, file <file\_name>, line <line\_number> CLIMSG=<CLI\_msg>.**

**Explanation:** XML Extender encountered an internal error while using CLI.

**User Response:** Contact your Software Service Provider. Potentially this error can be caused by incorrect user input. When reporting the error, be sure to include all output messages, trace log, and how to reproduce the problem. Where possible, send any DADs, XML documents, and table definitions which apply.

---

**DXXG007E Locale <locale> is inconsistent with DB2 code page <code\_page>.**

**Explanation:** The server operating system locale is inconsistent with DB2 code page.

**User Response:** Correct the server operating system locale and restart DB2.

---

**DXXG008E Locale <locale> is not supported.**

**Explanation:** The server operating system locale can not be found in the code page table.

**User Response:** Correct the server operating system locale and restart DB2.

---

**DXXQ000E <Element> is missing from the DAD file.**

**Explanation:** A mandatory element is missing from the document access definition (DAD) file.

**User Response:** Add the missing element to the DAD file.

---

**DXXQ001E Invalid SQL statement for XML generation.**

**Explanation:** The SQL statement in the document access definition (DAD) or the one that overrides it is not valid. A SELECT statement is required for generating XML documents.

**User Response:** Correct the SQL statement.

---

**DXXQ002E Cannot generate storage space to hold XML documents.**

**Explanation:** The system is running out of space in memory or disk. There is no space to contain the resulting XML documents.

**User Response:** Limit the number of documents to be generated. Reduce the size of each documents by removing some unnecessary element and attribute nodes from the document access definition (DAD) file.

---

**DXXQ003W Result exceeds maximum.**

**Explanation:** The user-defined SQL query generates more XML documents than the specified maximum. Only the specified number of documents are returned.

**User Response:** No action is required. If all documents are needed, specify zero as the maximum number of documents.

---

**DXXQ004E The column <column\_name> is not in the result of the query.**

**Explanation:** The specified column is not one of the columns in the result of the SQL query.

**User Response:** Change the specified column name in the document access definition (DAD) file to make it one of the columns in the result of the SQL query. Alternatively, change the SQL query so that it has the specified column in its result.

---

**DXXQ004W The DTD ID was not found in the DAD.**

**Explanation:** In the DAD, VALIDATION is YES but the DTDID element is not specified. NO validation check is performed.

**User Response:** No action is required. If validation is needed, specify the DTDID element in the DAD file.

---

**DXXQ005E Wrong relational mapping. The element <element\_name> is at a lower level than its child column <column\_name>.**

**Explanation:** The mapping of the SQL query to XML is incorrect.

**User Response:** Make sure that the columns in the result of the SQL query are in a top-down order of the relational hierarchy. Also make sure that there is a

single-column candidate key to begin each level. If such a key is not available in a table, the query should generate one for that table using a table expression and the DB2 built-in function generate\_unique().

---

**DXXQ006E An attribute\_node element has no name.**

**Explanation:** An attribute\_node element in the document access definition(DAD) file does not have a name attribute.

**User Response:** Ensure that every attribute\_node has a name in the DAD file.

---

**DXXQ007E The attribute\_node <attribute\_name> has no column element or RDB\_node.**

**Explanation:** The attribute\_node element in the document access definition (DAD) does not have a column element or RDB\_node.

**User Response:** Ensure that every attribute\_node has a column element or RDB\_node in the DAD.

---

**DXXQ008E A text\_node element has no column element.**

**Explanation:** A text\_node element in the document access definition (DAD) file does not have a column element.

**User Response:** Ensure that every text\_node has a column element in the DAD.

---

**DXXQ009E Result table <table\_name> does not exist.**

**Explanation:** The specified result table could not be found in the system catalog.

**User Response:** Create the result table before calling the stored procedure.

---

**DXXQ010E RDB\_node of <node\_name> does not have a table in the DAD file.**

**Explanation:** The RDB\_node of the attribute\_node or text\_node must have a table.

**User Response:** Specify the table of RDB\_node for attribute\_node or text\_node in the document access definition (DAD) file.

---

**DXXQ011E RDB\_node element of <node\_name> does not have a column in the DAD file.**

**Explanation:** The RDB\_node of the attribute\_node or text\_node must have a column.

**User Response:** Specify the column of RDB\_node for

attribute\_node or text\_node in the document access definition (DAD) file.

---

**DXXQ012E Errors occurred in DAD.**

**Explanation:** XML Extender could not find the expected element while processing the DAD.

**User Response:** Check that the DAD is a valid XML document and contains all the elements required by the DAD DTD. Consult the XML Extender publication for the DAD DTD.

---

**DXXQ013E The table or column element does not have a name in the DAD file.**

**Explanation:** The element table or column must have a name in the document access definition (DAD) file.

**User Response:** Specify the name of table or column element in the DAD.

---

**DXXQ014E An element\_node element has no name.**

**Explanation:** An element\_node element in the document access definition (DAD) file does not have a name attribute.

**User Response:** Ensure that every element\_node element has a name in the DAD file.

---

**DXXQ015E The condition format is invalid.**

**Explanation:** The condition in the condition element in the document access definition (DAD) has an invalid format.

**User Response:** Ensure that the format of the condition is valid.

---

**DXXQ016E The table name in this RDB\_node is not defined in the top element of the DAD file.**

**Explanation:** All tables must be defined in the RDB\_node of the top element in the document access definition (DAD) file. Sub-element tables must match the tables defined in the top element. The table name in this RDB\_node is not in the top element.

**User Response:** Ensure that the table of the RDB node is defined in the top element of the DAD file.

---

**DXXQ017E The column in the result table <table\_name> is too small.**

**Explanation:** An XML document generated by the XML Extender is too large to fit into the column of the result table.

**User Response:** Drop the result table. Create another

result table with a bigger column. Rerun the stored procedure.

---

**DXXQ018E The ORDER BY clause is missing from the SQL statement.**

**Explanation:** The ORDER BY clause is missing from the SQL statement in a document access definition (DAD) file that maps SQL to XML.

**User Response:** Edit the DAD file. Add an ORDER BY clause that contains the entity-identifying columns.

---

**DXXQ019E The element objids has no column element in the DAD file.**

**Explanation:** The objids element does not have a column element in the document access definition (DAD) file that maps SQL to XML.

**User Response:** Edit the DAD file. Add the key columns as sub-elements of the element objids.

---

**DXXQ020I XML successfully generated.**

**Explanation:** The requested XML documents have been successfully generated from the database.

**User Response:** No action is required.

---

**DXXQ021E Table <table\_name> does not have column <column\_name>.**

**Explanation:** The table does not have the specified column in the database.

**User Response:** Specify another column name in DAD or add the specified column into the table database.

---

**DXXQ022E Column <column\_name> of <table\_name> should have type <type\_name>.**

**Explanation:** The type of the column is wrong.

**User Response:** Correct the type of the column in the document access definition (DAD).

---

**DXXQ023E Column <column\_name> of <table\_name> cannot be longer than <length>.**

**Explanation:** The length defined for the column in the DAD is too long.

**User Response:** Correct the column length in the document access definition (DAD).

---

**DXXQ024E Can not create table <table\_name>.**

**Explanation:** The specified table can not be created.

**User Response:** Ensure that the user ID creating the table has the necessary authority to create a table in the database.

---

**DXXQ025I XML decomposed successfully.**

**Explanation:** An XML document has been decomposed and stored in a collection successfully.

**User Response:** No action is required.

---

**DXXQ026E XML data <xml\_name> is too large to fit in column <column\_name>.**

**Explanation:** The specified piece of data from an XML document is too large to fit into the specified column.

**User Response:** Increase the length of the column using the ALTER TABLE statement or reduce the size of the data by editing the XML document.

---

**DXXQ028E Cannot find the collection <collection\_name> in the XML\_USAGE table.**

**Explanation:** A record for the collection cannot be found in the XML\_USAGE table.

**User Response:** Verify that you have enabled the collection.

---

**DXXQ029E Cannot find the DAD in XML\_USAGE table for the collection <collection\_name>.**

**Explanation:** A DAD record for the collection cannot be found in the XML\_USAGE table.

**User Response:** Ensure that you have enabled the collection correctly.

---

**DXXQ030E Wrong XML override syntax.**

**Explanation:** The XML\_override value is specified incorrectly in the stored procedure.

**User Response:** Ensure that the syntax of XML\_override is correct.

---

**DXXQ031E Table name cannot be longer than maximum length allowed by DB2.**

**Explanation:** The table name specified by the condition element in the DAD is too long.

**User Response:** Correct the length of the table name in document access definition (DAD).



---

**DXXQ032E Column name cannot be longer than maximum length allowed by DB2.**

**Explanation:** The column name specified by the condition element in the DAD is too long.

**User Response:** Correct the length of the column name in the document access definition (DAD).

---

**DXXQ033E Invalid identifier starting at <identifier>**

**Explanation:** The string is not a valid DB2 SQL identifier.

**User Response:** Correct the string in the DAD to conform to the rules for DB2 SQL identifiers.

---

**DXXQ034E Invalid condition element in top RDB\_node of DAD: <condition>**

**Explanation:** The condition element must be a valid WHERE clause consisting of join conditions connected by the conjunction AND.

**User Response:** See the XML Extender documentation for the correct syntax of the join condition in a DAD.

---

**DXXQ035E Invalid join condition in top RDB\_node of DAD: <condition>**

**Explanation:** Column names in the condition element of the top RDB\_node must be qualified with the table name if the DAD specifies multiple tables.

**User Response:** See the XML Extender documentation for the correct syntax of the join condition in a DAD.

---

**DXXQ036E A Schema name specified under a DAD condition tag is longer than allowed.**

**Explanation:** An error was detected while parsing text under a condition tag within the DAD. The condition text contains an id qualified by a schema name that is too long.

**User Response:** Correct the text of the condition tags in document access definition (DAD).

---

**DXXQ037E Cannot generate <element> with multiple occurrences.**

**Explanation:** The element node and its descendants have no mapping to database, but its multi\_occurrence equals YES.

**User Response:** Correct the DAD by either setting the multi\_occurrence to NO or create a RDB\_node in one of its descendants.

---

## Tracing

The XML Extender includes a trace facility that records XML Extender server activity. The trace file is not limited in size and can impact performance.

The trace facility records information in a server file about a variety of events, such as entry to or exit from an XML Extender component or the return of an error code by an XML Extender component. Because it records information for many events, the trace facility should be used only when necessary, for example, when you are investigating error conditions. In addition, you should limit the number of active applications when using the trace facility. Limiting the number of active applications can make isolating the cause of a problem easier.

Use the DXXTRC command at an OS/390 and z/OS server to start or stop tracing. You can issue the command from the USS command line, from TSO, or from JCL. You must have SYSADM, SYSCTRL, or SYSMINT authority to issue the command.

## Starting the trace

### Purpose

Records the XML Extender server activity. To start the trace, apply the `on` option to `DXXTRC`, along with the name of an existing directory to contain the trace file. When the trace is turned on, the file, `dxxDB2.trc`, is placed in the specified directory. The trace file is not limited in size.

### Format

#### Starting the trace from the USS command line:

```
▶▶ dxxtrc on trace_directory ◀◀
```

#### Starting the trace from TSO:

```
call 'dxx.load(dxxtrc)' 'on "trace_directory"' asis
```

#### Starting the trace from JCL:

```
//STEP EXEC PGM=DXXTRC,
// PARM='on "trace_directory"'
```

### Parameters

Table 61. Trace parameters

Parameter	Description
<code>trace_directory</code>	Name of an existing USS path and directory where the <code>dxxdb2.trc</code> is placed. Required, no default.

### Examples

The following examples show starting the trace, with file, `dxxdb2.trc`, in the `/u/user1/dxx/trace` directory.

#### From USS:

```
dxxtrc on /u/user1/trace
```

#### From TSO:

```
call 'dxx.load(dxxtrc)' 'on "/u/user1/dxx/trace"' asis
```

#### From JCL:

```
//STEP EXEC PGM=DXXTRC,
// PARM='on "/u/user1/dxx/trace"'
```

## Stopping the trace

### Purpose

Turns the trace off. Trace information is no longer logged. Because running the trace log file size is not limited and can impact performance, it is recommended to turn trace off in a production environment.

