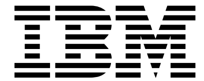
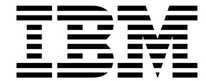
IBM® DB2® Universal Database



XML Extender Administration and Programming

Version 7

IBM® DB2® Universal Database



XML Extender Administration and Programming

Version 7

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 281.

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1999, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Tables. vii

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 ix
| What's new in this book for DB2 UDB Version
| Fixpak 2 x
Including this book in the DB2 UDB Version 7
Information Center xi
Highlighting conventions. xi
How to read syntax diagrams xii
Related information xiv

Part 1. Introduction. 1

Chapter 1. Introduction to the XML Extender 3
XML documents 3
XML applications 4
Why XML and DB2? 4
Integrating XML into DB2. 5
  Administration tools 5
  Storage and access methods 6
  DTD repository 6
  Document Access Definitions (DADs) . . . 6
  XML column: Structured document storage and retrieval 6
  XML collection: Integrated data management 10

Chapter 2. Getting started with XML Extender. 13
Scenario for the lessons 14
Lesson: Store an XML document in an XML column. 14
  The scenario 14
  Planning 15
  Setting up. 19
  Creating the XML column 20
Lesson: Composing an XML document . . . 27
  The tutorial scenario 27
  Planning 28
  Setting up. 31
  Creating the XML collection: preparing the DAD file 32
  Composing the XML document 38
Cleaning up the tutorial environment . . . 39

Part 2. Administration 41

Chapter 3. Preparing to use the XML Extender: administration 43
Set-up requirements 43
  Software requirements 43
  Installation requirements. 43
  Authorization requirements 43
Administration tools 44
Administration planning 44
  Choosing an access and storage method . 44
  Planning for XML columns 46
  Planning for XML collections 53

Chapter 4. Administering XML data . . . 65
Starting the administration wizard 65
  Setting up the administration wizard. . . 65
  Invoking the administration wizard . . . 67
Enabling a database for XML 69
  Using the administration wizard 69
  From the DB2 command shell 69
Storing a DTD in the DTD repository . . . 70
  Using the administration wizard 70
  From the DB2 command shell 71
Defining XML columns or collections . . . 71
Work with XML columns 72
  Creating or editing the DAD file 72
  Creating or altering an XML table. . . . 76
  Enabling XML columns 77
  Indexing side tables 80
  Disabling XML columns 81
Work with XML collections 83
  Creating or editing the DAD file for the mapping scheme 83
  Enabling XML collections 105
  Disabling XML collections 107
Disabling a database for XML. 108
  Before you begin 109
  Using the administration wizard 109

© Copyright IBM Corp. 1999, 2000

From the DB2 command shell. 109

Part 3. Programming 111

Chapter 5. Managing XML column data 113
UDT and UDF names 114
Storing data. 114
Retrieving data. 116
Retrieving an entire document 117
Retrieving element contents and attribute values. 119
Updating XML data 121
Searching XML documents. 123
Searching the XML document by structure 124
Using the Text Extender for structural text search. 126
Deleting XML documents 128
| Limitations when invoking functions from JDBC 129

Chapter 6. Managing XML collection data 131
Composing XML documents from DB2 data 131
Before you begin 131
Composing the XML document 132
Dynamically overriding values in the DAD file. 135
Decomposing XML documents into DB2 data 139
Enabling an XML collection for decomposition 140
Decomposition table size limits 140
Before you begin 140
Decomposing the XML document . . . 141
Accessing an XML collection 143
Updating data in an XML collection . . 144
Deleting an XML document from an XML collection 145
Retrieving XML documents from an XML collection 146
Searching an XML collection 146

Part 4. Reference. 149

Chapter 7. XML Extender administration command: dxxadm. 151
High-level syntax 151
Administration command options 151
enable_db 152
disable_db 153
enable_column. 155
disable_column 157
enable_collection 159
disable_collection 161

Chapter 8. XML Extender user-defined types 163

Chapter 9. XML Extender user-defined functions 165
Storage functions 166
XMLVarcharFromFile() 167
XMLCLOBFromFile() 168
XMLFileFromVarchar() 169
XMLFileFromCLOB() 170
Retrieval functions 171
Content(): retrieve from XMLFILE to a CLOB. 172
Content(): retrieve from XMLVARCHAR to an external server file 174
Content(): retrieval from XMLCLOB to an external server file 176
Extracting functions 178
extractInteger() and extractIntegers() . . 179
extractSmallint() and extractSmallints() 181
extractDouble() and extractDoubles() . . 182
extractReal() and extractReals() 184
extractChar() and extractChars() 185
extractVarchar() and extractVarchars() . . 186
extractCLOB() and extractCLOBs() . . . 188
extractDate() and extractDates() 190
extractTime() and extractTimes() 191
extractTimestamp() and extractTimestamps() 193
Update function 195
Purpose 195
Syntax 195
Parameters 195
Return type 195
Example 196
| Usage. 196

Chapter 10. XML Extender stored procedures 201
Specifying include files 201
Calling XML Extenders stored procedures 202
| Increasing the CLOB limit 202
Before you begin 203
Administration stored procedures 203
dxxEnableDB(). 204

dxxDisableDB() 205
dxxEnableColumn() 206
dxxDisableColumn() 208
dxxEnableCollection() 209
dxxDisableCollection() 210
Composition stored procedures 211
dxxGenXML() 212
dxxRetrieveXML() 216
Decomposition stored procedures 220
dxxShredXML() 221
dxxInsertXML() 223

Chapter 11. Administrative support tables 225
DTD reference table 225
XML usage table 226

Chapter 12. Diagnostic information . . . 227
Handling UDF return codes 227
Handling stored procedure return codes . . 228
SQLSTATE codes 228
Messages 233
Error messages. 233
Diagnostic tracing. 246
| Starting the trace 248
| Stopping the trace 249

Part 5. Appendixes 251

Appendix A. DTD for the DAD file . . . 253

Appendix B. Samples 261
XML DTD 261
XML document: getstart.xml 261
Document access definition files 262
DAD file: XML column 263
DAD file: XML collection - SQL mapping 263
DAD file: XML - RDB_node mapping . . 265

Appendix C. Code page considerations 269
| Terminology 269
| DB2 and XML Extender code page
| assumptions 270
Encoding declaration considerations . . . 272
Legal encoding declarations 272
Consistent encodings and encoding declarations. 273
Declaring an encoding 275
| Conversion scenarios 275
| Preventing inconsistent XML documents . . 277

| **Appendix D. The XML Extender limits . . 279**

Notices 281
Trademarks 283

Glossary 285

Index 291

Contacting IBM 299
Product Information 299

Tables

1. SALES_TAB table 14
2. Elements and attributes to be searched 17
3. Side-table columns to be indexed 25
4. The XML Extender UDTs 47
5. Simple location path syntax 52
6. The XML Extender's restrictions using location path 52
7. The schema for the DTD_REF DTD table 71
8. The XML Extender storage functions 114
9. The XML Extender default cast functions 115
10. The XML Extender storage UDFs 115
11. The XML Extender retrieval functions 116
12. The XML Extender default cast functions 117
13. The XML Extender extracting functions 120
14. enable_db parameters 152
15. disable_db parameters 153
16. enable_column parameters 155
17. disable_column parameters. 157
18. enable_collection parameters 159
19. disable_collection parameters 161
20. The XML Extender UDTs 163
21. The XML Extender user-defined functions 165
22. XMLVarcharFromFile parameter 167
23. XMLCLOBFromFile parameter 168
24. XMLFileFromVarchar parameters 169
25. XMLFileFromCLOB() parameters 170
26. XMLFILE to a CLOB parameter 172
27. XMLVarchar to external server file parameters 174
28. XMLCLOB to external server file parameters 176
29. extractInteger and extractIntegers function parameters 179
30. extractSmallint and extractSmallints function parameters 181
31. extractDouble and extractDoubles function parameters 182
32. extractReal and extractReals function parameters 184
33. extractChar and extractChars function parameters 185
34. extractVarchar and extractVarchars function parameters 186
35. extractCLOB and extractCLOBs function parameters 188
36. extractDate and extractDates function parameters 190
37. extractTime and extractTimes function parameters 191
38. extractTimestamp and extractTimestamps function parameters. 193
39. The UDF Update parameters 195
40. Update function rules 196
41. dxxEnableDB() parameters 204
42. dxxDisableDB() parameters. 205
43. dxxEnableColumn() parameters 206
44. dxxDisableColumn() parameters 208
45. dxxEnableCollection() parameters 209
46. dxxDisableCollection() parameters 210
47. dxxGenXML() parameters 212
48. dxxRetrieveXML() parameters 216
49. dxxShredXML() parameters 221
50. dxxInsertXML() parameters. 223
51. DTD_REF table. 225
52. XML_USAGE table 226
53. SQLSTATE codes and associated message numbers 228
54. Trace parameters 248
55. Using UDFs and stored procedures when the XML file is imported into the database 270
56. Using UDFs and stored procedures when the XML file is exported from the database 271
57. Encoding declarations supported by XML Extender 272
58. XML Extender limits 279

© Copyright IBM Corp. 1999, 2000

About this book

This section describes the following information:

- "Who should use this book"
- "How to use this book"
- "Highlighting conventions" on page xi
- "How to read syntax diagrams" on page xii
- "Related information" on page xiv

Who should use this book

This book is intended for the following people:

- People 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 and related topics, refer to the following Web site:

  http://www.w3c.org/XML

  To learn more about DB2, refer to 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/xmlext/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.

© Copyright IBM Corp. 1999, 2000

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.

What's new in this book for DB2 UDB Version Fixpak 2

This document contains new or rewritten information about:

- Setting up and starting the administration wizard
- How the Update UDF modifies XML documents
- How the XMLClob UDF returns results
- How to increase CLOB limits for stored procedures
- DAD requirement changes:
  - Including stylesheet processing instructions when composing new XML documents
  - Using a missing or empty condition for first RDB node when only one table is specified.
- How to cast parameter markers with JDBC
- Working with code pages:
  - Legal encoding declarations
  - Conversion assumptions
  - Conversion scenarios
- XML Extender parameter limits appendix

Changed information is marked with a vertical change bar, "|".

| See the readme file to learn about all the defect fixes for this fixpak.

Check the Support page of the XML Extender web site for our FAQ and known problems list for additional information about fixes and solutions to common questions: http://www.ibm.com/software/data/db2/extenders/xmlext/support.html

Including this book in the DB2 UDB Version 7 Information Center

The XML Extender documentation can be included in the DB2 information Center using the following steps:

- Copy the HTML files for this book from the `DOC` subdirectory for your language of the XML product install to the subdirectory under the DB2 UDB documentation directory for your language:

  For Windows, the Information Center directory is `sqllib\doc\html\db2sx`

  For UNIX, the Information Center directory is *`install directory`*`/doc/html/db2sx`
- Restart the Information Center and this book will be included on the **Books** tab.

Highlighting conventions

This books 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.

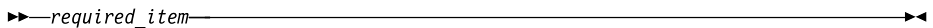
  The ►►—— symbol indicates the beginning of a statement.

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

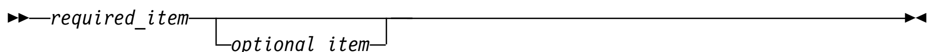
  The ►—— symbol indicates that a statement is continued from the previous line.

  The ——►◄ symbol indicates the end of a statement.

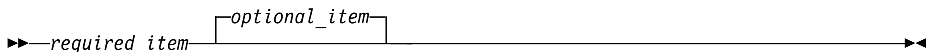
  Diagrams of syntactical units other than complete statements start with the ►—— symbol and end with the ——► symbol.
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

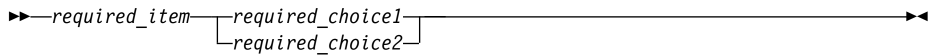


  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

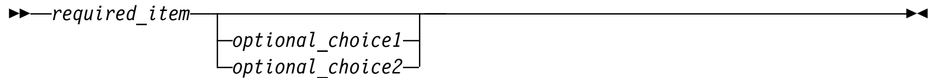


- If you can choose from two or more items, they appear vertically, in a stack.

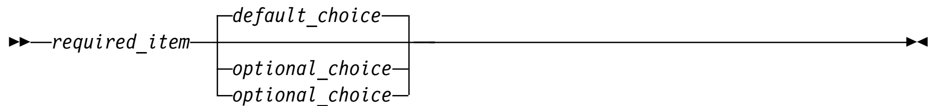
  If you *must* choose one of the items, one item of the stack appears on the main path.



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

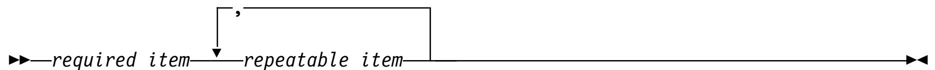


If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- 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 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
Call Level Interface Guide and Reference	SC09–2950	This book describes how to write applications using CLI to access DB2 servers.
DB2 Application Development Guide	SC09–2949	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.
DB2 Extender page	N/A	This page contains information about the DB2 Extenders as well as technologies that are pertinent to the extenders. The Web address of the DB2 Extenders page is: http://www.software.ibm.com/data/db2/extenders
DB2 SQL Reference for Universal Database Parts 1 and 2	• Part 1: SC09–2974 • Part 2: SC09–2975	These books describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views.
• *DB2 Universal Database Administration Guide: Implementation* • *DB2 Universal Database Administration Guide: Performance* • *DB2 Universal Database Administration Guide: Planning*	• Implementation: SC09–2944 • Performance: SC09–2945 • Planning: SC09–2946	These books describe how to design, implement, and maintain a DB2 database.

Document	Order Number	Description
DB2 Universal Database Image, Audio, and Video Extenders Administration and Programming	SC26–9929	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.
DB2 Universal Database Text Extender Administration and Programming	SC26–9930	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.

Part 1. Introduction

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

© Copyright IBM Corp. 1999, 2000

Chapter 1. Introduction to the XML Extender

The IBM® DB2® Extenders™ family provides data and metadata management solutions to handle traditional and nontraditional data. The XML Extender helps you integrate the power of IBM's DB2 Universal Database (DB2 UDB)™ with the flexibility of XML.

DB2's XML Extender provides the ability to store and access XML documents, or generate XML documents from existing relational date and shred (decompose, storing untagged element or attribute content) XML documents into relational data. XML Extender provides new data types, functions, and stored procedures to manage your XML data in DB2.

The XML Extender is available on the following operating systems:
- Windows NT
- AIX
- Sun Solaris
- Linux
- NUMA-Q

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 for their particular tasks. However, because users tend to share data between their separate applications, they are continually faced with the problem of replicating, transforming, exporting, or saving their data as a different format that can be imported into another application. This can be a critical problem in business applications because many of these transforming processes tend to drop some of the data, or they require at least that users go through the tedious process of ensuring that data is consistent. This 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 to save the data into 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 need to be written to convert the data into a form that an application requires, however applications change quickly and become out of date. Applications that convert data to HTML provide presentation solutions, but the data presented cannot be practically used for other purposes. If there

© Copyright IBM Corp. 1999, 2000

were another method that separated data from 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. XML is not the only interchange format. However, XML has emerged as the accepted standard for data interchange. By adhering to this standard, applications can finally share data without needing to transform data 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. With XML, this can be done with ease. In today's interconnected world, an application vendor will not be able to compete unless it provides XML interchange utilities built into its applications. So, in this example, your project management application could export tasks in XML, which could then be imported as is into your calendar application (if the information conforms to an accepted DTD).

Why XML and DB2?

Even though XML solves many problems by providing a standard format for data interchange, there are still other problems to overcome. When building an enterprise data application, you need to answer questions such as:

- How often do I want to replicate the data?
- What kind of information needs to 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 into the database, you can more directly (and more quickly) obtain the XML results that your other applications need for their particular purpose. 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 your traditional relational data.

Based on the application, you can choose whether to store entire XML documents in DB2 as a nontraditional user-defined data type, or you can map the XML content as traditional data in relational tables. For nontraditional 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 UDB Text Extender provides.

With the XML Extender, your application can:

- Store entire XML documents as *column data* in an application table or externally as a local file, while extracting desired XML element or attribute values into *side tables* for search. Using the XML column method, you can:
  - Perform fast search on XML elements or attributes of SQL general *data types* that have been extracted into side tables 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
- Compose or decompose contents of XML documents with one or more relational tables, using the XML collection storage and access method

Integrating XML into DB2

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 usage methods for your XML data.
- A DTD repository for you to store DTDs used to validate XML data: DTD_REF
- A mapping scheme called the Document Access Definition (DAD) file for you map XML documents to relational data

Administration tools

The XML Extender administration tools help you enabling your database and table columns for XML, and map XML data to DB2 relational structures. The XML Extender provides several administration tools for your use, depending on whether you want to develop an application to perform your administration tasks or whether you simply want to use a wizard. You can use the following tools to complete administration tasks for the XML Extender:

- The XML Extender administration wizards provide a graphical user interface for administration tasks.