### Format

#### Stopping the trace from the USS command line:

```
▶▶—dxxtrc—off—————▶▶
```

#### Stopping the trace from TSO:

```
call 'dxx.load(dxxtrc)' 'off' asis
```

#### Stopping the trace from JCL:

```
//STEP EXEC PGM=DXXTRC,
// PARM='off'
```

### Examples

The following examples demonstrate stopping the trace.

#### From USS:

```
dxxtrc off
```

#### From TSO:

```
call 'dxx.load(dxxtrc)' 'off' asis
```

#### From JCL:

```
//STEP EXEC PGM=DXXTRC,
// PARM='off'
```

---

## Part 5. Appendixes



## Appendix A. DTD for the DAD file

This section describes the document type declarations (DTD) for the document access definition (DAD) file. The DAD file itself is a tree-structured XML document and requires a DTD. The DTD file name is `dxxdad.dtd`. Figure 12 shows the DTD for the DAD file. The elements of this file are described following the figure.

```
<?xml encoding="US-ASCII"?>

 <!ELEMENT DAD (dtdid?, validation, (Xcolumn | Xcollection))>
 <!ELEMENT dtdid (#PCDATA)>
 <!ELEMENT validation (#PCDATA)>
 <!ELEMENT Xcolumn (table*)>
 <!ELEMENT table (column*)>
 <!ATTLIST table name CDATA #REQUIRED
 key CDATA #IMPLIED
 orderBy CDATA #IMPLIED>

 <!ELEMENT column EMPTY>
 <!ATTLIST column
 name CDATA #REQUIRED
 type CDATA #IMPLIED
 path CDATA #IMPLIED
 multi_occurrence CDATA #IMPLIED>

 <!ELEMENT Xcollection (SQL_stmt?, objids?, prolog, doctype, root_node)>
 <!ELEMENT SQL_stmt (#PCDATA)>
 <!ELEMENT objids (column+)>
 <!ELEMENT prolog (#PCDATA)>
 <!ELEMENT doctype (#PCDATA | RDB_node)*>
 <!ELEMENT root_node (element_node)>
 <!ELEMENT element_node (RDB_node*,
 attribute_node*,
 text_node?,
 element_node*,
 namespace_node*,
 process_instruction_node*,
 comment_node*)>

 <!ATTLIST element_node
 name CDATA #REQUIRED
 ID CDATA #IMPLIED
 multi_occurrence CDATA "NO"
 BASE_URI CDATA #IMPLIED>

 <!ELEMENT attribute_node (column | RDB_node)>
 <!ATTLIST attribute_node
 name CDATA #REQUIRED>

 <!ELEMENT text_node (column | RDB_node)>
 <!ELEMENT RDB_node (table+, column?, condition?)>
 <!ELEMENT condition (#PCDATA)>
 <!ELEMENT comment_node (#PCDATA)>
 <!ELEMENT namespace_node (EMPTY)>
 <!ATTLIST namespace_node
 name CDATA #IMPLIED
 value CDATA #IMPLIED>

 <!ELEMENT process_instruction_node (#PCDATA)>
```

Figure 12. The DTD for the document access definition (DAD)

The DAD file has four major elements:

- DTDID
- validation
- Xcolumn
- Xcollection

Xcolumn and Xcollection have child element and attributes that aid in the mapping of XML data to relational tables in DB2. The following list describes the major elements and their child elements and attributes. Syntax examples are taken from Figure 12 on page 237.

#### **DTDID element**

Specifies the ID of the DTD stored in the DTD\_REF table. The DTDID points to the DTD that validates the XML documents or guides the mapping between XML collection tables and XML documents. DTDID is optional in DADs for XML columns and XML collections. For XML collection, this element is required for validating input and output XML documents. For XML columns, it is only needed to validate input XML documents. DTDID must be the same as the SYSTEM ID specified in the doctype of the XML documents.

**Syntax:** `<!ELEMENT dtdid (#PCDATA)>`

#### **validation element**

Indicates whether or not the XML document is to be validated with the DTD for the DAD. If YES is specified, then the DTDID must also be specified.

**Syntax:** `<!ELEMENT validation(#PCDATA)>`

#### **Xcolumn element**

Defines the indexing scheme for an XML column. It is composed of zero or more tables.

**Syntax:** `<!ELEMENT Xcolumn (table*)>`Xcolumn has one child element, table.

#### **table element**

Defines one or more relational tables created for indexing elements or attributes of documents stored in an XML column.

##### **Syntax:**

```
<!ELEMENT table (column+)>
<!ATTLIST table name CDATA #REQUIRED
key CDATA #IMPLIED
orderBy CDATA #IMPLIED>
```

The table element has one attribute:

##### **name attribute**

Specifies the name of the side table

The table element has one child element:

#### **key attribute**

The primary single key of the table

#### **orderBy attribute**

The names of the columns that determine the sequence order of multiple-occurring element text or attribute values when generating XML documents.



## column element

Specifies the column of the table that contains the value of a location path of the specified type.

### Syntax:

```
<!ATTLIST column
 name CDATA #REQUIRED
 type CDATA #IMPLIED
 path CDATA #IMPLIED
 multi_occurrence CDATA #IMPLIED>
```

The column element has the following attributes:

### name attribute

Specifies the name of the column. It is the alias name of the location path which identifies an element or attribute

### type attribute

Defines the data type of the column. It can be any SQL data type.

### path attribute

Shows the location path of an XML element or attribute and must be the simple location path as specified in Table 3.1.a (fix link) .

### multi\_occurrence attribute

Specifies whether this element or attribute can occur more than once in an XML document. Values can be YES or NO.

## Xcollection

Defines the mapping between XML documents and an XML collection of relational tables.

**Syntax:** <!ELEMENT Xcollection(SQL\_stmt\*, prolog, doctype, root\_node)>Xcollection has the following child elements:

### SQL\_stmt

Specifies the SQL statement that the XML Extender uses to define the collection. Specifically, the statement selects XML data from the XML collection tables, and uses the data to generate the XML documents in the collection. The value of this element must be a valid SQL statement. It is only used for composition, and only a single SQL\_stmt is allowed. For decomposition, more than one value for SQL\_stmt can be specified to perform the necessary table creation and insertion.

**Syntax:** <!ELEMENT SQL\_stmt #PCDATA >

### prolog

The text for the XML prolog. The same prolog is supplied to all documents in the entire collection. The value of prolog is fixed.

**Syntax:** <!ELEMENT prolog #PCDATA>

### doctype

Defines the text for the XML document type definition.

**Syntax:** <!ELEMENT doctype #PCDATA | RDB\_node>doctype can be specified in one of the following ways:

- Define an explicit value. This value is supplied to all documents in the entire collection.
- When using decomposition, specify the child element, RDB\_node, that can be mapped to and stored as column data of a table.

doctype has one child element:

**RDB\_node**

Defines the DB2 table where the content of an XML element or value of an XML attribute is to be stored or from where it will be retrieved. The RDB\_node has the following child elements:

**table** Specifies the table in which the element or attribute content is stored.

**column**

Specifies the column in which the element or attribute content is stored.

**condition**

Specifies a condition for the column. Optional.

**root\_node**

Defines the virtual root node. root\_node must have one required child element, element\_node, which can be used only once. The element\_node under the root\_node is actually the root\_node of the XML document.

**Syntax:** <!ELEMENT root\_node(element\_node)>

**element\_node**

Represents an XML element. It must be defined in the DTD specified for the collection. For the RDB\_node mapping, the root element\_node must have a RDB\_node to specify all tables containing XML data for itself and all of its child nodes. It can have zero or more attribute\_nodes and child element\_nodes, as well as zero or one text\_node. For elements other than the root element no RDB\_node is needed.

**Syntax:**

An element\_node is defined by the following child elements:

**RDB\_node**

(Optional) Specifies tables, column, and conditions for XML data. The RDB\_node for an element only needs to be defined for the RDB\_node mapping. In this case, one or more tables must be specified. The column is not needed since the element content is specified by its text\_node. The condition is optional, depending on the DTD and query condition.

**child nodes**

(Optional) An element\_node can also have the following child nodes:

**element\_node**

Represents child elements of the current XML element

**attribute\_node**

Represents attributes of the current XML element

**text\_node**

Represents the CDATA text of the current XML element

**attribute\_node**

Represents an XML attribute. It is the node defining the mapping between an XML attribute and the column data in a relational table.

**Syntax:**

The `attribute_node` must have definitions for a `name` attribute, and either a `column` or a `RDB_node` child element. `attribute_node` has the following attribute:

**name** The name of the attribute.

`attribute_node` has the following child elements:

**Column**

Used for the SQL mapping. The column must be specified in the `SELECT` clause of `SQL_stmt`.

**RDB\_node**

Used for the `RDB_node` mapping. The node defines the mapping between this attribute and the column data in the relational table. The table and column must be specified. The condition is optional.

**text\_node**

Represents the text content of an XML element. It is the node defining the mapping between an XML element content and the column data in a relational table.

**Syntax:** It must be defined by a `column` or an `RDB_node` child element:

**Column**

Needed for the SQL mapping. In this case, the column must be in the `SELECT` clause of `SQL_stmt`.

**RDB\_node**

Needed for the `RDB_node` mapping. The node defines the mapping between this text content and the column data in the relational table. The table and column must be specified. The condition is optional.



---

## Appendix B. Samples

This appendix shows the sample objects that are used with examples in this book.

- “XML DTD”
- “XML document: getstart.xml” on page 244
- “Document access definition files” on page 244
  - “DAD file: XML column” on page 244
  - “DAD file: XML collection - SQL mapping” on page 245
  - “DAD file: XML - RDB\_node mapping” on page 248

---

### XML DTD

The following DTD is used for the getstart.xml document that is referenced throughout this book and shown in Figure 14 on page 244.

```
<!xml encoding="US-ASCII"?>

<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
<!ELEMENT Customer (Name, Email)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key,Quantity,ExtendedPrice,Tax, Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
```

*Figure 13. Sample XML DTD: getstart.dtd*

---

## XML document: getstart.xml

The following XML document, `getstart.xml`, is the sample XML document that is used in examples throughout this book. It contains XML tags to form a purchase order.

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd">
<Order key="1">
 <Customer>
 <Name>American Motors</Name>
 <Email>parts@am.com</Email>
 </Customer>
 <Part color="black ">
 <key>68</key>
 <Quantity>36</Quantity>
 <ExtendedPrice>34850.16</ExtendedPrice>
 <Tax>6.000000e-02</Tax>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>BOAT </ShipMode>
 </Shipment>
 <Shipment>
 <ShipDate>1998-08-19</ShipDate>
 <ShipMode>AIR </ShipMode>
 </Shipment>
 </Part>
 <Part color="red ">
 <key>128</key>
 <Quantity>28</Quantity>
 <ExtendedPrice>38000.00</ExtendedPrice>
 <Tax>7.000000e-02</Tax>
 <Shipment>
 <ShipDate>1998-12-30</ShipDate>
 <ShipMode>TRUCK </ShipMode>
 </Shipment>
 </Part>
</Order>
```

*Figure 14. Sample XML document: getstart.xml*

---

## Document access definition files

The following sections contain document access definition (DAD) files that map XML data to DB2 relational tables, using either XML column or XML collection access modes.

- “DAD file: XML column”
- “DAD file: XML collection - SQL mapping” on page 245 shows a DAD file for an XML collection using SQL mapping.
- “DAD file: XML - RDB\_node mapping” on page 248 show a DAD for an XML collection that uses RDB\_node mapping.

### DAD file: XML column

This DAD file contains the mapping for an XML column, defining the table, side tables, and columns that are to contain the XML data.

```

<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxx_install/dad.dtd">
<DAD>
 <dttdid>dxx_install/samples/dtd/getstart.dtd</dttdid>
 <validation>YES</validation>

 <Xcolumn>
 <table name="order_side_tab">
 <column name="order_key"
 type="integer"
 path="/Order/@key"
 multi_occurrence="NO"/>
 <column name="customer"
 type="varchar(50)"
 path="/Order/Customer/Name"
 multi_occurrence="NO"/>
 </table>
 <table name="part_side_tab">
 <column name="price"
 type="decimal(10,2)"
 path="/Order/Part/ExtendedPrice"
 multi_occurrence="YES"/>
 </table>
 <table name="ship_side_tab">
 <column name="date"
 type="DATE"
 path="/Order/Part/Shipment/ShipDate"
 multi_occurrence="YES"/>
 </table>
 </Xcolumn>
</DAD>

```

Figure 15. Sample DAD file for an XML column: *getstart\_xcolumn.dad*

## DAD file: XML collection - SQL mapping

This DAD file contains an SQL statement that specifies the DB2 tables, columns, and conditions that are to contain the XML data.