- The **dxxadm** command provides a command line option for administration tasks.
- The XML Extender administration stored procedures provide application development options for administration tasks.

Storage and access methods

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

XML column
: This method helps you stored intact XML documents in DB2. XML column 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 in turn can be indexed for fast structural search.

XML collection
: This method helps you to map XML document structures to DB2 tables so that you can either compose XML documents from existing DB2 data, or decompose (store untagged element or attribute content) XML documents into DB2 data. This method is good for data interchange applications, particularly when the contents of XML documents are frequently updated.

DTD repository

The XML Extender provides an XML *Document Type Definition (DTD) repository*, which is a set of declarations for XML elements and attributes. When a database is *enabled* for XML, 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 to insert their own DTDs. The *DTDs* in the DTD_REF table are used to validate XML documents.

Document Access Definitions (DADs)

You specify how structured XML documents are to be handled in a *document access definition (DAD)*. The DAD itself is an XML formatted document. It associates XML document structure to a DB2 database when using either XML columns or XML collections. The structure of the DAD is different when defining an XML column, as opposed to an XML collection.

DAD files are managed using the XML_USAGE table, created when you enable a database for XML.

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 1.

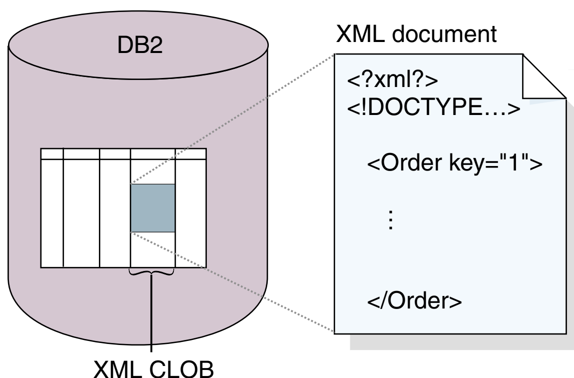


Figure 1. Storing structured XML documents in a DB2 table column

The XML Extender provides following the user-defined types (UDTs) for use with XML columns:

- XMLVarchar
- XMLCLOB
- XMLFILE

All the XML Extender's UDTs have the prefix db2xml, which is the *schema name* of the DB2 XML Extender UDTs. These data types are used to identify the storage type of XML documents in the application table. The XML Extender supports legacy flat files; you are not required to store XML documents inside DB2. You can also store XML documents as files on the *local file system*, as specified by a local file name.

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 queries. 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 elements and attributes as base data types
- Update: Updates entire XML documents or specified element and attribute values

The extract functions allow you to perform powerful searches on general SQL data types. Additionally, you can use the DB2 UDB Text Extender 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 prefix db2xml, which is the schema name of the DB2 XML Extender UDFs. The UDFs are applied to XML UDTs, and they are used for XML columns.

Location path

A *location path* is a sequence of *XML tags* that identify an XML element or attribute. The XML Extender uses the location path to identify 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 following situations:

- To identify the elements and attributes to be extracted, when extracting UDFs
- To specify the mapping file between an XML element or attribute and a DB2 column when defining the indexing scheme in the DAD for XML columns
- To identify an XML element or attribute when using the Text Extender for structural-text search

Figure 2 on page 9 shows an example of a location path and its relationship to the structure of the XML document.

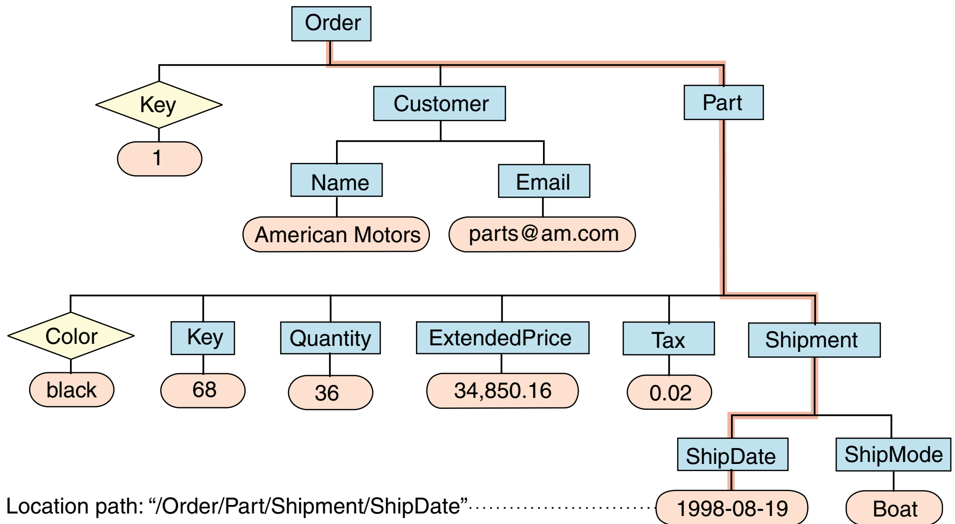


Figure 2. Storing documents as structured XML documents in a DB2 table column

To specify the location path, the XML Extender uses a subset of the *XML Stylesheet Language Transformation (XSLT)* and *XML Path Language (XPath)*. This book uses the term *location path*, which is defined in the specification of XPath. The location path is a sequence of XML tags that identify an XML element or attribute. This book also uses the XSLT or XPath abbreviated syntax of the *absolute location path*, which is specified in the XPath specifications. The absolute location path is the full path name of an object.

XSLT is a language for transforming XML documents into other XML documents. It is designed for use as part of *XML Stylesheet Language (XSL)*, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.

XPath is a language for addressing parts of an XML document, which is designed to be used by XSLT. Every location path can be expressed using the syntax defined for XPath.

For more information about XSLT and XPath, see the following Web pages:
- For XSLT, see: http://www.w3.org/TR/WD-xslt
- For XPath, see: http://www.w3.org/TR/xpath

See "Location path" on page 50 for syntax and restrictions.

XML column terminology

This section describes XML concepts and terminology that are referred to in this book.

document access definition (DAD)
: For XML column, a mapping of XML document structure to DB2 side tables that are indexed for structural queries.

DXX_INSTALL
: The XML Extender installation directory.

side table
: Additional tables that the XML Extender creates to improve performance when searching elements or attributes in an XML column.

XML column
: A method of storing and accessing XML documents by enabling a DB2 column for XML data types and storing an intact XML document in the enabled column. Also refers to a column that has been enabled for XML, using one of the administration tools.

XML table
: An application table that includes one or more columns that has been enabled for XML, using one of the administration tools.

XML UDF
: A DB2 user-defined function provided by the XML Extender.

XML UDT
: A DB2 user-defined type provided by the XML Extender.

XML collection: Integrated data management

Traditional SQL data is either *decomposed* from incoming XML documents or used to *compose* outgoing XML documents. 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 11.

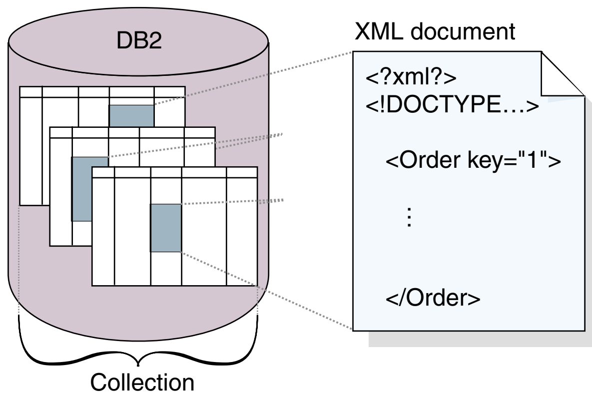


Figure 3. Storing documents as untagged data in DB2 tables

The XML collection is defined in a DAD file, which specifies how elements and attributes are mapped to one or more relational tables. You can define a collection name by enabling it, and then use it with stored procedures to compose or decompose 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. SQL mapping uses SQL SELECT statements to define the DB2 tables and conditions uses for the collection. RDB_node mapping uses XPath-based RDB_node to define the tables, columns, and conditions.

Stored procedures are provided to compose or decompose XML documents. The stored procedures use the prefix db2xml, which is the *schema name* of the XML Extender. Use the following stored procedures with XML collections:

- Composition:
  - dxxGenXML(): uses a DAD file for an XML collection to compose XML documents
  - dxxRetrieveXML(): uses an enabled XML collection to compose XML documents
- Decomposition:
  - dxxShredXML(): uses a DAD file for an XML collection to decompose XML documents
  - dxxInsertXML(): uses an enabled XML collection to decompose XML documents

XML collection terminology

The following terms are unique to the XML Extender and are used frequently in this book.

composition
Generating XML documents from existing relational data as defined by a DAD file.

decomposition
Storing XML documents as untagged, relational data as defined by a DAD file.

document access definition (DAD)
For XML collection, a mapping of XML document structures to DB2 data structures for composing or decomposing XML documents.

DXX_INSTALL
The XML Extender installation directory.

XML collection
A method of storing and accessing XML data using a set of relational tables. Untagged data can be composed into XML documents, or it can be decomposed from XML documents. Also refers to the set of tables into which or from which XML documents are composed or decomposed.

XML stored procedures
Stored procedures to compose or decompose XML documents.

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 DB2 Command Window with 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 DB2 UDB Version 6.1 or higher. If Version 6.1 is used, you must install Fixpak 2. Additionally, the steps in these lessons assume you are using Windows NT®.

The lessons are as follows:
- Store an intact XML document in a DB2 table column
  - Plan the XML UDT in which to store the document and the XML elements and attributes to be frequently searched.
  - Set up the database and tables
  - Enable the database for XML
  - Insert the DTD into the DTD repository table
  - Prepare a DAD for an XML column
  - Add a column into an existing table of XML type
  - 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
  - Enable the database for XML
  - Prepare a document access definition (DAD) file for an XML collection
  - Compose the XML document from existing data

© Copyright IBM Corp. 1999, 2000

- 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 to extract key information from it to be stored in XML documents.

Lesson: Store an XML document in an XML column

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 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 data types 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 261.

The SALES_TAB is described in Table 1.

Table 1. SALES_TAB table

Column name	Data type
INVOICE_NUM	CHAR(6) NOT NULL PRIMARY KEY

Table 1. SALES_TAB table (continued)

Column name	Data type
SALES_PERSON	VARCHAR(20)
ORDER	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 they can be 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 Figure 4 on page 16.

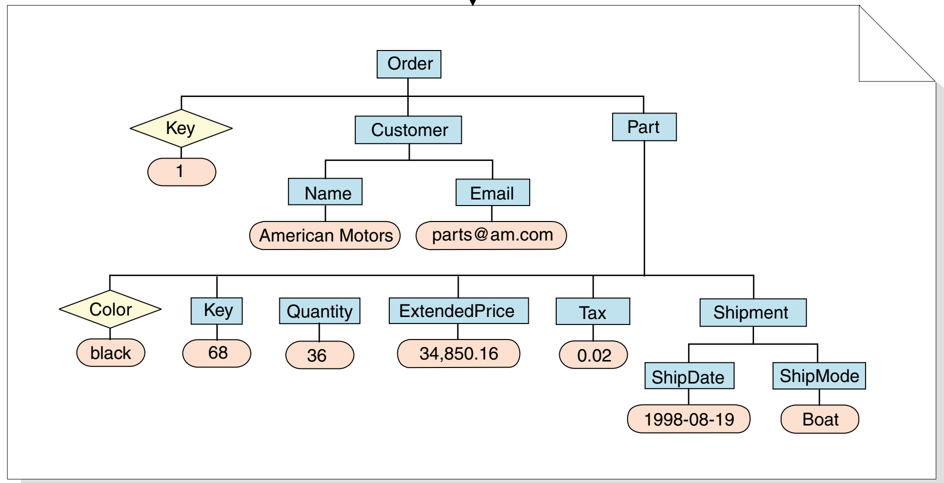
This lesson also provides a sample DTD for you to use in understanding and validating the XML document structure. You can see the DTD file in “Appendix B. Samples” on page 261. It matches the structure in Figure 4 on page 16.

DTD

```
<?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)>
```

Raw data

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM
"d:\dxx\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>
      ⋮
  </Part>
</Order>
```



=Attribute =Element =Value

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

Determining the XML data type for the XML column

The XML Extender provides XML user data 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, the elements and attributes that will be searched or extracted most frequently, or 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

In this tutorial, you will be creating 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 be in their own tables.

Additionally, you also need to 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, the elements and attributes are mapped to the following side tables:

ORDER_SIDE_TAB

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

PART_SIDE_TAB

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

SHIP_SIDE_TAB

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

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 drive and directory where you installed the XML Extender, for example `c:\dxx\samples\cmd`), and they are as follows:

getstart_db.cmd
: Creates the database and populates four tables.

getstart_prep.cmd
: Binds the database with the XML Extender stored procedures and the DB2 CLI.

getstart_insertDTD.cmd
: Inserts the DTD used to validate the XML document in the XML column.

getstart_createTabCol.cmd
: Creates an application table that will have an XML-enabled column.

getstart_alterTabCol.cmd
: Alters the application table by adding the column that will be enabled for XML.

getstart_enableCol.cmd
: Enables the XML column.

getstart_createIndex.cmd
: Creates indexes on the side tables for the XML column.

getstart_insertXML.cmd
Inserts the XML document into the XML column.

getstart_queryCol.cmd
Runs a select statement on the application table and returns the XML document.

getstart_clean.cmd
Cleans up the tutorial environment.

Setting up

In this section, you prepare the database for use with the XML Extender. You will:

1. Create the database.
2. Enable the database.

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.

To create the database:

1. Change to the `*DXX_INSTALL*\samples\cmd` directory, where *DXX_INSTALL* is the drive and directory where you installed the XML Extender. The `c:\dxx` directory is assumed in these lessons. Change these values if your drive and directory are different.
2. Open the DB2 Command Window from the Windows NT Start menu, or enter the following command from the Windows NT command prompt:
   ```
   DB2CMD
   ```
3. From the DB2 Command Window, run the following command:
   ```
   getstart_db.cmd
   ```

Enabling the database

To store XML information in the database, you need to enable it for the XML Extender. When you enable a database for XML, the XML Extender:

- Creates all the user-defined types (UDTs) and user-defined functions (UDFs).
- Creates and populates control tables with the necessary metadata that the XML Extender requires.
- Creates the db2xml schema and assigns the necessary privileges.

To enable the database for XML:

From the DB2 Command Window, run the following script to enable the SALES_DB database:

```
getstart_prep.cmd
```

This script binds the database to the XML Extender stored procedures and the DB2 CLI. It also runs the **dxxadm** command option that enables the database:

```
dxxadm enable_db SALES_DB
```

Creating the 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 XMLFILE type, 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. For the purposes of this tutorial, you will store the provided XML document in the XML column.

In this lesson, you will store the document in the SALES_TAB table. To store the document, 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 decide to validate XML documents, you must store the DTD in this repository. The DTD for this tutorial is `c:\dxx\samples\dtd\getstart.dtd`.

To insert the DTD:

From the DB2 Command Window, enter the following SQL INSERT command, all on the same line:

```
DB2 CONNECT TO SALES_DB
DB2 INSERT into db2xml.dtd_ref values('c:\dxx\samples\dtd\getstart.dtd',
    db2xml.XMLClobFromFile('c:\dxx\samples\dtd\getstart.dtd'), 0, 'user1',
    'user1', 'user1')
```

You can also run the following command file to insert the DTD:

```
getstart_insertDTD.cmd
```

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 `c:\dxx\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 than those 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 "c:\dxx\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 to specify the DTD ID identifier that associates the DAD with the XML document DTD and specifies the DTD location at the client.
   ```
   <dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
   ```
   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 20. For example, the path you used for the DTDID might be different that the above string if you are working on a different machine drive.
5. Specify opening and closing <validation></validation> tags to indicate that 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. The method defines that the XML data is to be stored in an 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 opening and closing <column></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 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 50 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, `c:\dxx\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 file contains path statements that might need to be changed to match your environment in order to be run successfully.

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:

From the DB2 Command Window, enter the following CREATE TABLE statement:

```
DB2 CONNECT TO SALES_DB

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

Alternatively, you can run the following command file to create the table:

```
getstart_createTabCol.cmd
```

Adding the column of XML type

Now, add a new column into 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 8. XML Extender user-defined types” on page 163. In this tutorial, you will store the document as XMLVARCHAR.

To add the column of XML type:

From the DB2 Command Window, enter the following SQL statement:

```
DB2 ALTER TABLE SALES_TAB ADD ORDER DB2XML.XMLVARCHAR
```

Alternatively, you can run the following command file to alter the table:

```
getstart_alterTabCol.cmd
```

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, and the side table columns. You can specify the default view when querying the XML document. In this lesson, you will specify the 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 ties 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 tablespace.

To enable the column for XML:

From the DB2 Command Window, enter the following command:

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

Alternatively, you can run the following command file to enable the column for XML:

```
getstart_enableCol.cmd
```

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. You should only update the XML document using the UDFs supplied by the XML Extender. 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:

From the DB2 Command Window, enter the following SQL SELECT statement:

```
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 step, 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 3 describes these columns, which you will index:

Table 3. 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:

Enter the following SQL commands from the DB2 Command Window:

```
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)
```

Alternatively, you can run the following command file to create the indexes:

```
getstart_createIndex.cmd
```

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 into 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 into a column of the XML UDT XMLVARCHAR, you will use the default casting function. See “Storing data” on page 114 for more information about the storage default casting functions and the XML Extender-provided UDFs.

To store the XML document:

Important: Open the XML document c:\dxx\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.

From the DB2 Command Window, enter the following SQL INSERT command:

```
DB2 INSERT INTO SALES_TAB (INVOICE_NUM, SALES_PERSON, ORDER) VALUES('123456',
   'Sriram Srinivasan', db2xml.XMLVarcharFromFile('c:\dxx\samples\cmd\getstart.xml'))
```

When you store the XML document, the XML Extender automatically updates the side tables.

Alternatively, you can run the following command file to store the document:

```
getstart_insertXML.cmd
```

To verify that the tables have been updated, run the following SELECT statements for the tables from the DB2 Command Window:

```
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:

Enter the following SELECT statement from the DB2 Command Window:

```
DB2 "SELECT DISTINCT SALES_PERSON FROM SALES_TAB S, PART_SIDE_TAB P
   WHERE PRICE > 2500.00 AND
   S.INVOICE_NUM=P.INVOICE_NUM"
```

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

Alternatively, you can run the following command file to search the document:

```
getstart_queryCol.cmd
```

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

The tutorial scenario

You have been given the task of taking information in an existing purchase order database, SALES_DB, and extracting key 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
ORDER_KEY	INTEGER
CUSTOMER	VARCHAR(16)
CUSTOMER_NAME	VARCHAR(16)
CUSTOMER_EMAIL	VARCHAR(16)

PART_TAB

Column name	Data type
PART_KEY	INTEGER
COLOR	CHAR(6)
QUANTITY	INTEGER
PRICE	DECIMAL(10,2)
TAX	REAL

Column name	Data type
ORDER_KEY	INTEGER

SHIP_TAB

Column name	Data type
DATE	DATE
MODE	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 5 on page 29.

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 261. You can see that it matches the structure in Figure 5 on page 29.

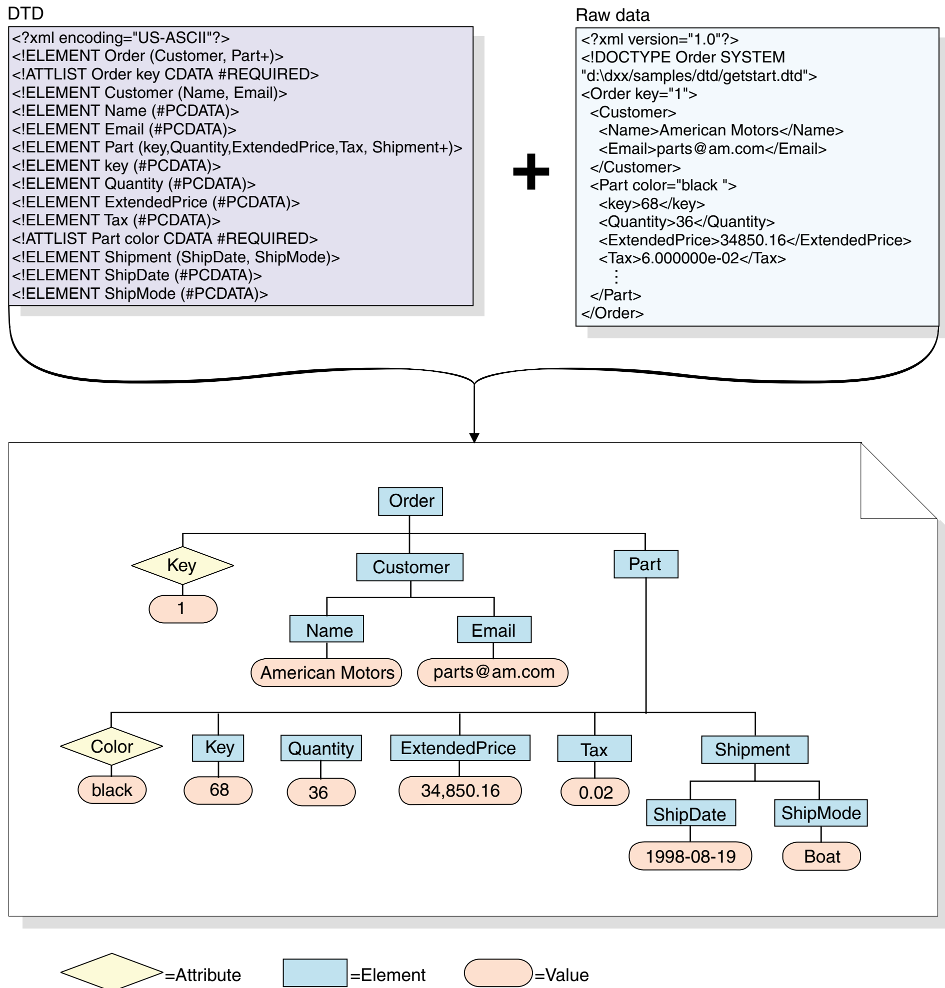


Figure 5. The hierarchical structure of the DTD and XML document

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 6 on page 30.

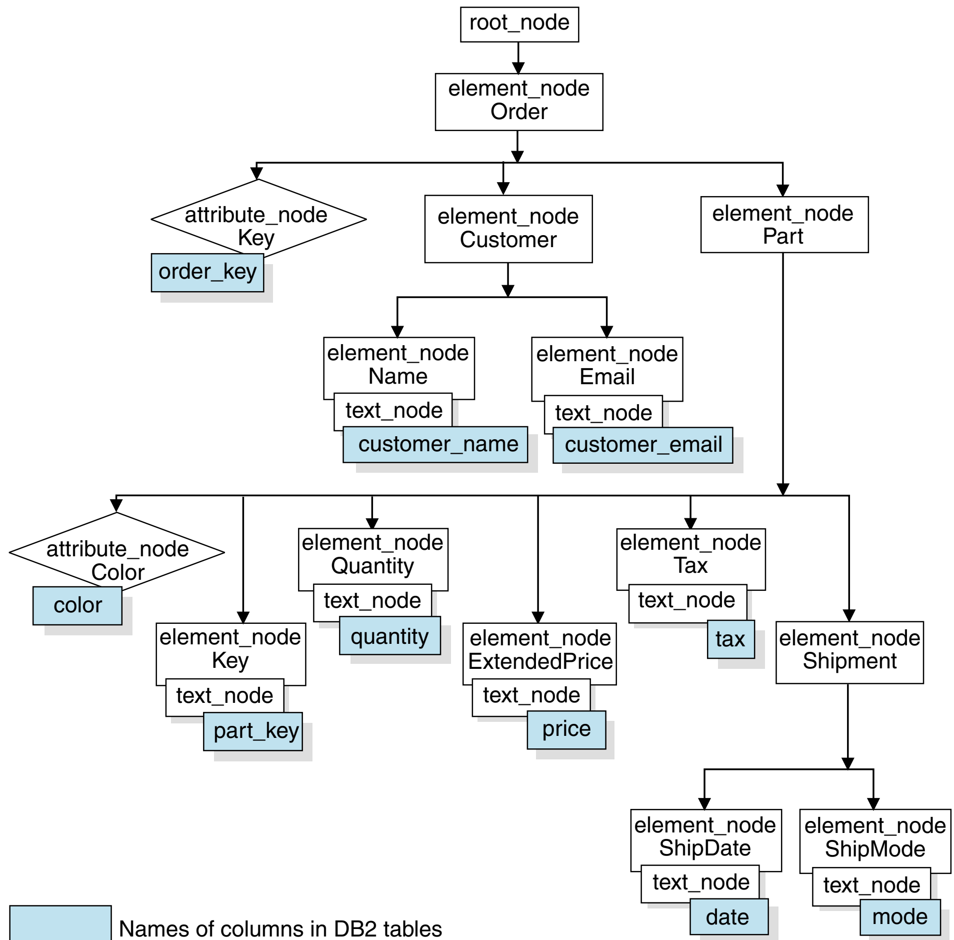


Figure 6. XML document mapped to relational table columns

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 6, 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.

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 drive and directory where you installed the XML Extender, for example `c:\dxx\samples\cmd`), and they are as follows:

getstart_db.cmd
: Creates the database and populates four tables.

getstart_prep.cmd
: Binds the database with the XML Extender stored procedures and the DB2 CLI.

getstart_stp.cmd
: Runs the stored procedure to compose the XML collection.

getstart_exportXML.cmd
: Exports the XML document from the database for use in an application.

getstart_clean.cmd
: Cleans up the tutorial environment.

Setting up

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. Change to the `*DXX_INSTALL*\samples\cmd` directory, where *DXX_INSTALL* is the drive and directory where you installed the XML Extender. The `c:\dxx` directory is assumed in these lessons. Change these values if your drive and directory are different.
2. Open the DB2 Command Window from the Windows NT Start menu, or enter the following command from the Windows NT command prompt:
   ```
   DB2CMD
   ```
3. From the DB2 Command Window, run the following command:
   ```
   getstart_db.cmd
   ```

Enabling the database

To store XML information in the database, you need to enable it for the XML Extender. When you enable a database for XML, the XML Extender:

- Creates all the user-defined types (UDTs) and user-defined functions (UDFs).
- Creates and populates control tables with the necessary metadata that the XML Extender requires.
- Creates the db2xml schema and assigns the necessary privileges.

Important: If you have completed the XML column lesson and have not cleaned up your environment, you might be able skip this step.

To enable the database for XML:

From the DB2 Command Window, run the following script to enable the SALES_DB database:

```
getstart_prep.cmd
```

This script binds the database to the XML Extender stored procedures and the DB2 CLI. It also runs the **dxxadm** command option that enables the database:

```
dxxadm enable_db SALES_DB
```

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 28 you determined which columns are in the relational database where the data exists, and how the data from the tables will be structured into 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 `c:\dxx\samples\dad\getstart_xcollection.dad`. It has some minor differences to the file generate in the following steps. If you use it for the lesson, note that the file paths might be different than in your environment.

To create the DAD file for composing an XML document:

1. From the `c:\dxx\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 "c:\dxx\dtd\dad.dtd">
   ```

   The XML Extender assumes that you installed the product in `c:\dxx`. If this is not correct, change this value to the drive and directory that you specified during the installation of this product here and in the following steps.
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 using the DTD you inserted into the DTD repository table.

```
<validation>NO</validation>
```

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.

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

6. Specify 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 58 to learn more about mapping schemes.) Enter the following statement:

```
<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 substr(char(timestamp(generate_unique())),16)
     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>
```

This SQL statement uses the following guidelines when using SQL mapping. Refer to Figure 6 on page 30 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 an entity are grouped together, and 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 DB2 built-in 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 59 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 "c:\dxx\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 54 for more information about these nodes. Figure 6 on page 30 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 names"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> <!-- 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 `c:\dxx\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 path statements 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 can 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 105.

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. From the DB2 Command Window, enter the following command to run the stored procedure:

   ```
   getstart_stp.cmd
   ```

   The XML document has been composed and is stored in the RESULT_TAB table.

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

   - `c:\dxx\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`.
   - `c:\dxx\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 the XML Extender retrieval function, Content():

```
DB2 CONNECT TO SALES_DB

DB2 SELECT db2xml.Content(db2xml.xmlvarchar(doc),
   'c:\dxx\samples\cmd\getstart.xml') FROM RESULT_TAB
```

Alternatively, you can run the following command to export the file:

```
getstart_exportXML.cmd
```

This lesson teaches you how to get one or more composed XML documents using DB2 stored procedure's result set feature to allow you to fetch each row to get each document. As you get each row of document, you can export it to a file, which is the simplest way to demonstrate this feature. For more efficient ways of fetching data see the CLI examples in `c:\dxx\samples\cli`.

Cleaning up the tutorial environment

If you want to clean up the tutorial environment, you can run the `getstart_clean.cmd` file. This file:

- Disables the XML column, ORDER
- Drops tables created in the tutorial
- Deletes the DTD from the DTD reference table

This command file does 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:

1. From the DB2 Command Window, run the following command:
   ```
   getstart_clean.cmd
   ```
2. If you want to disable the database, you can run the following XML Extender command from the DB2 Command Window:
   ```
   dxxadm disable_db 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.
3. If you want to drop the database, you can run the following command from the DB2 Command Window:
   ```
   db2 drop database SALES_DB
   ```
   This command drops the SALES_DB.

Part 2. Administration

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

© Copyright IBM Corp. 1999, 2000

Chapter 3. Preparing to use the XML Extender: administration

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

Set-up requirements

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

Software requirements

The XML Extender is available on AIX, Windows NT, and Sun Solaris.

Required software: The XML Extender requires DB2 Universal Database Version 7.1 or higher.

Optional software:

- For structural text search, the DB2 Universal Database Text Extender Version 7.1 or higher
- For the XML Extender administration tool:
  - DB2 UDB JDBC (available with DB2 UDB Version 7.1 or higher)
  - JDK 1.1.7 or JRE 1.1.7 (available with the DB2 UDB Control Center)
  - JFC 1.1 with Swing 1.1 (available with the DB2 UDB Control Center)

Installation requirements

See the README file for your operating system for the following tasks:

- Binding the XML Extender to your DB2 UDB database.

  For security reasons, you must bind the XML Extender to each database. For details on how to complete the bind, see "Before you begin" on page 203, or for an example see

  `DXX_INSTALL\samples\cmd\getstart_prep.cmd`
- Viewing the set up instructions on UNIX.
- Creating a database for XML access.

Authorization requirements

You need DB2ADM authority to perform administration tasks.

© Copyright IBM Corp. 1999, 2000

Administration tools

The XML Extender provides three methods for administration: the XML Extender administration wizard, the XML Extender administration command, and the XML Extender *stored procedures*.

- The administration wizard prompts you through the administration tasks and is the recommended method for administration. Use of this tool is described in the administration tasks in "Chapter 4. Administering XML data" on page 65.
- The administration command, **dxxadm**, provides options for the various administration tasks. Use of this command is described in the administration tasks in "Chapter 4. Administering XML data" on page 65 and in "Chapter 7. XML Extender administration command: **dxxadm**" on page 151.
- The administration stored procedures also provide options for various administration tasks. These stored procedures are described in "Administration stored procedures" on page 203.

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 pre-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 you 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.
- Your XML documents exceed 2 gigabytes and you must decompose them.

You use the document access definition (DAD) file to associate XML data with DB2 tables through these two access and storage methods. Figure 7 shows how the DAD specifies the access and storage methods.

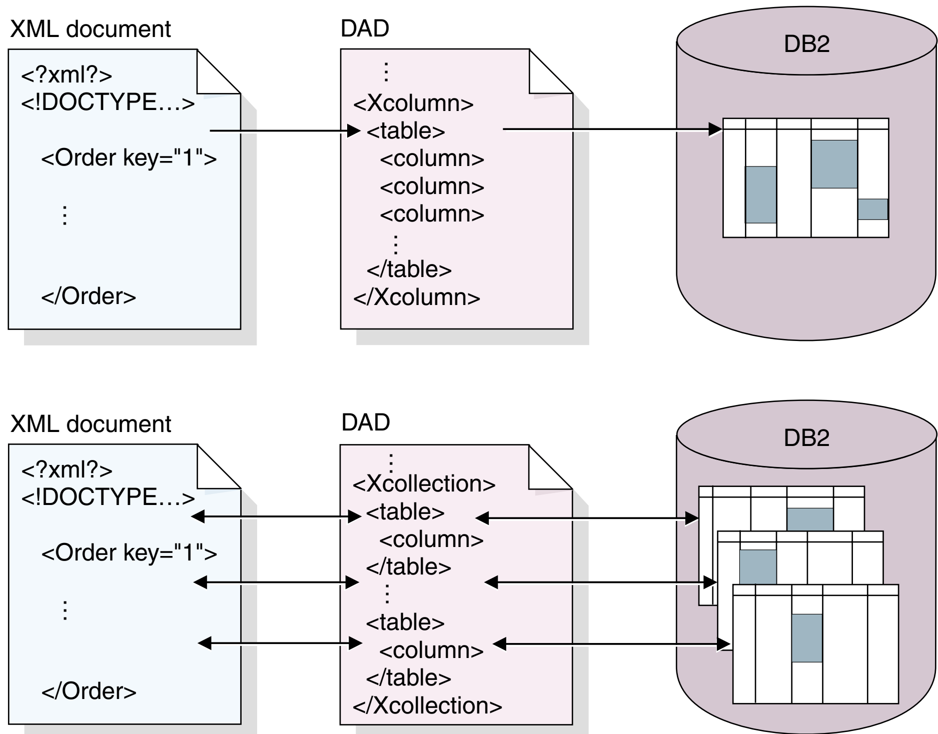


Figure 7. The DAD file maps the XML document structure to DB2 and specifies the access and storage method.