```

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>SELECT o.order_key, customer_name, customer_email, p.part_key, color, quantity,
price, tax, ship_id, date, mode from order_tab o, part_tab p,
table(select db2xml.generate_unique(),
as ship_id, date, mode, part_key from ship_tab)
s
WHERE o.order_key = 1 and
p.price > 20000 and
p.order_key = o.order_key and
s.part_key = p.part_key
ORDER BY order_key, part_key, ship_id</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>

```

*Figure 16. Sample DAD file for an XML collection using SQL mapping: order\_sql.dad (Part 1 of 2)*



```

<root_node>
<element_node name="Order">
 <attribute_node name="key">
 <column name="order_key"/>
 </attribute_node>
 <element_node name="Customer">
 <element_node name="Name">
 <text_node><column name="customer_name"/></text_node>
 </element_node>
 <element_node name="Email">
 <text_node><column name="customer_email"/></text_node>
 </element_node>
 </element_node>
 <element_node name="Part">
 <attribute_node name="color">
 <column name="color"/>
 </attribute_node>
 <element_node name="key">
 <text_node><column name="part_key"/></text_node>
 </element_node>
 <element_node name="Quantity">
 <text_node><column name="quantity"/></text_node>
 </element_node>
 <element_node name="ExtendedPrice">
 <text_node><column name="price"/></text_node>
 </element_node>
 <element_node name="Tax">
 <text_node><column name="tax"/></text_node>
 </element_node>
 <element_node name="Shipment" multi_occurrence="YES">
 <element_node name="ShipDate">
 <text_node><column name="date"/></text_node>
 </element_node>
 <element_node name="ShipMode">
 <text_node><column name="mode"/></text_node>
 </element_node>
 </element_node>
 </element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>

```

Figure 16. Sample DAD file for an XML collection using SQL mapping: order\_sql.dad (Part 2 of 2)

## DAD file: XML - RDB\_node mapping

This DAD file uses <RDB\_node> elements to define the DB2 tables, columns, and conditions that are to contain XML data.

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "dxx_install/dtd/dad.dtd">
<DAD>
 <dtid>dxx_install/samples/dtd/getstart.dtd</dtid>
 <validation>YES</validation>
<Xcollection>
 <prolog>?xml version="1.0"?</prolog>
 <doctype>!DOCTYPE Order SYSTEM "dxx_install/samples/dtd/getstart.dtd"</doctype>
 <root_node>
 <element_node name="Order">
 <RDB_node>
 <table name="order_tab"/>
 <table name="part_tab"/>
 <table name="ship_tab"/>
 <condition>
 order_tab.order_key = part_tab.order_key AND
 part_tab.part_key = ship_tab.part_key
 </condition>
 </RDB_node>
 <attribute_node name="key">
 <RDB_node>
 <table name="order_tab"/>
 <column name="order_key"/>
 </RDB_node>
 </attribute_node>
 <element_node name="Customer">
 <text_node>
 <RDB_node>
 <table name="order_tab"/>
 <column name="customer"/>
 </RDB_node>
 </text_node>
 </element_node>
 <element_node name="Part">
 <RDB_node>
 <table name="part_tab"/>
 <table name="ship_tab"/>
 <condition>
 part_tab.part_key = ship_tab.part_key
 </condition>
 </RDB_node>
 <attribute_node name="key">
 <RDB_node>
 <table name="part_tab"/>
 <column name="part_key"/>
 </RDB_node>
 </attribute_node>
 </element_node>
 </element_node>
 </root_node>
</Xcollection>
```

Figure 17. Sample DAD file for an XML collection using RDB\_node mapping: order\_rdb.dad (Part 1 of 3)

```

<element_node name="Quantity">
 <text_node>
 <RDB_node>
 <table name="part_tab"/>
 <column name="quantity"/>
 </RDB_node>
 </text_node>
</element_node>
<element_node name="ExtendedPrice">
 <text_node>
 <RDB_node>
 <table name="part_tab"/>
 <column name="price"/>
 <condition>
 price > 2500.00
 </condition>
 </RDB_node>
 </text_node>
</element_node>
<element_node name="Tax">
 <text_node>
 <RDB_node>
 <table name="part_tab"/>
 <column name="tax"/>
 </RDB_node>
 </text_node>
</element_node>

```

Figure 17. Sample DAD file for an XML collection using RDB\_node mapping: order\_rdb.dad (Part 2 of 3)

```

<element_node name="shipment">
 <RDB_node>
 <table name="ship_tab"/>
 <condition>
 part_key = part_tab.part_key
 </condition>
 </RDB_node>
 <element_node name="ShipDate">
 <text_node>
 <RDB_node>
 <table name="ship_tab"/>
 <column name="date"/>
 <condition>
 date > "1966-01-01"
 </condition>
 </RDB_node>
 </text_node>
 </element_node>
 <element_node name="ShipMode">
 <text_node>
 <RDB_node>
 <table name="ship_tab"/>
 <column name="mode"/>
 </RDB_node>
 </text_node>
 </element_node>
 <element_node name="Comment">
 <text_node>
 <RDB_node>
 <table name="ship_tab"/>
 <column name="comment"/>
 </RDB_node>
 </text_node>
 </element_node>
</element_node> <! -- end of element Shipment>
</element_node> <! -- end of element Part ---->
</element_node> <! -- end of element Order ---->
</root_node>
</Xcollection>
</DAD>

```

Figure 17. Sample DAD file for an XML collection using RDB\_node mapping: order\_rdb.dad (Part 3 of 3)

---

## Appendix C. Code page considerations

XML documents and other related files must be encoded properly for each client or server that accesses the files. The XML Extender makes some assumptions when processing a file, you need to understand how it handles code page conversions. The primary considerations are:

- Ensuring that the actual code page of the client retrieving an XML document from DB2 matches the encoding of the document.
- Ensuring that, when the document is processed by an XML parser, the encoding declaration of the XML document is also consistent with the document's actual encoding.
- Determining how parsers and other tools handle line endings and determining how to present files so that they are processed.

The following sections describe the issues for these considerations, how you can prepare for possible problems, and how the XML Extender and DB2 support code pages when documents are passed from client to server, and to the database.

---

### Terminology

The following terms are used in the section:

**document encoding**

The code page of an XML document.

**document encoding declaration**

The name of the code page specified in the XML declaration. For example, the following encoding declaration specifies `ibm-1047`:

```
<?xml version="1.0" encoding="ibm-1047"?>
```

**consistent document**

A document in which the code page matches the encoding declaration.

**inconsistent document**

A document in which the code page does not match the encoding declaration.

**client code page**

The application code page. The default client code page is the value of the operating system locale on a Windows or UNIX client.

**server code page, or server operating system locale code page**

The operating system locale of the HFS file system on USS, that is in the same OS/390 system as the XML-enabled database. The XML Extender uses the `n1_langinfo` environment option to determine the value of the server code page.

**database code page**

The encoding of the stored data, determined at database create time. This value defaults to the server operating system locale.

---

### DB2 and XML Extender code page assumptions

When DB2 sends or receives an XML document, it does not check the encoding declaration. Rather, it checks the code page for the client to see if it matches the database code page. If they are different, DB2 converts the data in the XML document to match the code page of:

- The database, when importing the document, or a document fragment, into a database table.
- The database, when decomposing a document into one or more database tables.
- The client, when exporting the document from the database and presenting the document to the client.
- The server, when processing a file with a UDF that returns data in a file on the server's file system.

## **Assumptions for importing an XML document**

When an XML document is imported into the database, it is generally imported as an XML document to be stored in an XML column, or for decomposition for an XML collection, where the element and attribute contents will be saved as DB2 data. When a document is imported, DB2 converts the document encoding to that of the database. DB2 assumes that the document is in the code page specified in the "Source code page" column in the table below. Table 62 on page 253 summarizes the conversions that DB2 makes when importing an XML document.

Table 62. Using UDFs and stored procedures when the XML file is imported into the database

If you are...	This is the source code page for conversion	This is the target code page for conversion	Comments
Inserting DTD file into DTD_REF table	Client code page	Database code page	
Enabling a column or enabling a collection with stored procedures, or using administration commands that import DAD files	Client code page — the code page used to bind DXXADMIN during installation, when enabling in USS.	Database code page	
Using user-defined functions: <ul style="list-style-type: none"> <li>XMLVarcharFromFile()</li> <li>XMLCLOBFromFile()</li> <li>Content(): retrieve from XMLFILE to a CLOB</li> </ul>	Server code page	Database code page	The database code page is converted to the client code page when the data is presented to the client
Using stored procedures for decomposition	Client code page	Database code page	<ul style="list-style-type: none"> <li>Document to be decomposed is assumed to be in client code page. Data from decomposition is stored in tables in database code page</li> <li>Use the CCSID option for DAD and XML files described in “Consistent encodings in USS” on page 257, when the calling application runs in USS.</li> </ul>

## Assumptions for exporting an XML document

When an XML document is exported from the database, it is exported based on a client request to present one of the following objects:

- An XML document from an XML column
- The query results of XML documents in an XML column
- A composed XML document from an XML collection

When a document is exported, DB2 converts the document encoding to that of the client or server, depending on where the request originated and where the data is to be presented. Table 63 on page 254 summarizes the conversions that DB2 makes when exporting an XML document.

Table 63. Using UDFs and stored procedures when the XML file is exported from the database

If you are...	DB2 converts the ...	Comments
Using user-defined functions: <ul style="list-style-type: none"> <li>XMLFileFromVarchar()</li> <li>XMLFileFromCLOB()</li> <li>Content(): retrieve from XMLVARCHAR to an external server file</li> </ul>	Database code page to server code page	
Composing XML documents with a stored procedure that are stored in a result table, which can be queried and exported.	Database code page to client code page when result set is presented to client	<ul style="list-style-type: none"> <li>When composing documents, the XML Extender copies the encoding declaration specified by the tag in the DAD, to the newly created document. It should match the client code page when presented.</li> <li>Use the CCSID option for DAD files described in "Consistent encodings in USS" on page 257, when the calling application runs in USS.</li> </ul>

## Encoding declaration considerations

The *encoding declaration* specifies the code page of the XML document's encoding and appears on the XML declaration statement. When using the XML Extender, it is important to ensure that the encoding of the document matches the code page of the client or the server, depending on where the file is located.

## Legal encoding declarations

You can use any encoding declaration in XML documents, within some guidelines. In this section, these guidelines are defined, along with the supported encoding declarations.

If you use the encodings listed in Table 64, your application can be ported between IBM operating systems. If you use other encodings, your data is less likely to be portable.

For all operating systems, the following encoding declarations are supported. The following list describes the meaning of each column:

- **Encoding** specifies the encoding string to be used in the XML declaration.
- **OS** shows the operating system on which DB2 supports the given code page.
- **Code page** shows the IBM-defined code page associated with the given encoding

Table 64. Encoding declarations supported by XML Extender

Category	Encoding	OS	Code page
Unicode	UTF-8	AIX, SUN, Linux	1208
	UTF-16	AIX, SUN, Linux	1200



Table 64. Encoding declarations supported by XML Extender (continued)

Category	Encoding	OS	Code page
EBCDIC	ibm-037	OS/390 and z/OS	37
	ibm-273	OS/390 and z/OS	273
	ibm-277	OS/390 and z/OS	277
	ibm-278	OS/390 and z/OS	278
	ibm-280	OS/390 and z/OS	280
	ibm-284	OS/390 and z/OS	284
	ibm-297	OS/390 and z/OS	297
	ibm-500	OS/390 and z/OS	500
	ibm-1047	OS/390 and z/OS	1047
	ibm-1140	OS/390 and z/OS	1140
ASCII	iso-8859-1	AIX, Linux, Sun	819
	ibm-1252	Windows NT	1252
	iso-8859-2	AIX, Linux, Sun	912
	iso-8859-5	AIX, Linux	915
	iso-8859-6	AIX	1089
	iso-8859-7	AIX, Linux	813
	iso-8859-8	AIX, Linux	916
	iso-8859-9	AIX, Linux	920

The encoding string must be compatible with the code page of the document's destination. If a document is being returned from a server to a client, then its encoding string must be compatible with the client's code page. See "Consistent encodings and encoding declarations" for the consequences of incompatible encodings. See the following Web address for a list of code pages supported by the XML parser used by the XML Extender:

<http://www.ibm.com/software/data/db2/extenders/xml/ext/moreinfo/encoding.html>

## Consistent encodings and encoding declarations

When an XML document is processed or exchanged with another system, it is important that the encoding declaration corresponds to the actual encoding of the document. Ensuring that the encoding of a document is consistent with the client is important because XML tools, like parsers, generate an error for an entity that includes an encoding declaration other than that named in the declaration.

Figure 18 on page 256 shows that clients have consistent code pages with the document encoding and declared encoding.

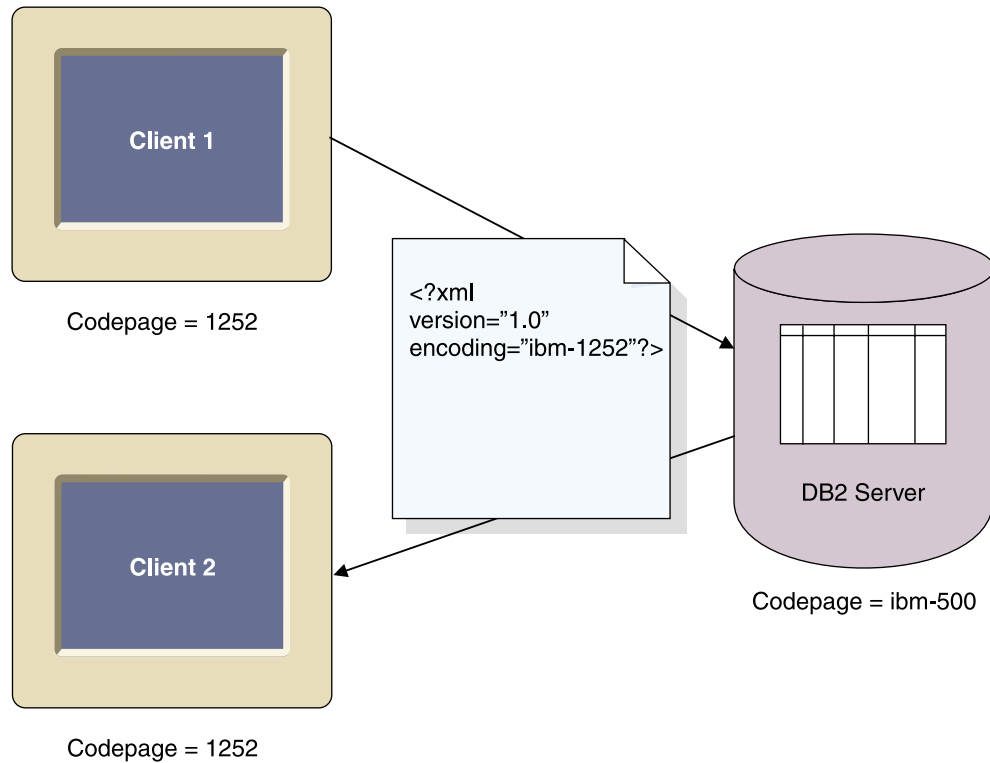


Figure 18. Clients have matching code pages

The consequences of having different code pages are the following possible situations:

- A conversion in which data is lost might occur, particularly if the source code page is Unicode and the target code page is not Unicode. Unicode contains the full set of character conversions. If a file is converted from UTF-8 to a code page that does not support all the characters used in the document, then data might be lost during the conversion.
- The declared encoding of the XML document might no longer be consistent with the actual document encoding, if the document is retrieved by a client with a different code page than the declared encoding of the document.

Figure 19 on page 257 shows an environment in which the code pages of the clients are inconsistent.

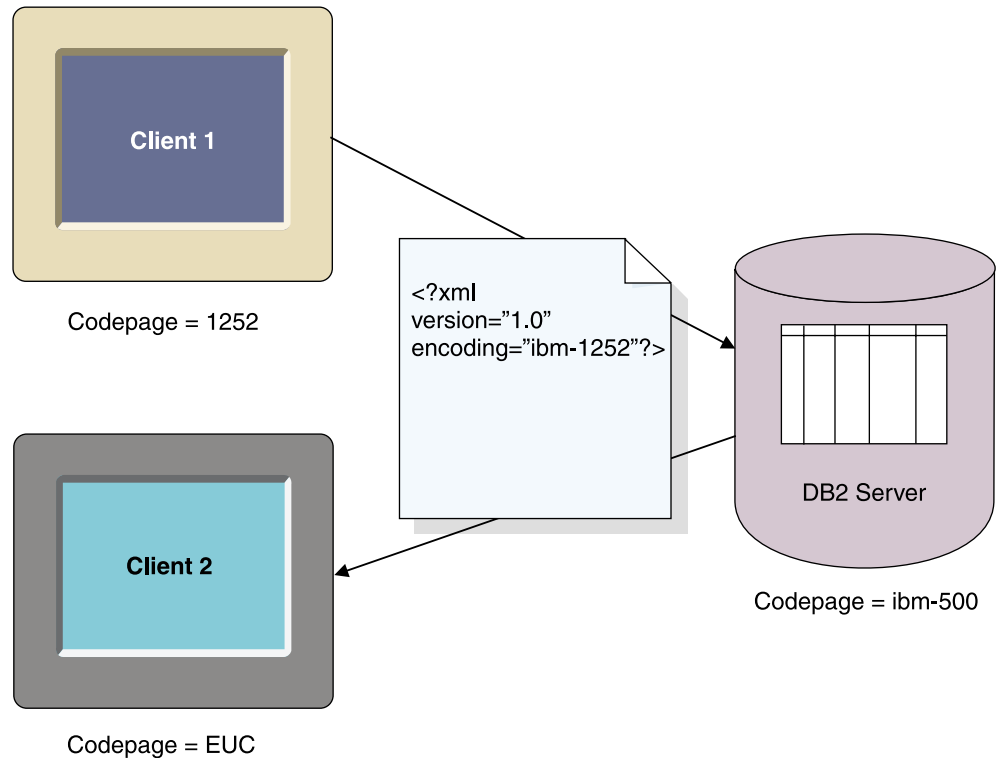


Figure 19. Clients have mismatching code pages

Client 2 receives the document in EUC, but the document will have an encoding declaration of ibm-1252.

## Consistent encodings in USS

When running applications in USS, there are two considerations:

- When you enable an XML column or collection, and specify a DAD file that is stored in HFS, bind the DXXADM package with the ENCODING option and specify the actual code page of the DAD file. The bind step for DXXADM is included in the DXXGPREP JCL job. See “Initializing the XML Extender environment using DXXGPREP” on page 39 to learn what this job does and how to run it. For example, if the DAD file has a code page of 1047, specify this value on the ENCODING option:

```
BIND PACKAGE (DB2XML) MEMBER(DXXADM) ENCODING(1047);
```

- When DAD files stored in HFS are used in a calling application, declare a host variable in the calling program with the coded character set identifier (CCSID) of the actual code page of the DAD file. If the file is created in HFS, the code page is that of HFS. If the file has been imported into HFS in binary mode, the code page might be different. This declaration ensures that DB2 converts the DAD code page, to the database code page. For example, if a DAD file is in 1047, use the following variable declaration:

```
EXEC SQL DECLARE :dadobj VARIABLE CCSID 1047;
```

- When XML files stored in HFS are used in a calling application for decomposition, declare a host variable in the calling program with the CCSID of the actual code page of the XML file. If the file is created in HFS, the code page is that of HFS. If the file has been imported into HFS in binary mode, the code

page might be different. This declaration ensures that DB2 converts the XML file code page, to the database code page. For example, if a DAD file is in 1047, use the following variable declaration:

```
EXEC SQL DECLARE :xmlobj VARIABLE CCSID 1047;
```

## Declaring an encoding

The default value of the encoding declaration is UTF-8, and the absence of an encoding declaration means the document is in UTF-8.

### To declare an encoding value:

In the XML document declaration specify the encoding declaration with the name of the code page of the client. For example:

```
<?xml version="1.0" encoding="ibm-1047" ?>
```

---

## Conversion scenarios

The XML Extender processes XML documents when:

- Storing and retrieving XML column data, using the XML column storage and access method
- Composing and decomposing XML documents

Documents undergo code page conversion when passed from a client or server, to a database. Inconsistencies or damage of XML documents is most likely to occur during conversions from code pages of the client, server, and database. When choosing the encoding declaration of the document, as well as planning what clients and servers can import or export documents from the database, consider the conversions described in the above tables, and the scenarios described below.

The following scenarios describe common conversion scenarios that can occur:

**Scenario 1:** This scenario is a configuration with consistent encodings, no DB2 conversion, and a document imported from the server. The document encoding declaration is ibm-1047, the server is ibm-1047, and the database is ibm-1047.

1. The document is imported into DB2 using the XMLClobFromFile UDF.
2. The document is extracted to the server.
3. DB2 does not need to convert the document because the server code page and database code page are identical. The encoding and declaration are consistent.

**Scenario 2:** This scenario is a configuration with consistent encodings, DB2 conversion, and a document imported from server and exported to client. The document encoding and declaration is ibm-1047 the client and server code pages are ibm-1047, and the database code pages are ibm-500.

1. The document is imported into DB2 using XMLClobFromFile UDF from the server.
2. DB2 converts the document from ibm-1047 and stores it in ibm-500. The encoding declaration and encoding are inconsistent in the database.
3. A client using ibm-1047 requests the document for presentation at the Web browser.
4. DB2 converts the document to ibm-1047, the client's code page. The document encoding and the declaration are now consistent at the client.

**Scenario 3:** This scenario is a configuration with inconsistent encodings, DB2 conversion, a document imported from the server and exported to a client. The document encoding declaration is ibm-1047 for the incoming document. The server code page is ibm-1047 and the client and database are ibm-500.

1. The document is imported into the database using a storage UDF.
2. DB2 converts the document to ibm-500 from ibm-1047. The encoding and declaration are inconsistent.
3. A client with a ibm-500 code page requests the document for presentation at a Web browser.
4. DB2 does not convert because the client and the database code pages are the same.
5. The document encoding and declaration are inconsistent because the declaration is ibm-1047 and the encoding is ibm-500. The document cannot be processed by an XML parser or other XML processing tools.

---

## Preventing inconsistent XML documents

The above sections have discussed how an XML document can have an inconsistent encoding, that is, the encoding declaration conflicts with the document's encoding. Inconsistent encodings can cause the lost of data and or unusable XML documents.

Use one of the following recommendations for ensuring that the XML document encoding is consistent with the client code page, before handing the document to an XML processor, such as a parser:

- When exporting a document from the database using the XML Extender UDFs, try one of the following techniques (assuming the XML Extender has exported the file, in the server code page, to the file system on the server):
  - Convert the document to the declared encoding code page
  - Override the declared encoding, if the tool has an override facility
  - Manually change the encoding declaration of the exported document to the document's actual encoding (that is, the server code page)
- When exporting a document from the database using the XML Extender stored procedures, try one of the following techniques (assuming the client is querying the result table, in which the composed document is stored):
  - Convert the document to the declared encoding code page
  - Override the declared encoding, if the tool has an override facility
  - Before running the stored procedure, have the client set the CCSID variable to force the client code page to a code page that is compatible with the encoding declaration of the XML document.
  - Manually change the encoding declaration of the exported document to the document's actual encoding (that is, the client code page)
- **Limitation when using Unicode and a Windows NT client:** On Windows NT, the operating system locale cannot be set to UTF-8. Use the following guidelines when importing or exporting documents:
  - When importing files and DTDs encoded in UTF-8, set the client code page to UTF-8, using:

```
db2set DB2CODEPAGE=1208
```

Use this technique when:

- Inserting a DTD into the DB2XML.DTD\_REF table

- Enabling a column or collection
- Decomposing stored procedures
- When using the Content() or XMLFromFile UDFs to import XML documents, documents must be encoded in the code page of the server's operating system locale, which cannot be UTF-8.
- When exporting an XML file from the database, set the client code page with the following command to have DB2 encode the resulting data in UTF-8:  
db2set DB2CODEPAGE=1208

Use this technique when:

- Querying the result table after composition
- Extracting data from an XML column using the extract UDFs
- When using the Content() or XMLxxxfromFile UDFs to export XML documents to files on the server file system, resulting documents are encoded in the code page of the server's operating system locale, which cannot be UTF-8.