The DAD file is an important part of administrating 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

The following sections describe the planning tasks for XML columns.

Validation

After you choose an access and storage method, you can determine whether to *validate* your data. You validate XML data using a DTD to ensures that the XML document is valid and to perform structured searches on your XML data. The DTD is stored in the DTD repository, or can be stored in the file system that the DB2 server has access to.

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:

- It is recommended that you 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" on page 70 to learn how to insert a DTD into the repository.
- 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.
- 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 entities and attribute defaults even when processing document fragments that cannot be validated.

XML user-defined types

You store an XML document in an XML column as a UDT. See Table 4 on page 48 for the available UDTs.

Table 4. The XML Extender UDTs

User-defined type column	Source data type	Usage description
XMLVARCHAR	VARCHAR(*varchar_len*)	Stores an entire XML document as VARCHAR inside DB2.
XMLCLOB	CLOB(*clob_len*)	Stores an entire XML document as *CLOB* inside 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.

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.

Multiple occurrence: When a document has multiple occuring 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. Figure 8 shows an XML column with side tables.

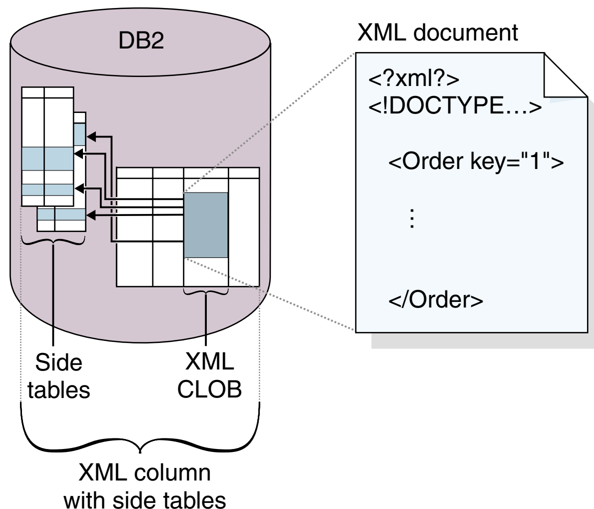


Figure 8. An XML column with side tables

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.

  For example, you might want to create an index on `/Order/Part/ExtendedPrice` and specify `/Order/Part/ExtendedPrice` to be of data type REAL. In this case, the XML Extender stores the value of `/Order/Part/ExtendedPrice` in the PRICE column in a side table.
- You can create multiple indexes on an XML column. Using the previous example, you can create two columns in two side tables, one for `ExtendedPrice` and one for `ShipDate`.
- 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.

Location path

A *location path* is a sequence of *XML tags* that identify an XML element or attribute. The XML Extender uses the location path in the following situations:

- When extracting UDFs to identify the elements and attributes to be extracted

- To specify the mapping file between an XML element or attribute and a DB2 column when defining the indexing scheme in the DAD for XML columns
- By the Text Extender for structural-text search

Figure 9 shows an example of a location path and its relationship to the structure of the XML document.

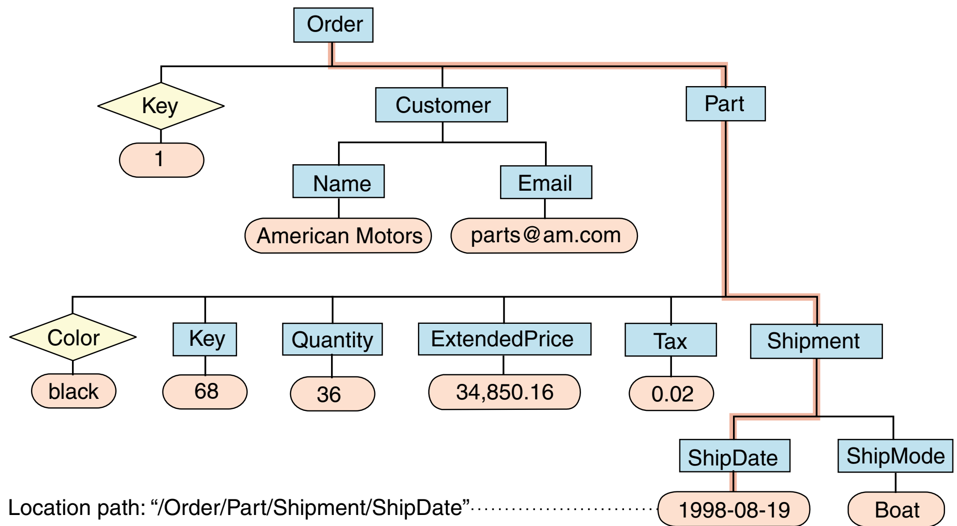


Figure 9. Storing documents as structured XML documents in a DB2 table column

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*.

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.

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 5 summarizes the syntax for simple location path.

Table 5. Simple location path syntax

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

XML Extender restrictions: The XML Extender has restrictions for using the location path when defining the element or attribute in the DAD. 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 6 describes the restrictions for location path. The numbers that are specified in the location path supported column refer to the syntax representations in "Location path syntax" on page 51.

Table 6. 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 5)
Extracting UDFs	1-8[1]

Table 6. The XML Extender's restrictions using location path (continued)

Use of the location path	Location path supported
Update UDF	1-8[1]
Text Extender's search UDF	1-8

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

The DAD file

For XML columns, the DAD primarily specifies how documents that are stored in an XML column are to be indexed. 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. 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. To learn how to insert a DTD into the repository, see “Storing a DTD in the DTD repository” on page 70. The DTD requirements differ depending on whether you are composing or decomposing XML documents.

- 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 10 on page 56 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 "c:\dtd\dad.dtd">
<DAD>
  ...
<Xcollection>
<SQL_stmt>
     ...
</SQL_stmt>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE DAD SYSTEM "c:\dxx\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 10. 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 style sheets, 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 11 shows how the hierarchical structure can be mapped to relational table columns.

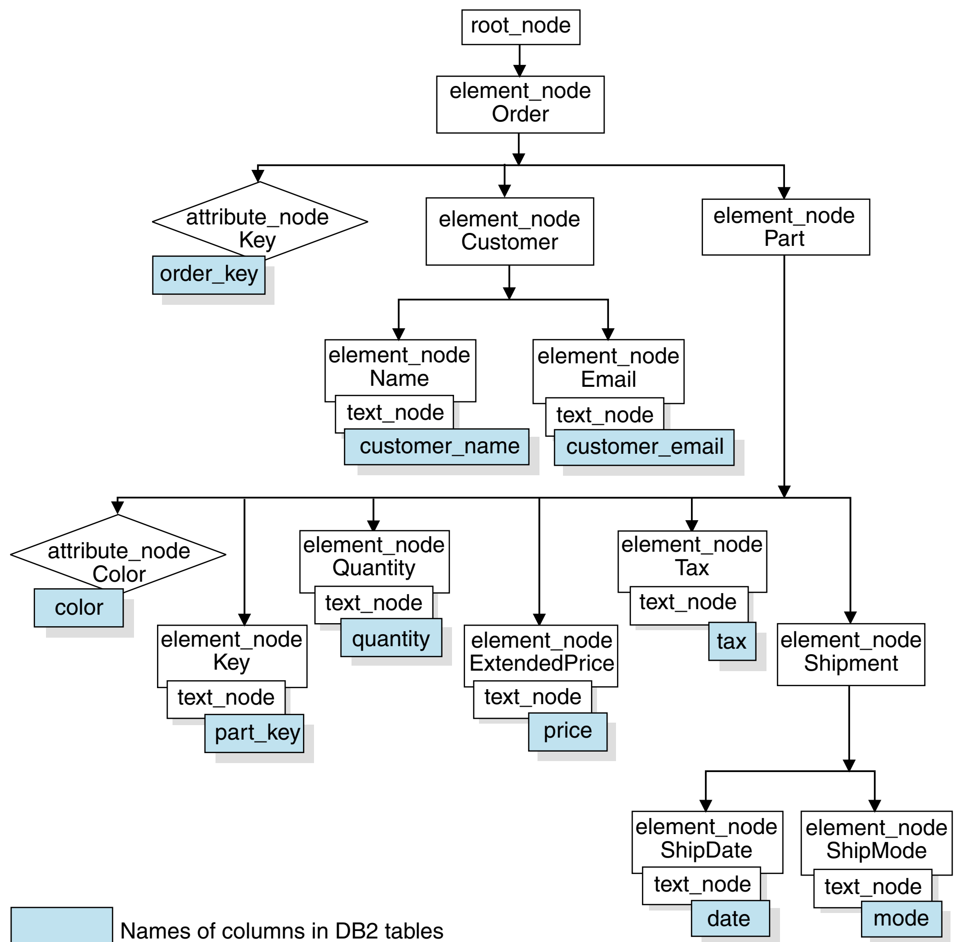


Figure 11. XML document structured mapped to relational table columns

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 XSLT model to define the hierarchy of the XML document.

SQL mapping
: Allows simple and direct mapping from relational data to XML documents through a single SQL statement and the *XSLT 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.

  The *RDB_node* 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 12 on page 59 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 "c:\dtd\dad.dtd">
<DAD>
  <dtdid>c:\dxx\samples\dad\getstart.dtd</dtdid>
  <validation>YES</validation>
  <Xcollection>
    <SQL_stmt>
       SELECT o.order_key, customer, p.part_key, quantity, price, tax, date,
              mode, comment
         FROM order_tab o, part_tab p,
               table(select substr(char(timestamp(generate_unique())),
               as ship_id, date, mode, from ship_tab) as 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 "c:\dxx\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 12. 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 proceed 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 built-in 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 built-in 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 substr(char(timestamp(generate_unique())),16) as
            ship_id, date, mode from ship_tab) as s
    ```

    In this example, the generate_unique() function is cast to a CHAR data type of TIMESTAMP, and it is 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 built-in 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. For example, using a *union*, expression, or operation is somewhat less powerful than the SQL-to-XML approach.

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-occuring 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 occuring elements, consider this table size restriction.

Chapter 4. Administering XML data

The XML Extender administration tasks consist of enabling your database and table columns for XML and mapping XML data to DB2 relational structures. The XML Extender provides several administration tools for your use, depending on whether you want to develop an application to perform your administration tasks or whether you simply want to use a wizard. You can use the following tools to complete administration tasks for the XML Extender:

- The XML Extender administration wizard
- The **dxxadm** command
- The XML Extender administration stored procedures

This chapter describes the administration tasks that are associated with the administration wizard and the **dxxadm** command. The administration stored procedures are described in "Administration stored procedures" on page 203.

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 44.

The following sections describe the XML Extender administration tasks:

1. "Starting the administration wizard"
2. "Enabling a database for XML" on page 69
3. "Storing a DTD in the DTD repository" on page 70
4. "Defining XML columns or collections" on page 71
5. "Work with XML columns" on page 72
6. "Work with XML collections" on page 83

Starting the administration wizard

This section contains information about setting up and invoking 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 (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:
```

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 userid. 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.

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 AIX, Sun Solaris, and Linux:

   Run the `dxxadmin` file.

   The administration wizard Logon window opens.

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

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:

   For stand-alone configurations (recommended):

   ```
   jdbc:db2:database_name
   ```

   Where:

   database_name
   The database to which you are connecting and storing XML documents.

   For example,

   ```
   jdbc:db2:sales_db
   ```

   For network configurations:

   ```
   jdbc:db2://server_name:port_number/database_name
   ```

   Where:

   server_name
   Is the name of the server where the XML Extender is located.

port_number
The port number used to connect to the server. To determine the port number, enter the following command from the DB2 command line at the server machine:

```
db2jstrt port#
```

Windows NT users can check the following file \winnt\system32\driver\etc\services for the port number.

database_name
The database to which you are connecting and storing XML documents.

For example,

```
jdbc:db2://host1.ibm.com:8080/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:

   For stand-alone configurations (default and recommended):

   ```
   COM.ibm.db2.jdbc.app.DB2DRIVER
   ```

   For network configurations:

   ```
   COM.ibm.db2.jdbc.net.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 a database
- Add a DTD to the DTD repository
- Work with DAD files for:
  - XML columns
  - XML collections
- Work with XML columns
- Work with XML collections

Enabling a database 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, using the current instance.

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

- Creates all the user-defined types (UDTs) and user-defined functions (UDFs)
- 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 database** from the LaunchPad window to enable the current database.

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

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

From the DB2 command shell

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

Syntax:

dxxadm enable_db

```
►►—dxxadm—enable_db—dbName—————————————————————————————————►◄
```

Parameters:

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

Example: Enables an existing database, called SALES_DB.

```
dxxadm enable_db SALES_DB
```

Storing a DTD in the DTD repository

You can use a DTD to validate XML data in an XML column or in an XML collection. The DTD validates the XML column and is used to define DAD files that are used for XML structural search and for collection composition and decomposition.

All DTDs are stored in the DTD repository, 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 46 and “Planning for XML collections” on page 53 to learn more about using DTDs.

You can insert the DTD from the DB2 command shell 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 is displayed.
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 DB2 command shell

Issue an SQL INSERT statement for the DTD_REF table using the schema in Table 7:

Table 7. The schema for the DTD_REF DTD 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('c:\dxx\samples\dtd\getstart.dtd',
    db2xml.XMLClobFromFile('c:\dxx\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.

Defining XML columns or collections

The following sections describe how to set up and define your database for XML columns or collections, and how to prepare the necessary data mapping schemes.

These sections are:

- "Work with XML columns" on page 72
- "Work with XML collections" on page 83

Work with XML columns

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 is used to map the element and attribute values that you want to index to DB2 tables. See "Location path" on page 50 to learn more about location path and it's syntax.

Creating or editing the DAD file

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 53 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 into 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 into 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 you have not imported any DTDs into the DTD repository for your database, you cannot validate your XML documents.
   - 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 50 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 which contains the column.

     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 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.

From the DB2 command shell

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 a 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 263:

   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 50 for location path syntax.
      - ***multi_occurrence***: an indication of whether this element or attribute can be used more than once in a document

      ```
      <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 with a column of XML type using the administration wizard, or using the DB2 command shell.

Using the adminstration 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_DB
   ```
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.

From the DB2 command shell

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 DB2 command shell. 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, if defined.
- 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 UDT.

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 into the **DAD file name** field, or click **...** to browse for an existing DAD file. For example:

`c:\dxx\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.

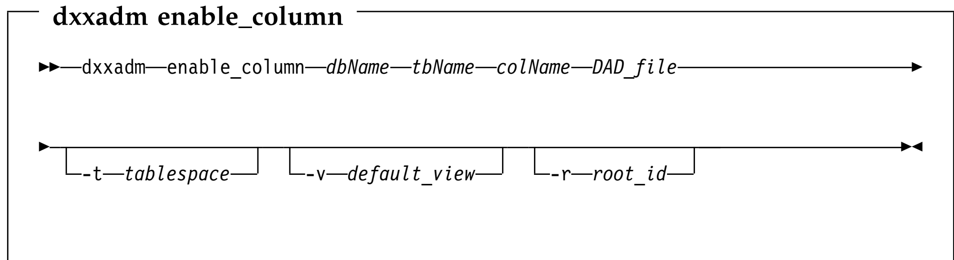
   When specified, the default view is automatically created when the column is enabled and joins the XML table and all of the related side tables.
9. Optionally, type the column name of the primary key in the application table in the **Root ID** field. This is recommended.

   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.

From the DB2 command shell

To enable an XML column, enter the following command:

Syntax:



Parameters:

dbName
: The name of the database.

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. The column name of the primary key in the application table and a unique identifier that associates all side tables with the application table. The XML Extender uses the value of *root_id* as a unique identifier to associate all side tables with the application table. Specifying the ROOT ID is recommended. 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 DB2 command shell. The DAD file and XML document can be found in “Appendix B. Samples” on page 261.

```
dxxadm enable_column SALES_DB sales_tab order getstart.dad
           -v sales_order_view -r invoice_num
```

In this example, the column ORDER is enabled in the table SALES_DB.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 schema:

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.

The DB2 CREATE INDEX command

Use the DB2 CREATE INDEX command.

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 DB2 command shell.

When the XML Extender enables 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 the 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.

From the DB2 command shell

To disable an XML column, enter the following command:

Syntax:

dxxadm disable_column

```
►►—dxxadm—disable_column—dbName—tbName—colName——————————————————►◄
```

Parameters:

dbName
: The name of the database.

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 DB2 command shell. The DAD file and XML document can be found in "Appendix B. Samples" on page 261.

```
dxxadm disable_column SALES_DB sales_tab order
```

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

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

Work with XML collections

Setting up XML collections requires creating a mapping scheme and optionally enabling the collection with a virtual name that associates the DB2 tables with a DAD file.

Although enabling the XML collection is not required, it does provide better performance.

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 be used for composing XML documents, only.

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.

Both methods of mapping use the *XPath data model*, which is described in "The DAD file" on page 54.

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 is displayed.
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 into 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 you have not imported any DTDs into the DTD repository for your database, you cannot validate your XML documents.

- 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 DAD SYSTEM "c:\dxx\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 substr(char(timestamp(generate_unique())),16)
  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 create 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.

From the DB2 command shell: 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 262. 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 59 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 substr(char(timestamp(generate_unique())),16)
     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 "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

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 253, 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 1 is to be 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 is displayed.
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 into 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 into 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 you have not imported any DTDs into the DTD repository for your database, you cannot validate your XML documents.
   - 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 create 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:

```
operator 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.

From the DB2 command shell: 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 262. 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 "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

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 node 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

```
expression = expression AND
expression = expression
```

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 262 for RDB_node mapping, as well as the DTD for the DAD file in “Appendix A. DTD for the DAD file” on page 253, 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 style sheets, 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: To create a DAD for decomposition:

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 is displayed.
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 into 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 into 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 you have not imported any DTDs into the DTD repository for your database, you cannot validate your XML documents.

- 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:

operator 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.

From the DB2 command shell: 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 262. 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 "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

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

```
expression = expression AND
expression = expression ...
```

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.

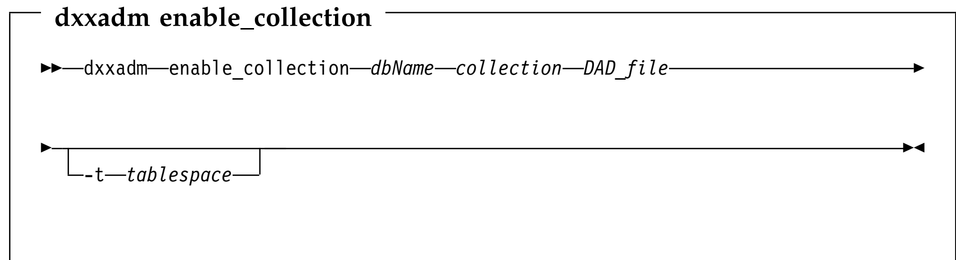
1. Set up and start the administration wizard. See "Starting the administration wizard" on page 65 for details.
2. Click **Work with XML Collections** from the LaunchPad window. The Select a Task window is displayed.
3. Click **Enable a Collection** and then **Next**. The Enable a Collection window is displayed.
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 into 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.

From the DB2 command shell

To enable an XML collection, enter the **dxxadm** command:

Syntax:



Parameters:

dbName
: The name of the database.

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 DB2 command shell. The DAD file uses SQL mapping and can be found in “DAD file: XML collection - SQL mapping” on page 263.

```
dxxadm enable_collection SALES_DB sales_ord getstart.dad
```

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 is displayed.
3. Click **Disable an XML Collection** and then **Next** to disable an XML collection. The Disable a Collection window is displayed.

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.

From the DB2 command shell

To disable an XML collection, enter the **dxxadm** command:

Syntax:

dxxadm disable_collection

```
►►—dxxadm—disable_collection—dbName—collection———————————————————►◄
```

Parameters:

dbName
: The name of the database.

collection
: The name of the XML collection. This value is used as a parameter for the XML collection stored procedures.

Example:

```
dxxadm disable_collection SALES_DB sales_ord
```

Disabling a database for XML

You disable the database 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 database to which you are connected, using the current instance.

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

- Deletes all the user-defined types (UDTs) and user-defined functions (UDFs)
- Deletes control tables with the metadata for the XML Extender
- Deletes the db2xml schema.

Before you begin

Disable any XML columns or collections in the database to be disabled.

Using the administration wizard

Use the following steps to disable 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 **Disable database** from the LaunchPad window to disable the current database.

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

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

From the DB2 command shell

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

Syntax:

dxxadm disable_db

```
>>--dxxadm--disable_db--dbName----------------------------------------><
```

Parameters:

dbName
    The name of the database that is to be disabled.

Example: disables an existing database, called SALES_DB.

```
dxxadm disable_db SALES_DB
```

Part 3. Programming

This part describes programming techniques for managing your XML data.

© Copyright IBM Corp. 1999, 2000

Chapter 5. 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 convert the SQL base type to the XML UDT
- XML Extender-provided user-defined functions (UDFs)

This book describes both methods for each task.

© Copyright IBM Corp. 1999, 2000

UDT and UDF 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.

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 prefix 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 8 on page 115 shows when to use each method.

Table 8. 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 9 shows the provided casting functions:

Table 9. 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 casted 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 10 shows the storage functions provided by the XML Extender.

Table 10. 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.

Table 10. The XML Extender storage UDFs (continued)

Storage user-defined function	Return type	Description
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('c:\dxx\samples\cmd\getstart.xml'))
```

The preceding example imports the XML object from the file named `c:\dxx\samples\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 117
- "Retrieving element contents and attribute values" on page 119

The XML Extender provides two methods for retrieving data: default casting functions and the Content() overloaded UDF. Table 11 on page 117 shows when to use each method.

Table 11. 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 is to 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 12 shows the provided casting functions:

Table 12. 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* 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 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* illustrates how an XML document is retrieved from a file to memory. This example assumes that the column BOOK 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 50 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 13 on page 120 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 50 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 13 shows the extracting functions, both in scalar and table format:

Table 13. The XML Extender extracting functions

Scalar function	Table function	Returned column name (table function)	Return type
extractInteger()	extractIntegers()	returnedInteger	INTEGER
extractSmallint()	extractSmallints()	returnedSmallint	SMALLINT

Table 13. The XML Extender extracting functions (continued)

Scalar function	Table function	Returned column name (table function)	Return type
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 returned when the attribute value of key = ″1″. The value is extracted as an integer 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;
```

Table function example:

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. However, 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 9 on page 115 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 casted 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('c:\dxx\samples\cmd\getstart.xml')
     WHERE sales_person = 'Sriram Srinivasan'
```

The preceding example updates the XML object from the file named `c:\dxx\samples\cmd\getstart.xml` to the column ORDER in the table SALES_TAB.

See Table 10 on page 115 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 update one value at a time, in a document, 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 50 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 196 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 that is to be updated.

path Is the location path of the element or attribute that is to be updated. See “Location path” on page 50 for location path syntax. See “Multiple occurrence” on page 198 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 occuring 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”
- “Using the Text Extender for structural text search” on page 126

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 and attribute values 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 Text Extender Administration and Programming*.

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/@color = /Order/Part/@color
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 attributes 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(varcharhandle,
 '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 Text Extender Administration and Programming*.

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 JDBC

When using parameter markers in functions, a JDBC restriction requires that the parameter marker for the function must be casted 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 6. 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 279.

You can increase the CLOB sizes for the stored procedures results as documented in "Increasing the CLOB limit" on page 202.

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 139

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 53 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 53 for complete details.

© Copyright IBM Corp. 1999, 2000

- 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 one column of VARCHAR, CLOB, XMLVARCHAR, or XMLCLOB type.

| Additionally, if you specify the validation element in the DAD file as YES, the XML Extender adds the column DXX_VALID of INTEGER type to the result table, and inserts a value of 1 for a valid XML document and 0 for an invalid document.

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(resultTabName) resultTabName,   /* 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 212 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 */
     SQL TYPE is CLOB_FILE  dadFile;          /* dad file */
     char                   result_tab[32];   /* name of the result table */
     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                  ovtype_ind;
     short                  ov_inde;
     short                  maxrow_ind;
     short                  numrow_ind;
     short                  returnCode_ind;
     short                  returnMsg_ind;

     EXEC SQL END DECLARE SECTION;

     /* create table */
     EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

     /* read data from a file to a CLOB */
     strcpy(dadfile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
     dadfile.name_length = strlen("c:\dxx\samples\dad\getstart_xcollection.dad");
     dadfile.file_options = SQL_FILE_READ;
     EXEC SQL VALUES (:dadfile) INTO :dad;
     strcpy(result_tab,"xml_order_tab");
     override[0] = '\0';
     overrideType = NO_OVERRIDE;
     max_row = 500;
     num_row = 0;
     returnCode = 0;
     msg_txt[0] = '\0';
     collection_ind = 0;
     dad_ind = 0;
     rtab_ind = 0;
     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 dxxGenXML(:dad:dad_ind;
          :result_tab:rtab_ind,
```

```
:overrideType:ovtype_ind,:override:ov_ind,
:max_row:maxrow_ind,:num_row:numrow_ind,
:returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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(collectionName) collectionName,  /* input */
               char(resultTabName) resultTabName,   /* 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 216 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    collection[32];   /* dad buffer */
char    result_tab[32];   /* name of the result table */
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   dadbuf_ind;
short   rtab_ind;
short   ovtype_ind;
short   ov_inde;
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);

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(result_tab,"xml_order_tab");
override[0] = '\0';
overrideType = NO_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collection_ind = 0;
rtab_ind = 0;
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 dxxRetrieve(:collection:collection_ind;
        :result_tab:rtab_ind,
        :overrideType:ovtype_ind,:override:ov_ind,
        :max_row:maxrow_ind,:num_row:numrow_ind,
        :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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.

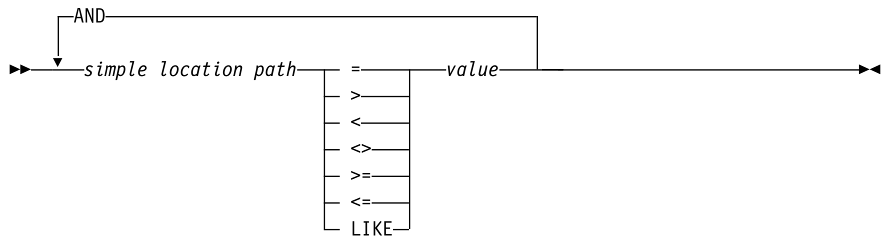
XML_OVERRIDE

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 5 on page 52 for syntax.

operators
: 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: A stored procedure using SQL_OVERRIDE.

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];    /* dad buffer */
         char    result_tab[32];    /* name of the result table */
         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   rtab_ind;
         short   ovtype_ind;
         short   ov_inde;
         short   maxrow_ind;
         short   numrow_ind;
         short   returnCode_ind;
         short   returnMsg_ind;

         EXEC SQL END DECLARE SECTION;

         /* create table */
         EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);
```

```
/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(result_tab,"xml_order_tab");
sprintf(override,"%s %s %s %s %s %s %s",
          "SELECT o.order_key, customer, p.part_key, quantity, price,",
          "tax, ship_id, date, mode ",
          "FROM order_tab o, part_tab p,",
          "table(select substr(char(timestamp(generate_unique())),16",
          "as ship_id,date,mode from ship_tab)as s",
          "WHERE p.price > 50.00 and s.date >'1998-12-01' AND",
          "p.order_key = o.order_key and s.part_key = p.part_key");
overrideType = SQL_OVERRIDE;
max_row = 500;
num_row = 0;
returnCode = 0;
msg_txt[0] = '\0';
collection_ind = 0;
rtab_ind = 0;
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 dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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: A stored procedure using XML_OVERRIDE.

```
include "dxx.h"
include "dxxrc.h"

       EXEC SQL INCLUDE SQLCA;
       EXEC SQL BEGIN DECLARE SECTION;
         char    collection[32];  /* dad buffer */
         char    result_tab[32];  /* name of the result table */
         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   dadbuf_ind;
```

```
short   rtab_ind;
short   ovtype_ind;
short   ov_inde;
short   maxrow_ind;
short   numrow_ind;
short   returnCode_ind;
short   returnMsg_ind;

EXEC SQL END DECLARE SECTION;

/* create table */
EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

/* initialize host variable and indicators */
strcpy(collection,"sales_ord");
strcpy(result_tab,"xml_order_tab");
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';
collection_ind = 0;
rtab_ind = 0;
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 dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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 into 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 53 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 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-occuring 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 occuring elements, consider this table size restriction.

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 53 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 required 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 221 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      dad;             /* DAD*/
         SQL TYPE is CLOB_FILE dadFile;         /* DAD file*/
         SQL TYPE is CLOB      xmlDoc;          /* input XML document */
         SQL TYPE is CLOB_FILE xmlFile;         /* input XMLfile */
         long                  returnCode;      /* 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;

/* initialize host variable and indicators */
strcpy(dadFile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
dadFile.name_length=strlen("c:\dxx\samples\dad\getstart_xcollection.dad")
dadFile.file_option=SQL_FILE_READ;
strcpy(xmlFile.name,"c:\dxx\samples\cmd\getstart_xcollection.xml");
xmlFile.name_length=strlen("c:\dxx\samples\cmd\getstart_xcollection.xml")
xmlFile.file_option=SQL_FILE_READ;
SQL EXEC VALUES (:dadFile) INTO :dad;
SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
returnCode = 0;
returnMsg[0] = '\0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
             :xmlDoc:xmlDoc_ind,
             :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,          /* input */
             long           returnCode,      /* output */
             varchar(1024)  returnMsg)       /* output */
```

See "dxxInsertXML()" on page 223 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                   collection[64];   /* name of an XML collection */
         SQL TYPE is CLOB_FILE xmlFile;          /* input XML file */
         SQL TYPE is CLOB      xmlDoc;           /* input XML doc */
         long                  returnCode;       /* error code */
         char                  returnMsg[1024];  /* error message text */
         short                 collection_ind;
         short                 xmlDoc_ind;
         short                 returnCode_ind;
         short                 returnMsg_ind;
         EXEC SQL END DECLARE SECTION;

         /* initialize host variable and indicators */
         strcpy(collection,"sales_ord")
         strcpy(xmlobj.name,"c:\dxx\samples\cmd\getstart_xcollection.xml");
         xmlobj.name_length=strlen("c:\dxx\samples\cmd\getstart_xcollection.xml")
         xmlobj.file_option=SQL_FILE_READ;
         SQL EXEC VALUES (:xmlFile) INTO (:xmlDoc);
         returnCode = 0;
         returnMsg[0] = '\0';
         collection_ind = 0;
         xmlobj_ind = 0;
         returnCode_ind = -1;
         returnMsg_ind = -1;

         /* Call the store procedure */
         EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
                      :xmlDoc:xmlDoc_ind,
                      :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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 into 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 216 for syntax and examples.

DAD file consideration: When you decompose XML documents into 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 substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode from ship_tab)as 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 administration command, user-defined data types (UDTs), user-defined functions (UDFs), and stored procedures. Message text is also provided for problem determination activities.

© Copyright IBM Corp. 1999, 2000

Chapter 7. XML Extender administration command: dxxadm

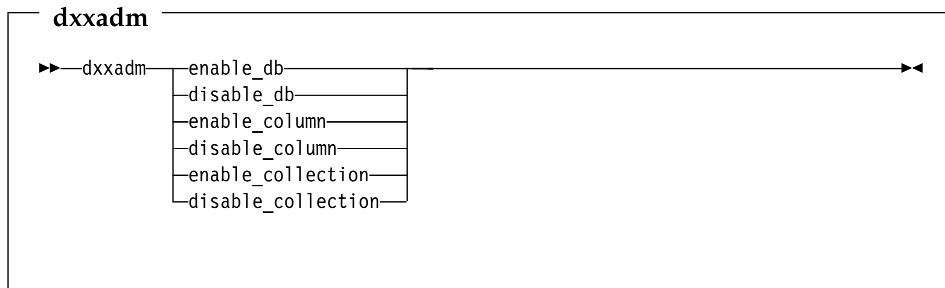
The XML Extender provides an administration command, **dxxadm**, for completing the following administration tasks:You preform the following XML Extender administration tasks by calling **dxxadm** using various command options:

- Enabling or disabling a database 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

The following syntax diagram provides the high-level syntax of the **dxxadm** command. Descriptions of each option are provided in the following sections.



Administration command options

The following sections describe each of the **dxxadm** command options that are available to system programmers:

- "enable_db" on page 152
- "disable_db" on page 153
- "enable_column" on page 155
- "disable_column" on page 157
- "enable_collection" on page 159
- "disable_collection" on page 161

© Copyright IBM Corp. 1999, 2000

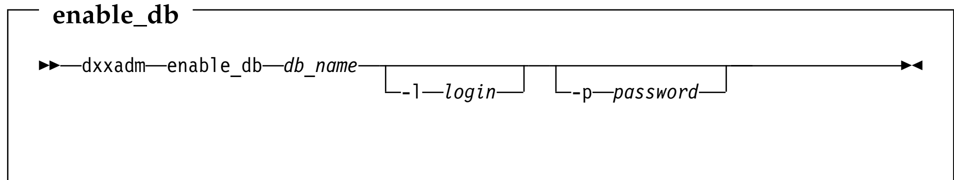
enable_db

Purpose

Connects to and enables a database so that it can be used with the XML Extender. When the database 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 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 225.
- 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 226.