---

## Line ending considerations

When storing XML and DAD files, consider that the file can be treated inconsistently by editors and parsers because DB2 for OS/390 and z/OS stores files with the NL as the line ending. Many tools do not recognize the NL line endings.

DB2 uses the NL line ending because:

- DB2 uses Character Data Representation Architecture (CDRA) as the basis for data conversions across systems through Distributed Relational Database Architecture (DRDA). For more information on CDRA, see *IBM Character Data Representation Architecture, Reference and Registry*.
- DB2 files are routinely accessed across operating systems

For example, SQL procedures source code, held in the DB2 catalog, is stored with [LF] as its line ending. In general files or documents can contain the following line endings: [CR], [CRLF], or [LF], as well as [NL], as in the following example:

```
This is line 1 of a UNIX document [LF]
This is line 1 of an Apple Macintosh document [CR]
This is line 1 of a DOS/Windows document [CR][LF]
This is line 1 of an OS/390 and z/OS USS file [NL].....
```

Most workstation tools recognize [LF], [CR], and [CR][LF], but not [NL], which is used by OS/390 and z/OS. Different line endings can appear together in one document.

Traditional MVS files rely on either:

- Fixed record length
- Variable record length with a length at the beginning of the record rather than line ending control characters.

Programs written with the C-runtime and USS or Open Edition (OE) rely on [NL] as the line ending.

## Processing XML documents with the linebrk utility

Use the linebrk utility to convert [NL] line endings [LF] line endings, or the reverse.

Download the utility from the DB2 XML Extender Web site:

<http://www.ibm.com/software/data/db2/extenders/xmlext/download.html>

Syntax:

►► `linebrk` `input_file_name` `output_file_name` `-nl` `-f` `-v` ►

Where:

*input\_file\_name*

Specifies the name of the file to be processed.

*output\_file\_name*

Specifies the name of the resulting file.

**-nl** Specifies that the file is to be converted from LF to NL.

**-f** Specifies that the file is to be converted from NL to LF.

**-v** Specifies the verbose option, which provides information as the command processes the file.





## Appendix D. The XML Extender limits

This appendix describes the limits for:

- XML Extender objects - Table 65
- values returned by user-defined functions - Table 66
- stored procedures parameters - Table 67
- administration support table columns - Table 68 on page 264

Table 65 describes the limits for XML Extender objects.

*Table 65. Limits for XML Extender objects*

<b>Object</b>	<b>Limit</b>
Maximum number of rows in a table in a decomposition XML collection	1024 rows from each decomposed XML document
Maximum bytes in XMLFile path name specified as a parameter value	512 bytes

Table 66 describes the limits values returned by XML Extender user-defined functions.

*Table 66. Limits for user-defined function value*

<b>User-defined functions returned values</b>	<b>Limit</b>
Maximum bytes returned by an extractCHAR UDF	254 bytes
Maximum bytes returned by an extractCLOB UDF	2 gigabytes
Maximum bytes returned by an extractVARCHAR UDF	4 kilobytes

Table 67 describes the limits for parameters of XML Extender stored procedures.

*Table 67. Limits for stored procedure parameters*

<b>Stored procedure parameters</b>	<b>Limit</b>
Maximum size of an XML document CLOB <sup>1</sup>	1 megabytes
Maximum size of a Document Access Definition (DAD) CLOB <sup>1</sup>	100 kilobytes
Maximum size of <i>collectionName</i>	30 bytes
Maximum size of <i>colName</i>	30 bytes
Maximum size of <i>dbName</i>	8 bytes
Maximum size of <i>defaultView</i>	128 bytes
Maximum size of <i>rootID</i>	128 bytes
Maximum size of <i>resultTabName</i>	18 bytes
Maximum size of <i>tablespace</i>	8 bytes
Maximum size of <i>tbName</i>	18 bytes

Table 67. Limits for stored procedure parameters (continued)

Stored procedure parameters	Limit
<b>Notes:</b>	
1. This size can be changed. See “Increasing the CLOB limit” on page 190 to learn how.	
2. If the value of the <i>tbName</i> parameter is qualified by a schema name, the entire name (including the separator character) must be no longer than 128 bytes.	

Table 68 describes the limits for the DB2XML.DTD\_REF table.

Table 68. XML Extender limits

DB2XML.DTD_REF table columns	Limit
Size of AUTHOR column	128 bytes
Size of CREATOR column	128 bytes
Size of UPDATOR column	128 bytes
Size of DTDID column	128 bytes
Size of CLOB column	100 kilobytes

Names can undergo expansion when DB2 converts them from the client code page to the database code page. A name might fit within the size limit at the client, but exceed the limit when the stored procedure gets the converted name. See the “National Language Support Application Development” section in the “Programming in Complex Environments” chapter of the *DB2 Universal Database for OS/390 and z/OS Application Programming and SQL Guide, Version 7* for more information.

---

## Appendix E. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited  
Office of the Lab Director  
1150 Eglinton Ave. East  
North York, Ontario  
M3C 1H7  
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

---

## Trademarks

The following terms, which may be denoted by an asterisk(\*), are trademarks of International Business Machines Corporation in the United States, other countries, or both.

ACF/VTAM	IBM
AISPO	IMS
AIX	IMS/ESA
AIX/6000	LAN DistanceMVS
AIXwindows	MVS/ESA
AnyNet	MVS/XA
APPN	Net.Data
AS/400	OS/2
BookManager	OS/390
CICS	OS/400
C Set++	PowerPC
C/370	QBIC
DATABASE 2	QMF
DataHub	RACF
DataJoiner	RISC System/6000
DataPropagator	RS/6000
DataRefresher	S/370
DB2	SP
DB2 Connect	SQL/DS
DB2 Extenders	SQL/400
DB2 OLAP Server	System/370
DB2 Universal Database	System/390
Distributed Relational Database Architecture	SystemView
DRDA	VisualAge
eNetwork	VM/ESA
Extended Services	VSE/ESA
FFST	VTAM
First Failure Support Technology	WebExplorer
	WIN-OS/2

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java or all Java-based trademarks and logos, and Solaris are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, other countries, or both.

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

Other company, product, or service names, which may be denoted by a double asterisk(\*\*) may be trademarks or service marks of others.



---

# Glossary

**absolute location path.** The full path name of an object. The absolute path name begins at the highest level, or "root" element, which is identified by the forward slash (/) or back slash (\) character.

**access and storage method.** Associates XML documents to a DB2 database through two major access and storage methods: XML columns and XML collections. See also *XML column* and *XML collection*.

**administrative support tables.** A tables used by a DB2 extender to process user requests on XML objects. Some administrative support tables identify user tables and columns that are enabled for an extender. Other administrative support tables contain attribute information about objects in enabled columns. Synonymous with metadata table.

**API.** See *application programming interface*.

**application programming interface (API).**

(1) A functional interface supplied by the operating system or by a separately orderable licensed program. An API allows an application program that is written in a high-level language to use specific data or functions of the operating system or the licensed programs.

(2) In DB2, a function within the interface, for example, the get error message API.

(3) In DB2, a function within the interface. For example, the get error message API.

**attribute.** See *XML attribute*.

**attribute\_node.** A representation of an attribute of an element.

**browser.** See *Web browser*.

**B-tree indexing.** The native index scheme provided by the DB2 engine. It builds index entries in the B-tree structure. Supports DB2 base data types.

**cast function.** A function that is used to convert instances of a (source) data type into instances of a different (target) data type. In general, a cast function has the name of the target data type. It has one single argument whose type is the source data type; its return type is the target data type.

**character large object (CLOB).** A character string of single-byte characters, where the string can be up to 2 GB. CLOBs have an associated code page. Text objects that contain single-byte characters are stored in a DB2 database as CLOBs.

**CLOB.** Character large object.

**column data.** The data stored inside of a DB2 column. The type of data can be any data type supported by DB2.

**compose.** To generate XML documents from relational data in an XML collection.

**condition.** A specification of either the criteria for selecting XML data or the way to join the XML collection tables.

**DAD.** See *Document access definition*.

**data interchange.** The sharing of data between applications. XML supports data interchange without needing to go through the process of first transforming data from a proprietary format.

**data source.** A local or remote relational or nonrelational data manager that is capable of supporting data access via an ODBC driver that supports the ODBC APIs.

**data type.** An attribute of columns and literals.

**datalink.** A DB2 data type that enables logical references from the database to a file that is stored outside the database.

**DBCLOB.** Double-byte character large object.

**decompose.** Separates XML documents into a collection of relational tables in an XML collection.

**default casting function.** Casts the SQL base type to a UDT.

**default view.** A representation of data in which an XML table and all of its related side tables are joined.

**distinct type.** See *user-defined type*.

**Document Access Definition (DAD).** Used to define the indexing scheme for an XML column or mapping scheme of an XML collection. It can be used to enable an XML Extender column of an XML collection, which is XML formatted.

**Document type definition (DTD).** A set of declarations for XML elements and attributes. The DTD defines what elements are used in the XML document, in what order they can be used, and which elements can contain other elements. You can associate a DTD with a document access definition (DAD) file to validate XML documents.

**double-byte character large object (DBCLOB).** A character string of double-byte characters, or a combination of single-byte and double-byte characters, where the string can be up to 2 GB. DBCLOBs have an

associated code page. Text objects that include double-byte characters are stored in a DB2 database as DBCLOBs.

**DTD.** (1) . (2) See *Document type definition*.

**DTD reference table (DTD\_REF table).** A table that contains DTDs, which are used to validate XML documents and to help applications to define a DAD. Users can insert their own DTDs into the DTD\_REF table. This table is created when a database is enabled for XML.

**DTD\_REF table.** DTD reference table.

**DTD repository.** A DB2 table, called DTD\_REF, where each row of the table represents a DTD with additional metadata information.

**EDI.** Electronic Data Interchange.

**Electronic Data Interchange (EDI).** A standard for electronic data interchange for business-to-business (B2B) applications.

**element.** See *XML element*.

**element\_node.** A representation of an element. An element\_node can be the root element or a child element.

**embedded SQL.** SQL statements coded within an application program. See *static SQL*.

**Extensible Stylesheet language (XSL).** A language used to express stylesheets. XSL consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics.

**Extensive Stylesheet Language Transformation (XSLT).** A language used to transform XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML.

**external file.** A file that exists in a file system external to DB2.

**foreign key.** A key that is part of the definition of a referential constraint and that consists of one or more columns of a dependent table.

**full text search.** Using the DB2 Text Extender, a search of text strings anywhere without regard to the document structure.

**index.** A set of pointers that are logically ordered by the values of a key. Indexes provide quick access to data and can enforce uniqueness on the rows in the table.

**Java Database Connectivity (JDBC).** An application programming interface (API) that has the same characteristics as Open Database Connectivity (ODBC) but is specifically designed for use by Java database

applications. Also, for databases that do not have a JDBC driver, JDBC includes a JDBC to ODBC bridge, which is a mechanism for converting JDBC to ODBC; JDBC presents the JDBC API to Java database applications and converts this to ODBC. JDBC was developed by Sun Microsystems, Inc. and various partners and vendors.

**JDBC.** Java Database Connectivity.

**join.** A relational operation that allows for retrieval of data from two or more tables based on matching column values.

**joined view.** A DB2 view created by the "CREATE VIEW" statement which join one more tables together.

**large object (LOB).** A sequence of bytes, where the length can be up to 2 GB. A LOB can be of three types: *binary large object* (BLOB), *character large object* (CLOB), or *double-byte character large object* (DBCLOB).

**LOB.** Large object.

**local file system.** A file system that exists in DB2

**location path.** Location path is a sequence of XML tags that identify an XML element or attribute. The location path identifies the structure of the XML document, indicating the context for the element or attribute. A single slash (/) path indicates that the context is the whole document. The location path is used in the extracting UDFs to identify the elements and attributes to be extracted. The location path is also used in the DAD file to specify the mapping between an XML element, or attribute, and a DB2 column when defining the indexing scheme for XML column. Additionally, the location path is used by the Text Extender for structural-text search.