Format



Parameters

Table 14. enable_db parameters

Parameter	Description
db_name	The name of the database in which the XML data resides.
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example enables the database SALES_DB.

```
dxxadm enable_db SALES_DB
```

disable_db

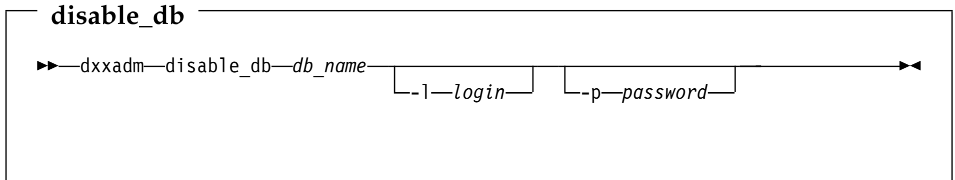
Purpose

Connects to and disables the XML-enabled database. When the database is disabled, it can no longer be used by the XML Extender. When the XML Extender disables the database, 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 225.
- 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 226.

Important: You must disable all XML columns before attempting to disable a database. The XML Extender cannot disable a database that contains columns or collections that are enabled for XML.

Format



Parameters

Table 15. disable_db parameters

Parameter	Description
db_name	The name of the database in which the XML data resides
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example disables the database SALES_DB.

```
dxxadm disable_db SALES_DB
```

enable_column

Purpose

Connects to a database 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

enable_column

```
>>--dxxadm--enable_column--db_name--tab_name--column_name--DAD_file---------->

>--+----------------+--+-----------------+--+------------+------------------>
   '--t--tablespace-'  '--v--default_view-'  '--r--root_id-'

>--+------------+--+---------------+--------------------------------------><
   '--l--login-'  '--p--password-'
```

Parameters

Table 16. enable_column parameters

Parameter	Description
db_name	The name of the database in which the XML data resides.
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.

Table 16. enable_column parameters (continued)

Parameter	Description
-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.
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example enables an XML column.

```
dxxadm enable_column SALES_DB SALES_TAB ORDER -v sales_order_view -r INVOICE_NUMBER
```

disable_column

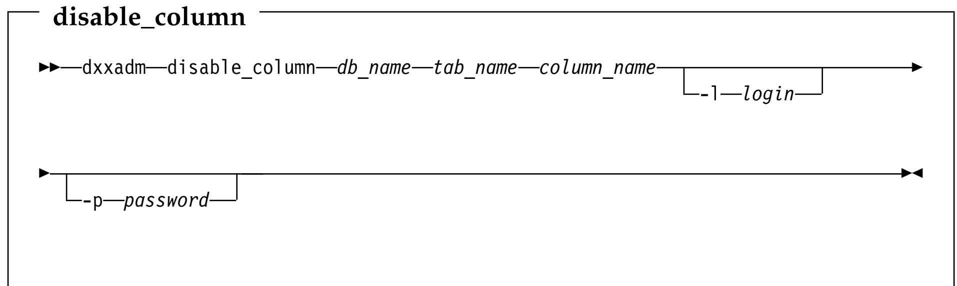
Purpose

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 table 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



Parameters

Table 17. disable_column parameters

Parameter	Description
db_name	The name of the database in which the data resides.
tab_name	The name of the table in which the XML column resides.
column_name	The name of the XML column.
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.

Table 17. disable_column parameters (continued)

Parameter	Description
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example disables an XML-enabled column.

```
dxxadm disable_column SALES_DB SALES_TAB ORDER
```

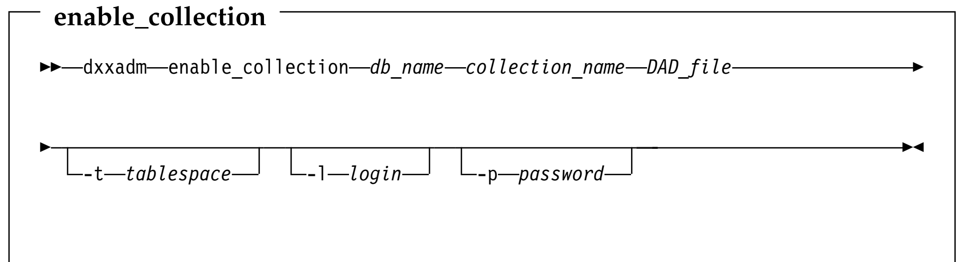
enable_collection

Purpose

Connects to a database 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



Parameters

Table 18. enable_collection parameters

Parameter	Description
db_name	The name of the database in which the data resides.
collection_name	The name of the XML collection.
DAD_file	The name of the DAD file that maps the XML document to the relational tables in the collection.
-t *tablespace*	The name of the table space, which is optional and associated with the collection. If not specified, the default table space is used.
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example enables an XML collection.

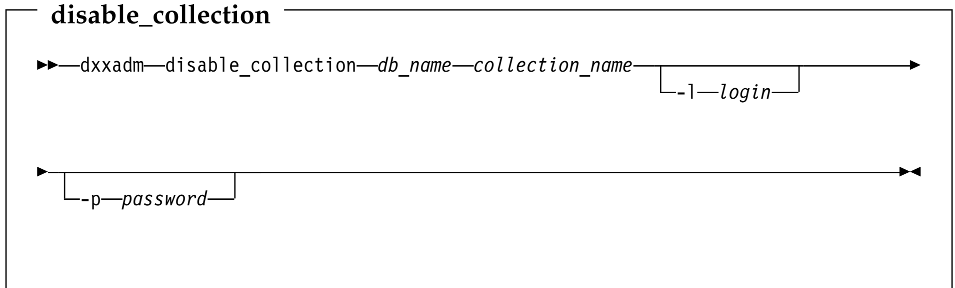
```
dxxadm enable_collection SALES_DB sales_ord getstart_xcollection.dad -t orderspace
```

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



Parameters

Table 19. disable_collection parameters

Parameter	Description
db_name	The name of the database in which the data resides.
collection_name	The name of the XML collection.
-l *login*	The user ID, which is optional, used to connect to the database, when specified. If not specified, the current user ID is used.
-p *password*	The password, which is optional, used to connect to the database, when specified. If not specified, the current password is used.

Examples

The following example disables an XML collection.

```
dxxadm disable_collection SALES_DB sales_ord
```

Chapter 8. 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 20 contains an overview of the UDTs.

Table 20. The XML Extender UDTs

User-defined type column	Source data type	Usage description
XMLVARCHAR	VARCHAR(*varchar_len*)	Stores an entire XML document as VARCHAR inside DB2.
XMLCLOB	CLOB(*clob_len*)	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.

© Copyright IBM Corp. 1999, 2000

Chapter 9. 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 166.

retrieval functions
: Retrieval functions retrieve XML documents from XML columns in a DB2 database. For syntax and examples, see "Retrieval functions" on page 171.

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 178.

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 195.

Table 21 provides a summary of the XML Extender functions.

Table 21. The XML Extender user-defined functions

Type	Function
Storage functions	XMLVarcharFromFile()
	XMLCLOBFromFile()
	XMLFileFromVarchar()
	XMLFileFromCLOB()

© Copyright IBM Corp. 1999, 2000

Table 21. The XML Extender user-defined functions (continued)

Type	Function
Retrieval functions	Content(): retrieve from XMLFile to a CLOB
	Content(): retrieve from XMLVarchar to an external server file
	Content(): retrieve from XMLCLOB to an external server file
Extracting functions	extractInteger() and extractIntegers()
	extractSmallint() and extractSmallints()
	extractDouble() and extractDoubles()
	extractReal() and extractReals()
	extractChar() and extractChars()
	extractVarchar() and extractVarchars()
	extractCLOB() and extractCLOBs()
	extractDate() and extractDates()
	extractTime() and extractTimes()
	extractTimestamp() and extractTimestamps()
Update function	Update()

When using parameter markers in UDFs, a 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 JDBC" on page 129 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 114. Additionally, the XML Extender provides UDFs to access XML documents from sources other than the UDT base data type and convert them to the desired UDT.

The XML Extender provides the following storage functions:

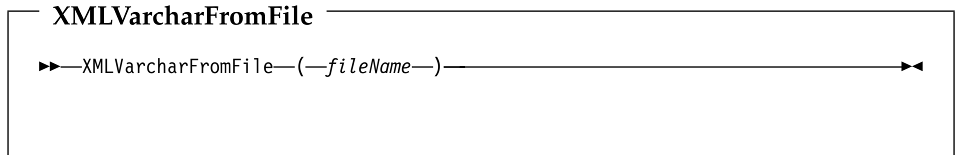
- "XMLVarcharFromFile()" on page 167
- "XMLCLOBFromFile()" on page 168
- "XMLFileFromVarchar()" on page 169
- "XMLFileFromCLOB()" on page 170

XMLVarcharFromFile()

Purpose

Reads an XML document from a server file and returns the document as an XMLVARCHAR type.

Syntax



Parameters

Table 22. XMLVarcharFromFile parameter

Parameter	Data type	Description
fileName	VARCHAR(512)	The fully qualified server file name.

Return type

XMLVARCHAR

Example

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('c:\dxx\samples\cmd\getstart.xml'))
```

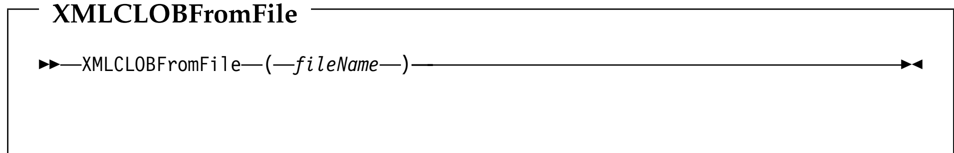
In this example, a record is inserted into the SALES_TAB table. The function XMLVarcharFromFile() imports the XML document from a file into DB2 and stores it as a XMLVARCHAR.

XMLCLOBFromFile()

Purpose

Reads an XML document from a server file and returns the document as an XMLCLOB type.

Syntax



Parameters

Table 23. XMLCLOBFromFile parameter

Parameter	Data type	Description
fileName	VARCHAR(512)	The fully qualified server file name.

Return type

XMLCLOB as LOCATOR

Example

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('c:\dxx\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

```
►►—XMLFileFromVarchar—(—buffer—,—fileName—)——————————————————————►◄
```

Parameters

Table 24. XMLFileFromVarchar parameters

Parameter	Data type	Description
buffer	VARCHAR(3K)	The memory buffer.
fileName	VARCHAR(512)	The fully qualified server file name.

Return type

XMLFILE

Example

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_buff;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO sales_tab(ID, NAME, ORDER)
  VALUES('1234', 'Sriram Srinivasan',
       XMLFileFromVarchar(:xml_buf, 'c:\dxx\samples\cmd\getstart.xml'))
```

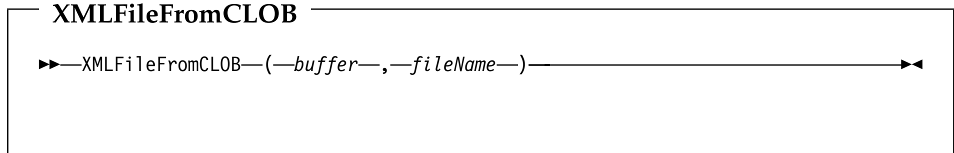
The column ORDER in the SALES_TAB table is defined as an XMLFILE type. The preceding example shows that if you have an XML document in your buffer, you can store it in a server file.

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



Parameters

Table 25. XMLFileFromCLOB() parameters

Parameters	Data type	Description
buffer	CLOB as LOCATOR	The buffer containing the XML document.
fileName	VARCHAR(512)	The fully qualified server file name.

Return type

XMLFILE

Example

The following example reads an XML document as CLOB locator, 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, 'c:\dxx\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

You can use the default casting functions to convert an XML UDT to the base data type as described in "Retrieving an entire document" on page 117. The XML Extender also provides an *overloaded function* Content(), which is used for retrieval.

The Content() function provides 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 172 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 174
  - "Content(): retrieval from XMLCLOB to an external server file" on page 176

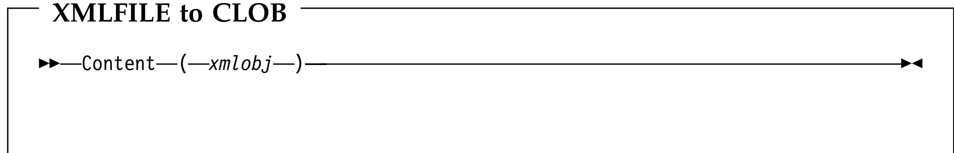
The examples in the following section assume you are using the DB2 command shell, in which you do not need to type "DB2" at the beginning of each command.

Content(): retrieve from XMLFILE to a CLOB

Purpose

Retrieves data from a server file and stores it in a CLOB LOCATOR.

Syntax



Parameters

Table 26. XMLFILE to a CLOB parameter

Parameter	Data type	Description
xmlobj	XMLFILE	The XML document.

Return type

CLOB (*clob_len*) as LOCATOR

clob_len for DB2 UDB is 2G.

Example

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;
```

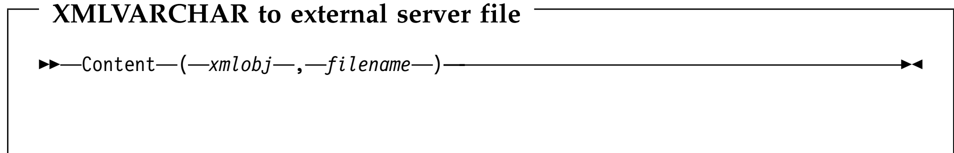
The column ORDER in the SALES_TAB table is of an XMLFILE type, so the Content() UDF retrieves data from a server file and stores it in a CLOB locator.

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



Important: If a file with the specified name already exists, the content function overrides its content.

Note:

Parameters

Table 27. XMLVarchar to external server file parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR	The XML document.
filename	VARCHAR(512)	The fully qualified server file name.

Return type

VARCHAR(512)

Example

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 c:\dxx\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>
```

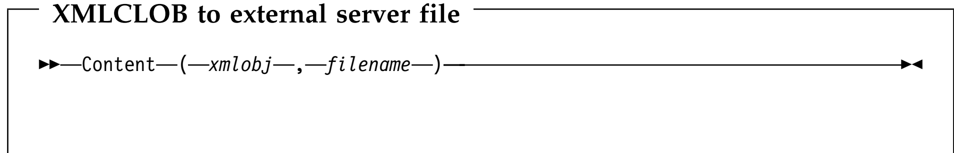
```
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>AIR </ShipMode>
            </Shipment>
            <Shipment>
                <ShipDate>1998-08-19</ShipDate>
                <ShipMode>BOAT </ShipMode>
            </Shipment>
       </Order>');

SELECT db2xml.Content(order, 'c:\dxx\samples\cmd\getstart_.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



Important: If a file with the specified name already exists, the content function overrides its content.

Parameters

Table 28. XMLCLOB to external server file parameters

Parameter	Data type	Description
xmlobj	XMLCLOB as LOCATOR	The XML document.
filename	VARCHAR(512)	The fully qualified server file name.

Return type

VARCHAR(512)

Example

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 c:\dxx\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>
    </Order>');

SELECT db2xml.Content(order, 'c:\dxx\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 latter is called the table function.

The XML Extender provides the following extracting functions:

- "extractInteger() and extractIntegers()" on page 179
- "extractSmallint() and extractSmallints()" on page 181
- "extractDouble() and extractDoubles()" on page 182
- "extractReal() and extractReals()" on page 184
- "extractChar() and extractChars()" on page 185
- "extractVarchar() and extractVarchars()" on page 186
- "extractCLOB() and extractCLOBs()" on page 188
- "extractDate() and extractDates()" on page 190
- "extractTime() and extractTimes()" on page 191
- "extractTimestamp() and extractTimestamps()" on page 193

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 29. extractInteger and extractIntegers function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

INTEGER

Returned column name (table function)

returnedInteger

Example

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 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;
```

Table function example:

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;
```

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 30. extractSmallint and extractSmallints function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

SMALLINT

Returned column name (table function)

returnedSmallint

Example

In the following example, the value of Quantity in all sales orders is extracted as SMALLINT

```
SELECT * from table(db2xml.extractSmallints(db2xml.File
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/Quantity')) 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 31. extractDouble and extractDoubles function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

DOUBLE

Returned column name (table function)

returnedDouble

Example

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(5,2));
INSERT into t1 values (db2xml.extractDouble(db2xml.XMLFile
   ('c:\dxx\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 * from table(db2xml.extractDoubles(db2xml.XMLFile
   ('c:\dxx\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 32. extractReal and extractReals function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

REAL

Returned column name (table function)

returnedReal

Example

In the following example, each value of Tax is extracted as a REAL.

```
SELECT * from table(db2xml.extractReals(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/Tax')) 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 33. extractChar and extractChars function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

CHAR

Returned column name (table function)

returnedChar

Example

In the following example, the value of Color is extracted as CHAR.

```
SELECT * from table(db2xml.extractChars(Order,
   ('c:\dxx\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 34. extractVarchar and extractVarchars function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

VARCHAR(4K)

Returned column name (table function)

returnedVarchar

Example

In a database with more than 1000 XML documents that are stored in the column ORDER in the SALES_TAB table, you might want to find all the customers who have ordered items that have an ExtendedPrice greater than 2500.00. The following SQL statement uses the extracting UDF in the SELECT clause:

```
SELECT extractVarchar(Order, '/Order/Customer/Name') from sales_order_view
   WHERE price > 2500.00
```

The UDF extractVarchar() takes the column ORDER as the input and the location path `/Order/Customer/Name` as the select identifier. The UDF returns the names of the customers. With the WHERE clause, the extracting function evaluates only those orders with an ExtendedPrice greater than 2500.00.

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 35. extractCLOB and extractCLOBs function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

CLOB(10K)

Returned column name (table function)

returnedCLOB

Example

In this example, all of the part elements are extracted from a purchase order.

```
SELECT returnedCLOB as part
   from table(db2xml.extractCLOBs(db2xml.XMLFile('c:\dxx\samples\xml\getstart.xml'),
   '/Order/Part')) 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 36. extractDate and extractDates function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

DATE

Returned column name (table function)

returnedDate

Example

In the following example, the value of ShipDate is extracted as DATE.

```
SELECT * from table(db2xml.extractDates(db2xml.XMLFile
   ('c:\dxx\samples\xml\getstart.xml'), '/Order/Part/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 37. extractTime and extractTimes function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type
TIME

Returned column name (table function)
returnedTime

Example
This example uses the book sample files. It searches the XML file `book.xml` for the time at which books were priced and returns the values as TIME.

XML file:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:\dxx\samples\book.dtd">
<book>
```

```
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

Extract example:

```
CREATE TABLE t1(SaleTime TIME);
INSERT INTO t1 values(db2xml.extractTime(doc, '/book/price/@time'));
SELECT * from t1;
```

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 38. extractTimestamp and extractTimestamps function parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLFILE, or XMLCLOB	The column name.
path	VARCHAR	The location path of the element or attribute.

Return type

TIMESTAMP

Returned column name (table function)

returnedTimestamp

Example

This example uses the book sample files. It searches the XML file `book.xml` for the time specifying when each book was priced and extracts the value as TIMESTAMP.

XML file:

```
<?xml version="1.0">
<DOCTYPE book SYSTEM "c:\dxx\samples\book.dtd">
<book>
<chapter id="1" date="07/01/97">
<section>This is a section in Chapter One.</section>
<chapter id="2" date="01/02/1997">
<section>This is a section in Chapter Two.</section>
</chapter>
<price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13.888888">
38.281
</price>
</book>
```

Extract example:

```
CREATE TABLE t1(SaleTime TIMESTAMP);
INSERT INTO t1 values(db2xml.extractTimestamp(doc, '/book/price/@timestamp'));
SELECT * from t1;
```

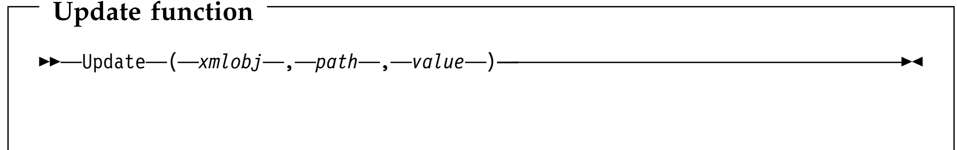
Update function

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 121.

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



Parameters

Table 39. The UDF Update parameters

Parameter	Data type	Description
xmlobj	XMLVARCHAR, XMLCLOB as LOCATOR	The column name.
path	VARCHAR	The location path of the element or attribute.
value	VARCHAR	The update string.

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 sales person Sriram Srinivasan.

```
UPDATE sales_tab
   set order = 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 actually 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 40 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 198

Table 40. Update function rules

Item or Node type	XML document code example	Status after update
XML Declaration	`<?xml  version='1.0' encoding='utf-8' standalone='yes' >`	The XML declaration is preserved: • Version information is preserved. • Encoding declaration is preserved and appears when specified in the original document. • Standalone declaration is preserved and appears when specified in the original document. • After update, single quotes are used to delineate values.

Table 40. Update function rules (continued)

Item or Node type	XML document code example	Status after update
DOCTYPE Declaration	<!DOCTYPE books SYSTEM "http://dtds.org/books.dtd" > <!DOCTYPE books PUBLIC "local.books.dtd" "http://dtds.org/books.dtd" > <!DOCTYPE books> -Any of <!DOCTYPE books (S ExternalID) ? [internal-dtd-subset] > -Such as <!DOCTYPE books [<!ENTITY mydog "Spot">] >? [internal-dtd-subset] >	The document type declaration is preserved: • Root element name is supported. • Public and system ExternalIDs are preserved and appear when specified in the original document. • Internal DTD subset is NOT preserved. Entities are replaced; defaults for attributes are processed and appear in the output documents. • After update, double quotes are used to delineate public and system URI values. • 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.
Processing Instructions	<?xml-stylesheet title="compact" href="datatypes1.xsl" type="text/xsl"?>	Processing instructions are preserved.
Comments	<!-- comment -->	Comments are preserved when inside the root element. Comments outside the root element are discarded.
Elements	<books> *content* </books>	Elements are preserved.

Table 40. Update function rules (continued)

Item or Node type	XML document code example	Status after update
Attributes	`id='1' date='01/02/1997"`	Attributes of elements are preserved. • After update, double quotes are used to delineate values. • Data within attributes is escaped. • Entities are replaced.
Text Nodes	`This chapter is about my dog &mydoc;.` `This chapter is about my dog Spot.`	Text nodes (element content) are preserved. • Data within text nodes is escaped. • Entities are replaced.

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 195 for more information.

Examples

The following examples show instances of an XML document 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

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

Chapter 10. 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 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 203, which assist users in completing administrative tasks
- "Composition stored procedures" on page 211, which generate XML documents using data in existing database tables
- "Decomposition stored procedures" on page 220, 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 279.

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.

© Copyright IBM Corp. 1999, 2000

Calling XML Extenders stored procedures

In general, call the XML Extender using the following syntax:

```
CALL db2xml.function_entry_point
```

Where:

db2xml
: Specifies the library of XML Extender stored procedures. On UNIX operating systems, the library is stored in the `sqllib/function` directory. On Windows operating systems, it is stored in the *DXX_INSTALL*/`bin` directory.

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 38 and “Chapter 6. Managing XML collection data” on page 131.

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

Bind your database with the XML Extender stored procedure and DB2 CLI bind files. You can use a sample command file, `getstart_prep.cmd`, to bind the files. This command file is in the `DXX_INSTALL/samples/cmd` directory.

1. Connect to the database. For example:
```
db2 "connect to SALES_DB"
```
2. Change to the `DXX_INSTALL/bnd` directory and bind the XML Extender to the database.
```
db2 "bind @dxxbind.lst"
```
3. Change to the `sqllib/bnd` directory and bind the CLI to the database.
```
db2 "bind @db2cli.lst"
```
4. Terminate the connection.
```
db2 "terminate"
```

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**.

dxxEnableDB()

Purpose

Enables the database. When the database 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 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 225.
- 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 226.

```
dxxEnableDB(char(dbName) dbName,       /* input */
            long          returnCode,   /* output */
            varchar(1024) returnMsg)    /* output */
```

Parameters

Table 41. dxxEnableDB() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

dxxDisableDB()

Purpose

Disables the database. When the XML Extender disables the database, 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 225.
- 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 226.

Important: You must disable all XML columns before attempting to disable a database. The XML Extender cannot disable a database that contains columns or collections that are enabled for XML.

```
dxxDisableDB(char(dbName)     dbName,       /* input */
                  long          returnCode,   /* output */
                  varchar(1024) returnMsg)    /* output */
```

Parameters

Table 42. dxxDisableDB() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	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.

```
dxxEnableColumn(char(dbName) dbName,      /* input */
                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 43. dxxEnableColumn() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
tbName	The name of the table containing the XML column.	IN
colName	The name of the XML column.	IN
DAD	A CLOB containing the DAD file.	IN
tablespace	The table space that contains the side tables other than the default table space. If not specified, the default table space is used.	IN
defaultView	The name of the default view joining the application table and side tables.	IN
rootID	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

Table 43. dxxEnableColumn() parameters (continued)

Parameter	Description	IN/OUT parameter
returnCode	The return code from the stored procedure.	OUT
returnMsg	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.

```
dxxDisableColumn(char(dbName) dbName,      /* input */
                 char(tbName) tbName,      /* input */
                 char(colName) colName,     /* input */
                 long          returnCode,  /* output */
                 varchar(1024)  returnMsg)   /* output */
```

Parameters

Table 44. dxxDisableColumn() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
tbName	The name of the table containing the XML column.	IN
colName	The name of the XML column.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

dxxEnableCollection()

Purpose

Enables an XML collection that is associated with an application table.

```
dxxEnableCollection(char(dbName) dbName,      /* input */
                    char(colName) colName,     /* input */
                    CLOB(100K)    DAD,         /* input */
                    char(tablespace) tablespace,  /* input */
                    long          returnCode,  /* output */
                    varchar(1024)  returnMsg)   /* output */
```

Parameters

Table 45. dxxEnableCollection() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
colName	The name of the XML collection.	IN
DAD	A CLOB containing the DAD file.	IN
tablespace	The table space that contains the side tables other than the default table space. If not specified, the default table space is used.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	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.

```
dxxDisableCollection(char(dbName) dbName,      /* input */
                     char(colName) colName,     /* input */
                     long          returnCode,  /* output */
                     varchar(1024)  returnMsg)   /* output */
```

Parameters

Table 46. dxxDisableCollection() parameters

Parameter	Description	IN/OUT parameter
dbName	The database name.	IN
colName	The name of the XML collection.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	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 135.

```
dxxGenXML(CLOB(100K)     DAD,            /* input */
          char(resultTabName) resultTabName,   /* input */
          integer        overrideType     /* input */
          varchar(1024)  override,        /* input */
          integer        maxRows,         /* input */
          integer        numRows,         /* output */
          long           returnCode,      /* output */
          varchar(1024)  returnMsg)       /* output */
```

Parameters

Table 47. dxxGenXML() parameters

Parameter	Description	IN/OUT parameter
DAD	A CLOB containing the DAD file.	IN
resultTabName	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

Table 47. dxxGenXML() parameters (continued)

Parameter	Description	IN/OUT parameter
overrideType	A flag to indicate the type of the following *override* parameter: • **NO_OVERRIDE**: No override. • **SQL_OVERRIDE**: Override by an SQL_stmt. • **XML_OVERRIDE**: Override by an XPath-based condition.	IN
override	Overrides the condition in the DAD file. The input value is based on the *overrideType*. • **NO_OVERRIDE**: A NULL string. • **SQL_OVERRIDE**: A valid SQL statement. Using this *overrideType* requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file. • **XML_OVERRIDE**: A string that contains one or more expressions in double quotation marks separated by "AND". Using this *overrideType* requires that RDB_node mapping is used in the DAD file.	IN
maxRows	The maximum number of rows in the result table.	IN
numRows	The actual number generated rows in the result table.	OUT
returnCode	The return code from the stored procedure.	OUT

Table 47. dxxGenXML() parameters (continued)

Parameter	Description	IN/OUT parameter
returnMsg	The message text that is returned in case of error.	OUT

Examples

The following example assumes that a result table is created with the name of XML_ORDER_TAB, and that the table has one column of XMLVARCHAR type.

```
#include "dxx.h"
#include "dxxrc.h"

              EXEC SQL INCLUDE SQLCA;
              EXEC SQL BEGIN DECLARE SECTION;
              SQL TYPE is CLOB(100K)  dad;              /* DAD */
              SQL TYPE is CLOB_FILE   dadFile;          /* dad file */
              char                    result_tab[32];   /* name of the result table */
              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                   ovtype_ind;
              short                   ov_inde;
              short                   maxrow_ind;
              short                   numrow_ind;
              short                   returnCode_ind;
              short                   returnMsg_ind;

              EXEC SQL END DECLARE SECTION;

              /* create table */
              EXEC CREATE TABLE xml_order_tab (xmlorder XMLVarchar);

              /* read data from a file to a CLOB */
              strcpy(dadfile.name,"e:\dxx\dad\litem3.dad");
              dadfile.name_length = strlen("e:\dxx\dad\litem3.dad");
              dadfile.file_options = SQL_FILE_READ;
              EXEC SQL VALUES (:dadfile) INTO :dad;
              strcpy(result_tab,"xml_order_tab");
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '\0';
              collection_ind = 0;
              dad_ind = 0;
```

```
rtab_ind = 0;
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 dxxGenXML(:dad:dad_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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 135.

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.

```
dxxRetrieveXML(char(collectionName) collectionName,   /* input */
               char(resultTabName) resultTabName,    /* input */
               integer       overrideType,     /* input */
               varchar(1024)  override,         /* input */
               integer       maxRows,          /* input */
               integer       numRows,          /* output */
               long          returnCode,       /* output */
               varchar(1024)  returnMsg)        /* output */
```

Parameters

Table 48. dxxRetrieveXML() parameters

Parameter	Description	IN/OUT parameter
collectionName	The name of an enabled XML collection.	IN
resultTabName	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

Table 48. dxxRetrieveXML() parameters (continued)

Parameter	Description	IN/OUT parameter
overrideType	A flag to indicate the type of the following *override* parameter: • **NO_OVERRIDE**: No override. • **SQL_OVERRIDE**: Override by an SQL_stmt. • **XML_OVERRIDE**: Override by an XPath-based condition.	IN
override	Overrides the condition in the DAD file. The input value is based on the *overrideType*. • **NO_OVERRIDE**: A NULL string. • **SQL_OVERRIDE**: A valid SQL statement. Using this *overrideType* requires that SQL mapping is used in the DAD file. The input SQL statement overrides the SQL_stmt in the DAD file. • **XML_OVERRIDE**: A string that contains one or more expressions in double quotation marks, separated by "AND". Using this *overrideType* requires that RDB_node mapping is used in the DAD file.	IN
maxRows	The maximum number of rows in the result table.	IN
numRows	The actual number of generated rows in the result table.	OUT
returnCode	The return code from the stored procedure.	OUT

Table 48. dxxRetrieveXML() parameters (continued)

Parameter	Description	IN/OUT parameter
returnMsg	The message text that is returned in case of error.	OUT

Examples

The following 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.

```
#include "dxx.h"
#include "dxxrc.h"

            EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char    collection[32];    /* dad buffer */
              char    result_tab[32];    /* name of the result table */
              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   dadbuf_ind;
              short   rtab_ind;
              short   ovtype_ind;
              short   ov_inde;
              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);

              /* initialize host variable and indicators */
              strcpy(collection,"sales_ord");
              strcpy(result_tab,"xml_order_tab");
              override[0] = '\0';
              overrideType = NO_OVERRIDE;
              max_row = 500;
              num_row = 0;
              returnCode = 0;
              msg_txt[0] = '\0';
              collection_ind = 0;
              rtab_ind = 0;
              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 dxxRetrieve(:collection:collection_ind;
       :result_tab:rtab_ind,
       :overrideType:ovtype_ind,:override:ov_ind,
       :max_row:maxrow_ind,:num_row:numrow_ind,
       :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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

The dxxShredXML() stored procedure is the pairing stored procedure of dxxGenXML(). 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 dxxShredXML() does not require the primary-foreign key relationship among joining tables, which is created by the XML Extender during the enable collection process. However, the join condition columns that are specified in the RDB_node of the root element_node must exist in the tables.

```
dxxShredXML(CLOB(100K)    DAD,            /* input */
            CLOB(1M)      xmlobj,         /* input */
            long          returnCode,     /* output */
            varchar(1024) returnMsg)      /* output */
```

Parameters

Table 49. dxxShredXML() parameters

Parameter	Description	IN/OUT parameter
DAD	A CLOB containing the DAD file.	IN
xmlobj	An XML document object in XMLCLOB type.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

Examples

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 dad;           /* DAD*/
              SQL TYPE is CLOB_FILE  dadFile; /* DAD file*/
              SQL TYPE is CLOB xmlDoc;        /* input XML document */
              SQL TYPE is CLOB_FILE xmlFile;  /* input XMLfile */
              long        returnCode;         /* 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;
```

```
/* initialize host variable and indicators */
strcpy(dadFile.name,"c:\dxx\samples\dad\getstart_xcollection.dad");
dadFile.name_length=strlen("c:\dxx\samples\dad\getstart_xcollection.dad");
dadFile.file_option=SQL_FILE_READ;
strcpy(xmlFile.name,"c:\dxx\samples\cmd\getstart.xml");
xmlFile.name_length=strlen("c:\dxx\samples\cmd\getstart.xml");
xmlFile.file_option=SQL_FILE_READ;
SQL EXEC VALUES (:dadFile) INTO :dad;
SQL EXEC VALUES (:xmlFile) INTO :xmlDoc;
returnCode = 0;
returnMsg[0] = '\0';
dad_ind = 0;
xmlDoc_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxShredXML(:dad:dad_ind;
                :xmlDoc:xmlDoc_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

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.

```
dxxInsertXML(char(collectionName) collectionName,   /* input */
             CLOB(1M)       xmlobj,           /* input */
             long           returnCode,       /* output */
             varchar(1024)  returnMsg)        /* output */
```

Parameters

Table 50. dxxInsertXML() parameters

Parameter	Description	IN/OUT parameter
collectionName	The name of an enabled XML collection.	IN
xmlobj	An XML document object in CLOB type.	IN
returnCode	The return code from the stored procedure.	OUT
returnMsg	The message text that is returned in case of error.	OUT

Examples

In the following example, the dxxInsertXML() call decomposes the input XML document `e:\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.

```
#include "dxx.h"
#include "dxxrc.h"

EXEC SQL INCLUDE SQLCA;
            EXEC SQL BEGIN DECLARE SECTION;
              char                  collection[64];  /* name of an XML collection */
              SQL TYPE is CLOB_FILE xmlDoc;          /* input XML document */
              long                  returnCode;      /* error code */
              char                  returnMsg[1024]; /* error message text */
              short                 collection_ind;
              short                 xmlDoc_ind;
              short                 returnCode_ind;
              short                 returnMsg_ind;
              EXEC SQL END DECLARE SECTION;

              /* initialize host variable and indicators */
              strcpy(collection,"sales_ord")
              strcpy(xmlobj.name,"c:\dxx\samples\cmd\getstart.xml");
              xmlobj.name_length=strlen("c:\dxx\samples\cmd\getstart.xml");
```

```
xmlobj.file_option=SQL_FILE_READ;
returnCode = 0;
returnMsg[0] = '\0';
collection_ind = 0;
xmlobj_ind = 0;
returnCode_ind = -1;
returnMsg_ind = -1;

/* Call the store procedure */
EXEC SQL CALL db2xml.dxxInsertXML(:collection:collection_ind;
                :xmlobj:xmlobj_ind,
                :returnCode:returnCode_ind,:returnMsg:returnMsg_ind);
```

Chapter 11. Administrative 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.