**locator.** A pointer which can be used to locate an object. In DB2, the large object block (LOB) locator is the data type which locates LOBs.

**mapping scheme.** A definition of how XML data is represented in a relational database. The mapping scheme is specified in the DAD. The XML Extender provides two types of mapping schemes: *SQL mapping* and *relational database node (RDB\_node) mapping*.

**metadata table.** See *administrative support table*.

**multiple occurrence.** An indication of whether a column element or attribute can be used more than once in a document. Multiple occurrence is specified in the DAD.

**node.** In database partitioning, synonymous with database partition.



**object.** In object-oriented programming, an abstraction consisting of data and the operations associated with that data.

**ODBC.** Open Database Connectivity.

**Open Database Connectivity.** A standard application programming interface (API) for accessing data in both relational and nonrelational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call level interface (CLI) specification of the X/Open SQL Access Group and was developed by Digital Equipment Corporation (DEC), Lotus, Microsoft, and Sybase. Contrast with *Java Database Connectivity*.

**overloaded function.** A function name for which multiple function instances exist.

**partition.** A fixed-size division of storage.

**path expression.** See *location path*.

**predicate.** An element of a search condition that expresses or implies a comparison operation.

**primary key.** A unique key that is part of the definition of a table. A primary key is the default parent key of a referential constraint definition.

**procedure.** See *stored procedure*.

**query.** A request for information from the database based on specific conditions; for example, a query might be a request for a list of all customers in a customer table whose balance is greater than 1000.

**RDB\_node.** Relational database node.

**RDB\_node mapping.** The location of the content of an XML element, or the value of an XML attribute, which are defined by the RDB\_node. The XML Extender uses this mapping to determine where to store or retrieve the XML data.

**relational database node (RDB\_node).** A node that contains one or more element definitions for tables, optional columns, and optional conditions. The tables and columns are used to define how the XML data is stored in the database. The condition specifies either the criteria for selecting XML data or the way to join the XML collection tables.

**result set.** A set of rows returned by a stored procedure.

**result table.** A table which contains rows as the result of an SQL query or an execution of a stored procedure.

**root element.** The top element of an XML document.

**root ID.** A unique identifier that associates all side tables with the application table.

**scalar function.** An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments enclosed in parentheses.

**schema.** A collection of database objects such as tables, views, indexes, or triggers. It provides a logical classification of database objects.

**section search.** Provides the text search within a section which can be defined by the application. To support the structural text search, a section can be defined by the Xpath's abbreviated location path.

**side table.** Additional tables created by the XML Extender to improve performance when searching elements or attributes in an XML column.

**simple location path.** A sequence of element type names connected by a single slash (/).

**SQL mapping.** A definition of the relationship of the content of an XML element or value of an XML attribute with relational data, using one or more SQL statements and the XSLT data model. The XML Extender uses the definition to determine where to store or retrieve the XML data. SQL mapping is defined with the SQL\_stmt element in the DAD.

**static SQL.** SQL statements that are embedded within a program, and are prepared during the program preparation process before the program is executed. After being prepared, a static SQL statement does not change, although values of host variables specified by the statement may change.

**stored procedure.** A block of procedural constructs and embedded SQL statements that is stored in a database and can be called by name. Stored procedures allow an application program to be run in two parts. One part runs on the client and the other part runs on the server. This allows one call to produce several accesses to the database.

**structural text index.** To index text strings based on the tree structure of the XML document, using the DB2 Text Extender.

**subquery.** A full SELECT statement that is used within a search condition of an SQL statement.

**table space.** An abstraction of a collection of containers into which database objects are stored. A table space provides a level of indirection between a database and the tables stored within the database. A table space:

- Has space on media storage devices assigned to it.
- Has tables created within it. These tables will consume space in the containers that belong to the table space. The data, index, long field, and LOB

portions of a table can be stored in the same table space, or can be individually broken out into separate table spaces.

**text\_node.** A representation of the CDATA text of an element.

**top element\_node.** A representation of the root element of the XML document in the DAD.

**UDF.** See *user-defined function*.

**UDT.** See *user-defined type*.

**uniform resource locator (URL).** An address that names an HTTP server and optionally a directory and file name, for example:  
<http://www.ibm.com/data/db2/extenders>.

**UNION.** An SQL operation that combines the results of two select statements. UNION is often used to merge lists of values that are obtained from several tables.

**URL.** Uniform resource locator.

**user-defined function (UDF).** A function that is defined to the database management system and can be referenced thereafter in SQL queries. It can be one of the following functions:

- An external function, in which the body of the function is written in a programming language whose arguments are scalar values, and a scalar result is produced for each invocation.
- A sourced function, implemented by another built-in or user-defined function that is already known to the DBMS. This function can be either a scalar function or column (aggregating) function, and returns a single value from a set of values (for example, MAX or AVG).

**user-defined type (UDT).** A data type that is not native to the database manager and was created by a user. See *distinct type*.

**user table.** A table that is created for and used by an application.

**validation.** The process of using a DTD to ensure that the XML document is valid and to allow structured searches on XML data. The DTD is stored in the DTD repository.

**valid document.** An XML document that has an associated DTD. To be valid, the XML document cannot violate the syntactic rules specified in its DTD.

**Web browser.** A client program that initiates requests to a Web server and displays the information that the server returns.

**well-formed document.** An XML document that does not contain a DTD. Although in the XML specification, a document with a valid DTD must also be well-formed.

**XML.** eXtensible Markup Language.

**XML attribute.** Any attribute specified by the ATTLIST under the XML element in the DTD. The XML Extender uses the location path to identify an attribute.

**XML collection.** A collection of relation tables which presents the data to compose XML documents, or to be decomposed from XML documents.

**XML column.** A column in the application table that has been enabled for the XML Extender UDTs.

**XML element.** Any XML tag or ELEMENT as specified in the XML DTD. The XML Extender uses the location path to identify an element.

**XML object.** Equivalent to an XML document.

**XML Path Language.** A language for addressing parts of an XML document. XML Path Language (XPath) is designed to be used by XSLT. Every location path can be expressed using the syntax defined for XPath. The XPath subset implemented in XML Extender is called *location path* in this book

**XML table.** An application table which includes one or more XML Extender columns.

**XML tag.** Any valid XML markup language tag, mainly the XML element. The terms tag and element are used interchangeably.

**XML UDF.** A DB2 user-defined function provided by the XML Extender.

**XML UDT.** A DB2 user-defined type provided by the XML Extender.

**XPath.** A language for addressing parts of an XML document. The XPath subset implemented in XML Extender is called *location path* in this book

**XPath data model.** The tree structure used to model and navigate an XML document using nodes.

**XSL.** XML Stylesheet Language.

**XSLT.** XML Stylesheet Language Transformation.

---

# Index

## A

- access and storage method
  - choosing an 46
  - planning 46
  - XML collections 52, 53
  - XML columns 52, 53
- access method
  - choosing an 46
  - introduction 5
  - planning an 46
  - XML collections 8
  - XML column 7
- access privileges 43
- adding
  - nodes 86, 91, 98
  - side tables 75, 76
- administration
  - dxxadm command 141
  - in OS/390 and z/OS 38
  - stored procedures 189
  - support tables
    - DTD\_REF 213
    - XML\_USAGE 213
  - tasks 69
  - tools 5, 45
- administration stored procedures
  - dxxDisableCollection() 197
  - dxxDisableColumn() 195
  - dxxDisableSRV() 193
  - dxxEnableCollection() 196
  - dxxEnableColumn() 194
  - dxxEnableSRV() 192
- administration wizard
  - Disable a Column window 81
  - Enable a Column window 78
  - logging in 67
  - setting up 65
  - Side-table window 75
  - specifying address 67
  - specifying JDBC driver 67
  - specifying user ID and password 67
  - starting 65
- administrative support tables
  - DTD\_REF 213
  - XML\_USAGE 213
- application programming in OS/390 and z/OS 38
- attribute\_node 54, 60
- authorization requirements 45
- available operating systems 3

## B

- B-tree indexing 50
- backup 45
- bibliography xiii
- binding stored procedures 191

## C

- calling stored procedures 189

- casting
  - default, functions 107
  - parameter markers 119
- casting function
  - managing XML column data 107
  - retrieval 110, 158
  - storage 108, 154
  - update 114, 183
- CCSID, declare in USS 123, 131, 257
- cleaning up, getting started 32
- client code page 251
- CLOB limit, increasing for stored procedures 190
- code pages
  - client 251
  - considerations 251
  - consistent encoding in USS 257
  - consistent encodings and declarations 255
  - conversion scenarios 258
  - database 251
  - DB2 assumptions 251
  - declaring an encoding 258
  - document encoding consistency 251, 252, 259
  - encoding declaration 254
  - exporting documents 253
  - importing documents 252
  - legal encoding declarations 254
  - line endings 251, 260
  - preventing inconsistent documents 259
  - server 251
  - supported encoding declarations 254
  - terminology 251
  - UDFs and stored procedures 252, 253
  - Windows NT UTF-8 limitation 259
  - XML Extender assumptions 251
- codes
  - messages and 221
  - SQLSTATE 217
- column data
  - available UDTs 48
  - storing XML documents as 73
- column type, for decomposition 60
- command options
  - disable\_column 147
- compatibility mode 42
- composing XML documents 27
- composite key
  - for decomposition 59
  - XML collections 59
- composition
  - dxxGenXML() 121, 123
  - dxxRetrieveXML() 121, 125
  - of XML documents 31
  - overriding the DAD file 126
  - stored procedures
    - dxxGenXML() 31, 199
    - dxxRetrieveXML() 203
  - XML collection 121

- conditions
  - optional 59
  - RDB\_node mapping 59
  - SQL mapping 55, 58
- consistent document 251
- Content() function
  - for retrieval 110
  - retrieval functions using 158
  - XMLCLOB to an external server file 162
  - XMLFile to a CLOB 160
  - XMLVarchar to an external server file 161
- conversion of code pages 258
- creating
  - a database 15, 26
  - DAD file 73
  - DB2XML schema 69, 72
  - indexes 20, 81
  - nodes 86, 91, 98
  - side tables 75, 76
  - UDFs 69, 72
  - UDTs 69, 72
  - XML collections 26
  - XML columns 16
  - XML table 76

## D

- DAD
  - node definitions
    - RDB\_node 59
- DAD file
  - attribute\_node 54
  - bind step for USS encodings 257
  - CCSIDs in USS 123, 131, 257
  - creating for XML collections 26
    - from the command line 87, 92, 98
    - RDB\_node mapping 89, 95
    - SQL mapping 84
  - creating for XML columns 17, 73
    - from the command line 75
    - using the administration wizard 73
  - declaring the encoding 258
  - defining side tables 14
  - DTD for the 237
  - editing for XML collections
    - from the command line 87, 92, 98
  - editing for XML columns
    - from the command line 75
    - using the administration wizard 73
  - element\_node 53, 59
  - examples of 244
    - RDB\_node mapping 248
    - SQL mapping 245
  - for XML collections 26
  - for XML column 77
  - for XML columns 52, 53
  - introduction to the 6
  - mapping scheme 26, 83
  - node definitions
    - attribute\_node 53
    - element\_node 53

- DAD file (*continued*)
  - node definitions (*continued*)
    - root\_node 53
    - text\_node 53
  - overriding the 126
  - planning for the 52, 53
    - tutorial 25
    - XML collections 52
    - XML column 17, 52
  - RDB\_node 59
  - root element\_node 59
  - root\_node 53
  - samples of 244
  - size limit 52, 53, 263
  - text\_node 54
- data types
  - XMLCLOB 151
  - XMLFile 151
  - XMLVarchar 151
- database
  - code page 251
  - creating 15, 26
  - enabling for XML 69, 72
  - relational 55
- DB2
  - and XML documents 3
  - composing XML documents 8
  - decomposing XML documents 8
  - integrating XML documents 5
  - storing untagged XML data 8
  - storing XML documents 5
- DB2XML 214
  - DTD\_REF table schema 213
  - schema for stored procedures 9
  - schema for UDFs 8
  - schema for UDFs and UDTs 107
  - schema for UDTs 8
  - XML\_USAGE table schema 213
- decomposition
  - collection table limit 263
  - composite key 59
  - DB2 table sizes 60, 130
  - dxxInsertXML() 131, 133
  - dxxShredXML() 131
  - of XML collections 130
  - specifying the column type for 60
  - specifying the orderBy attribute 59
  - specifying the primary key for 59
  - stored procedures
    - dxxInsertXML() 210
    - dxxShredXML() 207
- default view, side tables 50
- deleting
  - nodes 86, 91, 98
  - side tables 76
  - XML documents 119
- Disable a Column window 81
- disable\_collection subcommand 149
- disable\_column command 147
- disable\_server subcommand 145

- disabling
  - administration command 141
  - disable\_collection subcommand 149
  - disable\_column command 147
  - disable\_server subcommand 145
  - stored procedure 195, 197
  - XML collections
    - from the command line 103
    - stored procedure 197
    - using the administration wizard 103
  - XML columns
    - from the command line 82
    - stored procedure 195
    - using the administration wizard 81
- document access definition (DAD) 73
- document encoding declaration 251
- document type definition 70
- DTD
  - availability 4
  - for getting started lessons 13, 23
  - for the DAD 237
  - for validation 51
  - inserting 16
  - inserting from the command line 72
  - planning 13, 23
  - publication 4
  - repository
    - DTD\_REF 6, 213
    - storing in 70
  - structured searches 52
  - using multiple 51, 52
  - validating with a 52
- DTD\_REF table 70
  - column limits 263
  - inserting a DTD 71
  - schema 213
- DTDID 71, 213, 214
- DXX\_SEQNO for multiple occurrence 49
- dxxadm command
  - disable\_collection subcommand 149
  - disable\_column command 147
  - disable\_server subcommand 145
  - enable\_collection subcommand 148
  - enable\_column subcommand 146
  - enable\_server 70
  - enable\_server subcommand 143
  - introduction to 141
  - syntax 142
- dxxDisableCollection() stored procedure 197
- dxxDisableColumn() stored procedure 195
- dxxDisableSRV() stored procedure 193
- dxxEnableCollection() stored procedure 196
- dxxEnableColumn() stored procedure 194
- dxxEnableSRV() stored procedure 192
- dxxGenXML() 31
- dxxGenXML() stored procedure 121, 199
- DXXGPREP, initializing the XML Extender 39
- dxxInsertXML() stored procedure 131, 210
- dxxRetrieveXML() stored procedure 121, 203
- DXXROOT\_ID 19, 51
- dxxShredXML() stored procedure 131, 207
- dxxtrc command 232, 233, 234
- dynamically overriding the DAD file, composition 126

## E

- editing
  - side tables 75, 76
  - XML table 76
- element\_node 53, 59
- Enable a Column window 78
- enable\_collection subcommand 148
- enable\_column subcommand 146
- enable\_db command
  - creating XML\_USAGE table 213, 214
- enable\_server subcommand 143
- enabling
  - administration command 141
  - database tasks 69, 72
  - databases for XML
    - from the command line 72
    - using the administration wizard 69, 72
  - enable\_collection subcommand 148
  - enable\_column subcommand 146
  - enable\_server subcommand 143
  - servers for XML
    - from the command line 70
    - stored procedure 194, 196
  - XML collections
    - from the command line 102
    - requirements 130
    - stored procedure 196
    - using the administration wizard 102
  - XML columns
    - for &text; 118
    - from the command line 79
    - from the command shell 19
    - stored procedure 194
    - using the administration wizard 78
- encoding
  - bind step for USS 257
  - CCSID declarations in USS 123, 131, 257
  - consistent declarations 255
  - conversion 258
  - conversion by DB2 252, 253, 259
  - declaration considerations 254
  - declarations 251, 258
  - legal, declarations 254
  - of a document 251
  - supported declarations 254
  - XML documents 251
- existing DB2 data 8
- eXtensible Markup Language (XML) 4
- Extensive Stylesheet Language Transformation 61
- extractChar() function 171
- extractChars() function 171
- extractCLOB() function 175
- extractCLOBs() function 175
- extractDate() function 177
- extractDates() function 177
- extractDouble() function 167
- extractDoubles() function 167
- extracting functions
  - description of 153

## extracting functions *(continued)*

- extractChar() 171
- extractChars() 171
- extractCLOB() 175
- extractCLOBs() 175
- extractDate() 177
- extractDates() 177
- extractDouble() 167
- extractDoubles() 167
- extractInteger() 164
- extractIntegers() 164
- extractReal() 169
- extractReals() 169
- extractSmallint() 165
- extractSmallints() 165
- extractTime() 179
- extractTimes() 179
- extractTimestamp() 181
- extractTimestamps() 181
- extractVarchar() 173
- extractVarchars() 173
- introduction to 163
- table of 113

extractInteger() function 164

extractIntegers() function 164

extractReal() function 169

extractReals() function 169

extractSmallint() function 165

extractSmallints() function 165

extractTime() function 179

extractTimes() function 179

extractTimestamp() function 181

extractTimestamps() function 181

extractVarchar() function 173

extractVarchars() function 173

## F

FROM clause 58

full text search 8

function path, adding DB2XML schema 107

functions

- casting 108, 109, 110, 113, 114
- Content(): from XMLCLOB to file 162
- Content(): from XMLFILE to CLOB 160
- Content(): from XMLVARCHAR to file 161
- extractChar() 171
- extractChars() 171
- extractCLOB() 175
- extractCLOBs() 175
- extractDate() 177
- extractDates() 177
- extractDouble() 167
- extractDoubles() 167
- extracting 153, 163
- extractInteger() 164
- extractIntegers() 164
- extractReal() 169
- extractReals() 169
- extractSmallint() 165
- extractSmallints() 165
- extractTime() 179

## functions *(continued)*

- extractTimes() 179
- extractTimestamp() 181
- extractTimestamps() 181
- extractVarchar() 173
- extractVarchars() 173
- for XML columns 153
- generate\_unique 153, 188
- limitations when invoking from JDBC 119
- limits 263
- retrieval 109, 110
  - description 153
  - from external storage to memory pointer 158
  - from internal storage to external server file 158
  - introduction 158
- storage 108, 153, 154
- summary table of 154
- update 113, 114, 153, 183
- XMLCLOB to an external server file 162
- XMLCLOBFromFile() 156
- XMLFile to a CLOB 160
- XMLFileFromCLOB() 158
- XMLFileFromVarchar() 157
- XMLVarchar to an external server file 161
- XMLVarcharFromFile() 155

## G

generate\_unique() function 188

generate\_unique function

- description of 153
- introduction to 188

getting started lessons

- cleaning up 32
- collection tables 22
- composing the XML document 31
- creating DAD files 17, 25, 26, 28
- creating indexes 20
- creating the database 15, 26
- creating the XML collection 26
- creating the XML column 16
- defining side tables 14
- inserting the DTD 16
- introduction 11
- overview 11
- planning 13, 23
- searching the XML document 21
- storing the XML document 21

getting started scripts 15, 25

goal mode 42

## H

highlighting conventions x

## I

importing the DTD 70

include files for stored procedures 189

inconsistent document 251

indexes, for side tables 20, 81

- indexing
  - considerations 50
  - multiple indexes 50
  - ROOT ID 50
  - side tables 50
  - Text Extender structural-text 51
  - with side tables 14, 50
  - with Text Extender 50
  - with XML columns 50
  - XML columns 50
  - XML documents 50
  - XML documents with multiple occurrence 50
- initializing
  - the XML Extender 39
- installing
  - the XML Extender 37
- invoking the administration wizard 66

## J

- JCL jobs 38
- JDBC, limitations when invoking functions 154
- JDBC, limitations when invoking UDFs 119
- JDBC address, for wizard 67
- JDBC driver, for wizard 67
- join conditions
  - RDB\_node mapping 59
  - SQL mapping 58

## L

- limits
  - stored procedure parameters 121, 213
  - The XML Extender 263
- line endings
  - code page considerations 251, 260
- location path
  - introduction to 6, 61
  - simple 62
  - syntax 61
  - usage 62
  - XPath 6, 61
  - XSL 6, 61
  - XSLT 61
- logging in, for wizard 67

## M

- maintaining document structure 7
- management
  - retrieving column data 109
  - searching XML documents 115
  - storing column data 108
  - updating column data 113
  - XML collection data 121
- mapping scheme
  - creating the DAD for the 26, 83
  - determining RDB\_node mapping 56
  - determining SQL mapping 55
  - editing the DAD for the 83
  - figure of DAD for the 48

- mapping scheme (*continued*)
  - for XML collections 48
  - for XML columns 48
  - FROM clause 58
  - introduction 8
  - ORDER BY clause 58
  - RDB\_node mapping requirements 59
  - requirements 57
  - SELECT clause 57
  - SQL mapping requirements 57
  - SQL mapping scheme 57
  - SQL\_stmt 55
  - WHERE clause 58
- messages and codes 221
- multiple DTDs
  - XML collections 52
  - XML columns 51
- multiple occurrence
  - affecting table size 60, 130
  - deleting elements and attributes 135
  - DXX\_SEQNO 49
  - indexing XML documents with 50
  - one column per side table 49
  - order of elements and attributes 130
  - orderBy attribute 59
  - preserving the order of elements and attributes 136
  - recomposing documents with 59
  - searching elements and attributes 117
  - updating collections 134
  - updating elements and attributes 115, 135, 186
  - updating XML documents 115, 186
- multiple-occurrence attribute 29
- MVS environment 38

## N

- nodes
  - add new 86, 91, 98
  - attribute\_node 54
  - creating 86, 91, 98
  - DAD file configuration 28, 88, 92, 99
  - deleting 86, 91, 98
  - element\_node 53
  - RDB\_node 59
  - removing 86, 91, 98
  - root, creating 86
  - root\_node 53
  - text\_node 54

## O

- operating environment, OS/390 and z/OS 38
- operating systems supported 3
- ORDER BY clause 58
- orderBy attribute
  - for decomposition 59
  - for multiple occurrence 59
  - XML collections 59
- overloaded function, Content() 158
- overrideType
  - No override 126
  - SQL override 126

overrideType (continued)  
XML override 126  
overriding the DAD file 126

## P

parameter markers in functions 119, 154  
performance  
default views of side tables 50  
indexing side tables 50  
searching XML documents 50  
stopping the trace 234  
performance objectives 42  
planning  
a mapping scheme 55  
choosing a storage method 46  
choosing an access and storage method 46  
choosing an access method 46  
choosing to validate XML data 51, 52  
determining column UDT 13, 48  
determining document structure 23  
determining DTD 13, 23  
for the DAD 52, 53  
for XML collections 53  
for XML columns 48, 52  
getting started lessons 13, 23  
how to search XML column data 48  
mapping XML document and database 14, 24  
searching elements and attributes 13  
side tables 49  
the XML collections mapping scheme 55  
to index XML columns 50  
validating with multiple DTDs 51, 52  
primary key for decomposition 59  
primary key for side tables 19, 50, 51  
problem determination 215  
processing instructions 54, 95

## R

RDB\_node mapping  
composite key for decomposition 59  
conditions 59  
creating a DAD 89, 95  
decomposition requirements 59  
determining for XML collections 56  
requirements 59  
specifying column type for decomposition 60  
recovery 45  
registry variable, DB2CODEPAGE 251  
related information xiii  
relational tables 121  
removing  
nodes 86, 91, 98  
side tables 76  
repository, DTD 70  
retrieval functions  
Content() 158  
description of 153  
from external storage to memory pointer 158  
from internal storage to external server file 158  
introduction to 158

retrieval functions (continued)  
XMLCLOB to an external server file 162  
XMLFile to a CLOB 160  
XMLVarchar to an external server file 161

retrieval of data  
attribute values 111  
element contents 111  
entire document 110  
return codes  
stored procedures 216  
UDF 215

ROOT ID  
default view of side tables 50  
indexing 50  
indexing considerations 50  
specifying 19, 79  
root\_node 53

## S

sample files 15, 25  
samples, getting started 39  
samples, tutorial 41  
schema  
DB2XML 69, 107  
DTD\_REF table 71, 213  
name for stored procedures 9  
name for user defined functions 8  
XML\_USAGE table 213  
searching  
direct query on side tables 116  
document structure 116  
from a joined view 116  
multiple occurrence 117  
side tables 21  
Text Extender structural text 117  
text with Text Extender 119  
with extracting UDFs 117  
XML collection 136  
XML documents 21, 115  
security 43  
SELECT clause 57  
server code page 251  
service class 41  
setting up  
administration wizard 65  
the XML Extender 37  
setting up XML collections  
adding the DAD 83  
creating the DAD 83  
disabling 103  
editing the DAD 83  
enabling 101  
setting up XML columns  
altering an XML table 76  
creating an XML table 76  
creating the DAD 73  
disabling 81  
editing an XML table 76  
editing the DAD 73  
enabling 77  
side tables  
add new 75, 76



- side tables *(continued)*
  - create 75
  - default view 50
  - deleting 75
  - DXX\_SEQNO 49
  - DXXROOT\_ID 19
  - editing 75, 76
  - getting started lessons 14
  - indexing 14, 50
  - multiple occurrence 49
  - planning 49
  - removing 75, 76
  - ROOT ID 19
  - searching 14, 21, 115
  - specifying ROOT ID 79
  - updating 114
- sizes, limits 263
- software requirements 37
- SQL DELETE, to delete XML documents 119
- SQL mapping
  - creating a DAD 28, 84
  - determining for XML collections 55
  - FROM clause 58
  - ORDER BY clause 58
  - requirements 57
  - SELECT clause 57
  - SQL mapping scheme 57
  - WHERE clause 58
- SQL override 126
- SQL\_stmt
  - FROM clause 58
  - ORDER\_BY clause 58
  - SELECT clause 57
  - WHERE clause 58
- SQLSTATE codes 217
- starting
  - administration wizard 65
  - the administration wizard 66
  - the XML Extender 37
- storage functions
  - description of 153
  - introduction to 154
  - storage UDF table 109
  - XMLCLOBFromFile() 156
  - XMLFileFromCLOB() 158
  - XMLFileFromVarchar() 157
  - XMLVarcharFromFile() 155
- storage method
  - choosing a 46
  - introduction 5
  - planning a 46
  - XML collections 8
  - XML column 7
- storage UDFs 109, 114
- stored procedures
  - administration 189, 191
    - dxxDisableCollection() 197
    - dxxDisableColumn() 195
    - dxxDisableSRV() 193
    - dxxEnableCollection() 196
    - dxxEnableColumn() 194
- stored procedures *(continued)*
  - administration 189, 191 *(continued)*
    - dxxEnableSRV() 192
  - calling 189
  - code page considerations 252, 253, 259
  - composition 189, 198
    - dxxGenXML() 199
    - dxxRetrieveXML() 203
  - decomposition 189, 206
    - dxxInsertXML() 210
    - dxxShredXML() 207
  - dxxDisableCollection() 197
  - dxxDisableColumn() 195
  - dxxDisableSRV() 193
  - dxxEnableCollection() 196
  - dxxEnableColumn() 194
  - dxxEnableSRV() 192
  - dxxGenXML() 31, 121, 199
  - dxxInsertXML() 131, 210
  - dxxRetrieveXML() 121, 203
  - dxxShredXML() 131, 207
  - include files 189
  - initializing with DXXGPREP 191
  - return codes 216
  - update 189
- storing the DTD 70
- structure
  - hierarchical 24
  - of DTD 24
  - of mapping 14, 24
  - of XML document 24
  - relational tables 14, 24
- stylesheets 54, 95
- subcommands, dxxadm
  - disable\_collection 149
  - disable\_server 145
  - enable\_collection 148
  - enable\_column 146
  - enable\_server 143
- syntax diagram
  - disable\_collection subcommand 149
  - disable\_column command 147
  - disable\_server subcommand 145
  - dxxadm 142
  - enable\_collection subcommand 148
  - enable\_column subcommand 146
  - enable\_server subcommand 143
  - extractChar() function 171
  - extractChars() function 171
  - extractCLOB() function 175
  - extractCLOBs() function 175
  - extractDate() function 177
  - extractDates() function 177
  - extractDouble() function 167
  - extractDoubles() function 167
  - extractInteger() function 164
  - extractIntegers() function 164
  - extractReal() function 169
  - extractReals() function 169
  - extractSmallint() function 165
  - extractSmallints() function 165

- syntax diagram (*continued*)
  - extractTime() function 179
  - extractTimes() function 179
  - extractTimestamp() function 181
  - extractTimestamps() function 181
  - extractVarchar() function 173
  - extractVarchars() function 173
  - generate\_unique() function 188
  - how to read xi
  - location path 61
  - Update() function 183
  - XMLCLOB to an external server file Content() function 162
  - XMLCLOBFromFile() function 156
  - XMLFile to a CLOB Content() function 160
  - XMLFileFromCLOB() function 158
  - XMLFileFromVarchar() function 157
  - XMLVarchar to an external server file Content() function 161
  - XMLVarcharFromFile() function 155

## T

- table of UDFs 154
- tables sizes, for decomposition 60, 130
- Text Extender
  - enabling for search 118
  - enabling XML columns for 118
  - searching with 118
- text\_node 54, 60
- tracing
  - dxtrc command 232
  - starting 233
  - stopping 234
- transfer of documents between client and server, considerations 251
- troubleshooting 215
  - messages and codes 221
  - SQLSTATE codes 217
  - stored procedure return codes 216
  - tracing 232
  - UDF return codes 215
- TSO environment 38

## U

- UDFs
  - code page considerations 252, 253, 259
  - extractChar() 171
  - extractChars() 171
  - extractCLOB() 175
  - extractCLOBs() 175
  - extractDate() 177
  - extractDates() 177
  - extractDouble() 167
  - extractDoubles() 167
  - extracting functions 163
  - extractInteger() 164
  - extractIntegers() 164
  - extractReal() 169
  - extractReals() 169
  - extractSmallint() 165

- UDFs (*continued*)
  - extractSmallints() 165
  - extractTime() 179
  - extractTimes() 179
  - extractTimestamp() 181
  - extractTimestamps() 181
  - extractVarchar() 173
  - extractVarchars() 173
  - for XML columns 153
  - from external storage to memory pointer 158
  - from internal storage to external server file 158
  - generate\_unique() 188
  - limitations when invoking from JDBC 154
  - retrieval functions 158
  - return codes 215
  - searching with 117
  - storage 114
  - summary table of 154
  - Update() 114, 183
  - XMLCLOB to an external server file 162
  - XMLCLOBFromFile() 156
  - XMLFile to a CLOB 160
  - XMLFileFromCLOB() 158
  - XMLFileFromVarchar() 157
  - XMLVarchar to an external server file 161
  - XMLVarcharFromFile() 155
- UDTs
  - introduction to 8
  - summary table of 48
  - XML table 77
  - XMLCLOB 48
  - XMLFILE 48
  - XMLVARCHAR 48
- unique key column, generating 188
- Update() function 114, 183
- update function
  - description of 153
  - document replacement behavior 184
  - introduction to 183
- updating
  - how the Update() UDF replaces XML documents 184
  - side tables 114
  - XML column data 113
    - attributes 114
    - entire document 114
    - multiple occurrence 115, 186
    - specific elements 114
- usage for the location path 62
- user-defined functions (UDFs)
  - for XML columns 153
  - full text search 8
  - generate\_unique() 188
  - schema 8
  - searching with 117
  - summary table of 154
  - Update() 114, 183
- user-defined types
  - for XML columns 107
  - XMLCLOB 107
  - XMLFILE 107

user-defined types (*continued*)  
XMLVARCHAR 107  
user-defined types (UDTs) 7  
user ID and password, for wizard 67

## V

validate XML data  
  considerations 51, 52  
  deciding to 51, 52  
  DTD requirements 51, 52  
validating  
  DTD 70  
  performance impact 52, 53  
  using a DTD 51  
  XML data 51

## W

WHERE clause 58  
Windows NT UTF-8 limitation, code pages 259  
Workload Manager (WLM) 41

## X

XML 4  
XML applications 4  
XML collections  
  composition 121  
  creating 26  
  creating a DAD  
    RDB\_node mapping 89, 95  
    SQL mapping 84  
  creating the DAD  
    from the command line 87, 92, 98  
  DAD file, planning for 52  
  decomposition 130  
  definition of 6  
  determining a mapping scheme for 55  
  disabling 103  
    from the command line 103  
    using the administration wizard 103  
  DTD for validation 70  
  editing the DAD  
    from the command line 87, 92, 98  
  enabling 101  
    from the command line 102  
    using the administration wizard 102  
  introduction to 8  
  managing XML collection data 121  
  mapping scheme 55  
    creating the DAD 83  
    editing the DAD 83  
  mapping schemes 55  
  RDB\_node mapping 56  
  scenarios 47  
  searching 136  
  setting up 83  
  SQL mapping 55  
  storage and access methods 5, 8  
  validation 70

XML collections (*continued*)  
  when to use 47  
XML columns 109  
  adding 18  
  configuring 73  
  creating 16  
  creating the DAD 17  
    from the command line 75  
    using the administration wizard 73  
  DAD file, planning for 52  
  default view of side tables 50  
  definition of 6  
  determining column UDT 48  
  disabling  
    from the command line 82  
    using the administration wizard 81  
  editing the DAD  
    from the command line 75  
    using the administration wizard 73  
  elements and attributes to be searched 48  
  enabling 19  
    from the command line 79  
    using the administration wizard 78  
  figure of side tables 49  
  indexing 50  
  introduction to 7  
  location path 61  
  maintaining document structure 7  
  managing 107  
  planning 48  
  preparing the DAD 17  
  retrieving data  
    attribute values 111  
    element contents 111  
    entire document 110  
  sample DAD file 244  
  scenarios 46  
  setting up 73  
  side tables 20  
  storage and access methods 5, 7  
  storing data 108  
  the DAD for 52  
  UDFs 153  
  updating XML data  
    attributes 114  
    entire document 114  
    specific elements 114  
  view columns 20  
  view side tables 20  
  when to use 46  
  with side tables 50  
  XML type 18  
XML documents  
  B-tree indexing 50  
  code page assumptions 251  
  code page consistency 251, 252, 253, 259  
  composing 27, 121  
  creating indexes 20, 81  
  decomposition 130, 131  
  default casting functions 21  
  deleting 119

- XML documents *(continued)*
  - encoding declarations 254
  - exporting, code page conversion 253
  - importing, code page conversion 252, 259
  - indexing 50
  - introduction to 3
  - legal encoding declarations 254
  - mapping to tables 14, 24
  - searching 21, 115
    - direct query on side tables 116
    - document structure 116
    - from a joined view 116
    - multiple occurrence 117
    - Text Extender structural text 117
    - with extracting UDFs 117
  - stored in DB2 3
  - storing 21
  - supported encoding declarations 254
- XML DTD repository
  - DTD Reference Table (DTD\_REF) 6
  - introduction to 6
- XML Extender
  - available operating systems 3
  - capabilities 6
  - features 6
  - functions 153
  - initializing with DXXGPREP 39
  - installing 37
  - introduction to 3
- XML Extender stored procedures 189
- XML operating environment on OS/390 and z/OS 38
- XML override 126
- XML Path Language 6, 61
- XML repository 46
- XML table
  - creating 76
  - editing 76
- XML Toolkit for OS/390 and z/OS 11, 37
- XML\_USAGE table 213
- XMLCLOB to an external server file function 162
- XMLClobFromFile() function 156
- XMLFile to a CLOB function 160
- XMLFileFromCLOB() function 158
- XMLFileFromVarchar() function 157
- XMLVarchar to an external server file function 161
- XMLVarcharFromFile() function 155
- XPath 6, 61
- XSLT 55, 61

# Readers' Comments — We'd Like to Hear from You

DB2 Universal Database for OS/390 and z/OS  
XML Extender  
Administration and Programming  
Version 7

Publication No. SC26-9949-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you?  Yes  No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

\_\_\_\_\_  
Name

\_\_\_\_\_  
Address

\_\_\_\_\_  
Company or Organization

\_\_\_\_\_  
Phone No.



Fold and Tape

Please do not staple

Fold and Tape



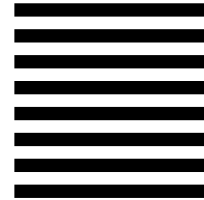
NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation  
Department HHX/H3 PO Box 49023  
SAN JOSE CA 95161-9023



Fold and Tape

Please do not staple

Fold and Tape





Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.

SC26-9949-00