The parameter limits listed in the support tables are documented in "Appendix D. The XML Extender limits" on page 279.

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 51.

Table 51. 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.
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.

© Copyright IBM Corp. 1999, 2000

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*). An XML_USAGE table is created at the time the database in enabled with the columns listed in Table 52.

Table 52. 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	The ID of the DTD in the DTD_REF table to define the DAD file. It is the foreign key.
DAD	The content of the DAD file that is associated with the 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.
access_mode	1 for XML collection, 0 for XML column

Restriction: The DTD can be modified by the application only when the USAGE_COUNT is zero.

Chapter 12. Diagnostic information

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 IBM Software Support. You should use the trace facility only under instruction from IBM Software Support.

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 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*.)

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.

© Copyright IBM Corp. 1999, 2000

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.

SQLSTATE values contain codes that supplement the SQLCODE messages. See "SQLSTATE codes" for a description of each SQLSTATE code that the XML Extender returns.

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" for the SQLSTATE values and the message number of associated messages that can be returned by the XML Extender. See "Messages" on page 233 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 233 and use the diagnostic information in the explanation.

SQLSTATE codes

Table 53 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 233 for information about each message.

Table 53. SQLSTATE codes and associated message numbers

SQLSTATE	Message No.	Description
00000	DXX*nnnn*I	No error has occurred.
01HX0	DXXD003W	The element or attribute specified in the path expression is missing from the XML document.

Table 53. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
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 53. 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 statment 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.

Table 53. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
	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.
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.

Table 53. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
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.
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.

Table 53. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
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 adminstrator 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 adminstrator 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 <*dtdid*> 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 occured 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 statment for XML generation.

Explanation: The SQL statment 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 statment.

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 descendents 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 descendents.

Diagnostic tracing

The XML Extender includes a trace facility that records XML Extender server activity. You should use the trace facility only under instruction of IBM Software Support.

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 to start or stop tracing. You can issue the command
| from a command line on an AIX, Windows NT, or Solaris server. 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 the directory that contains the trace file. When the trace is turned on, the file, `dxx`*`INSTANCE`*`.trc`, is placed in the specified directory. *INSTANCE* is the value of DB2INSTANCE. Each DB2 instance has its own log file.

Format

Starting the trace

```
▶▶—dxxtrc—on—trace_directory——————————————————————————————▶◀
```

Parameters

Table 54. Trace parameters

Parameter	Description
trace_directory	Name of the directory where the `dxx`*`INSTANCE`*`.trc` is placed. Required, no default.

Examples

The following example demonstrates starting the trace for an instance db2inst1 on AIX. The trace file, `dxxdb2inst1.trc`, is placed in the `/home/db2inst1/sqllib/log` directory.

```
dxxtrc on /home/db2inst1/sqllib/log
```

Stopping the trace

Purpose

Turns the trace off. No trace information is logged. Because running the trace can impact performance, it is recommended to turn trace off in a production environment.

Format

Stopping the trace

```
►►—dxxtrc—off———————————————————————————————————————————————►◄
```

Examples

This example shows that the trace facility is turned off.

```
dxxtrc off
```

Part 5. Appendixes

© Copyright IBM Corp. 1999, 2000

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 13 on page 254 shows the DTD for the DAD file. The elements of this file are described following the figure.

© Copyright IBM Corp. 1999, 2000

```
<?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 13. 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 13 on page 254.

DTDID element
: Specifies the ID of the DTD stored in the DTD_REF table. `DTDID` points to the DTD that validates the XML documents or guides the mapping between XML collection tables and XML documents. `DTDID` must be specified for XML collections. For XML columns, it is optional and is only needed if you want to create side tables for indexing on elements or attributes, or 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
Not yet implemented.

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"
- "Document access definition files" on page 262
  - "DAD file: XML column" on page 263
  - "DAD file: XML collection - SQL mapping" on page 263
  - "DAD file: XML - RDB_node mapping" on page 265

XML DTD

The following DTD is used for the `getstart.xml` document that is referenced throughout this book and shown in Figure 15 on page 262.

```
<!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 14. 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.

© Copyright IBM Corp. 1999, 2000

```
<?xml version="1.0"?>
<!DOCTYPE Order SYSTEM "c:\dxx\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 15. 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” on page 263
- “DAD file: XML collection - SQL mapping” on page 263 shows a DAD file for an XML collection using SQL mapping.
- “DAD file: XML - RDB_node mapping” on page 265 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 "c:\dxx\dad.dtd">
<DAD>
   <dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
   <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 16. Sample DAD file for an XML column

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 "c:\dxx\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 substr(char(timestamp(generate_unique())),16)
     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 "c:\dxx\samples\dtd\getstart.dtd"</doctype>
```

Figure 17. Sample DAD file for an XML collection using SQL mapping (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_occurence="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 17. Sample DAD file for an XML collection using SQL mapping (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 "c:\dxx\dtd\dad.dtd">
<DAD>
   <dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
   <validation>YES</validation>
<Xcollection>
      <prolog>?xml version="1.0"?</prolog>
      <doctype>!DOCTYPE Order SYSTEM "c:\dxx\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>
```

Figure 18. Sample DAD file for an XML collection using RDB_node mapping (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 18. Sample DAD file for an XML collection using RDB_node mapping (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 18. Sample DAD file for an XML collection using RDB_node mapping (Part 3 of 3)

Appendix C. Code page considerations

It is important to ensure that XML documents, and other related files, are encoded properly for each client or server that accesses the files. It is, therefore, important to understand the XML Extender assumptions when processing a file, and how it then 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 encoding.

The following sections describe the issues surrounding 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 actual code page of an XML document.

document encoding declaration
The name of the code page specified in the XML declaration. For example:

```
<?xml version="1.0" encoding="ibm037"?>
```

consistent document
A document in which the document encoding matches the document encoding declaration code page.

inconsistent document
A document in which the document encoding does not match the document encoding declaration code page.

DB2CODEPAGE registry (environment) variable
Specifies the code page of the data presented to DB2 from a database client application. DB2 gets the client's code page from the client's operating system locale, unless this variable is set. To DB2, this value overrides the client operating system locale if it is set.

© Copyright IBM Corp. 1999, 2000

client code page
: The application code page. If the DB2CODEPAGE variable is set, the client code page is the value of DB2CODEPAGE. Otherwise, the client code page is the client's operating system locale.

server code page, or server operating system locale code page
: The operating system locale on which the DB2 database is installed.

database code page:
: The encoding of the stored data, determined at database create time. If not explicitly specified with the USING CODESET clause, this value defaults to the server operating system locale.

DB2 and XML Extender code page assumptions

When DB2 receives or sends 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

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 55 summarizes the conversions that DB2 makes when importing an XML document.

Table 55. Using UDFs and stored procedures when the XML file is imported into the database

Processing method	Source code page for conversion	Destination code page for conversion	Comments
Inserting DTD file into DTD_REF table	Client code page	Database code page	

Table 55. Using UDFs and stored procedures when the XML file is imported into the database (continued)

Processing method	Source code page for conversion	Destination code page for conversion	Comments
Enable column or enable collection stored procedure, or administration commands, which import DAD file	Client code page	Database code page	
UDFs: • XMLVarcharFromFile() • XMLCLOBFromFile() • Content(): retrieve from XMLFILE to a CLOB	Server code page	Database code page	
Decomposition stored procedures	Client code page	Database code page	Document to be decomposed is assumed to be in client code page. Data from decomposition is stored in tables in database code page.

When an XML document is exported from the database, it is generally exported bases on a client request to present a document; a query of the contents of an XML document; or during the composition of a document from DB2 data. 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 56 summarizes the conversions that DB2 makes when exporting an XML document.

Table 56. Using UDFs and stored procedures when the XML file is exported from the database

Processing method	Conversion	Comments
UDF • XMLFileFromVarchar() • XMLFileFromCLOB()	Database code page to client code page at time when data is presented to client	
UDF • Content(): retrieve from XMLVARCHAR to an external server file	Database code page to server code page	

Table 56. Using UDFs and stored procedures when the XML file is exported from the database (continued)

Processing method	Conversion	Comments
Composition stored procedure: resulting tables stored in result table, which can be queried and exported.	Database code page to client code page when result set is presented to client	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.

Encoding declaration considerations

The encoding declaration declares the encoding of the XML document 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.

The recommended portable encodings for XML data are UTF-8 and UTF-16, according to the XML specification. Your application is interoperable between companies, if you use these encodings. If you use the encodings listed in Table 57, your application is likely to be portable 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 57. 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 57. Encoding declarations supported by XML Extender (continued)

Category	Encoding	OS	Code page
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
MBCS	gb2312	Windows NT	1386
	ibm-932, shift_jis78	AIX	932
	Shift_JIS	AIX 4.3 only, Windows NT	943
	IBM-eucCN	AIX, Sun	1383
	IBM-eucJP, EUC-JP	AIX, Linux, Sun	954, 33722
	euc-tw, IBM-eucTW	AIX, Sun	964
	euc-kr, IBM-eucKR	AIX	970
	big5	AIX, Sun, Windows NT	950

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 URL for a list of code pages supported by the XML parser used by the XML Extender:

`http://www.ibm.com/software/data/db2/extenders/xmlext/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 19 on page 274 shows that clients have consistent code pages with the document encoding and declared encoding.

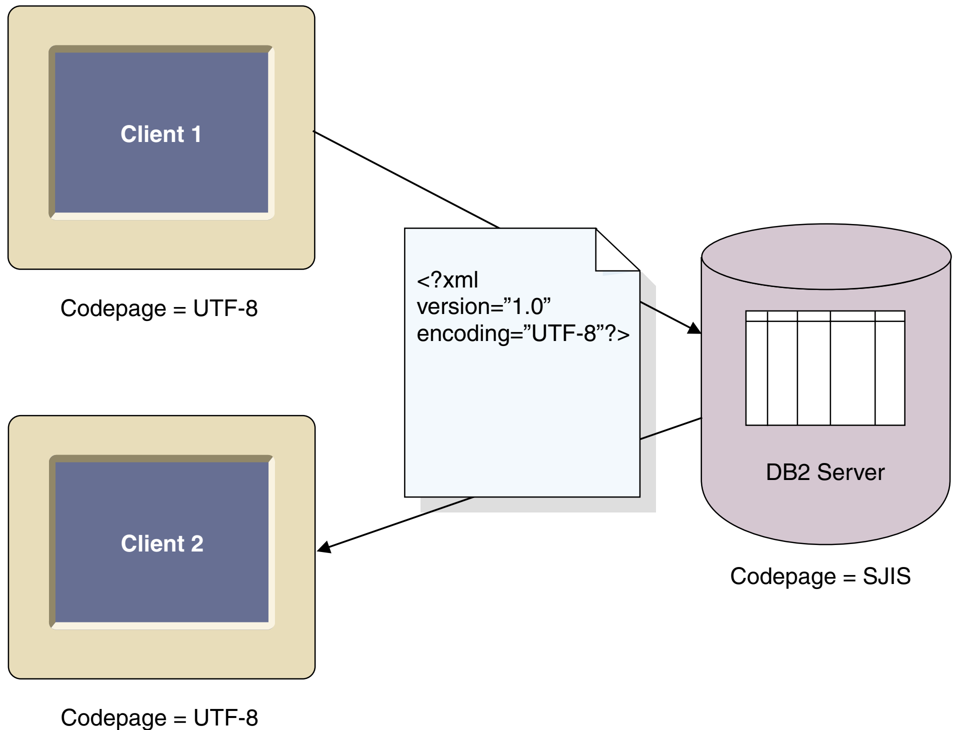


Figure 19. Clients have matching code pages

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 20 on page 275 shows an environment in which the code pages of the clients are inconsistent.

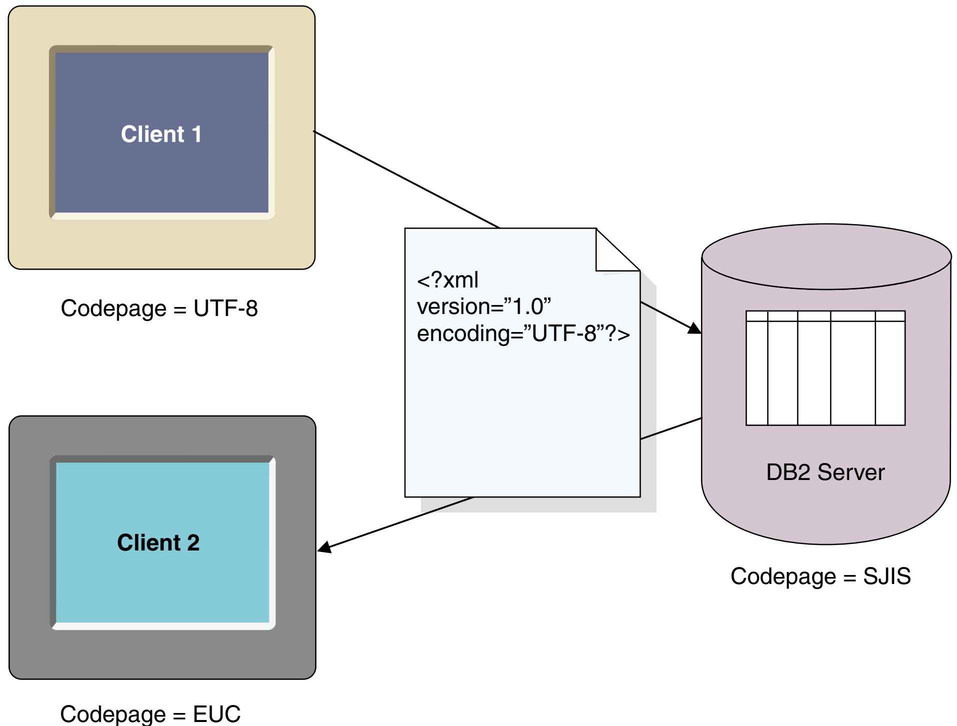


Figure 20. Clients have mismatching code pages

Client2 receives the document in EUC but the document will have an encoding declaration of UTF-8.

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="UTF-8" ?>
```

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 UTF-8, the server is UTF-8, and the database is UTF-8.

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 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 SJIS the client code page is SJIS, and the server and database code pages are UTF-8.

1. The document is imported into DB2 using XMLClobfromfile UDF from the server.
2. DB2 converts the document from SJIS and stores it in UTF-8. The encoding declaration and encoding are inconsistent in the database.
3. A client using SJIS requests the document for presentation at the Web browser.
4. DB2 converts document to SJIS, the client's code page. Document encoding and 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 SJIS for the incoming document. The server code page is SJIS, and the client and database are UTF-8.

1. The document is imported into the database using a storage UDF.
2. DB2 converts the document to UTF-8 from SJIS. The encoding and declaration are inconsistent.
3. A client with a UTF-8 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 SJIS and the encoding is UTF-8. The document cannot be processed by an XML parser or other XML processing tools.

Scenario 4: This scenario is a configuration with data loss, DB2 conversion, and a document imported from a UTF-8 server: The document encoding declaration is UTF-8, the server is UTF-8 and the database is SJIS.

1. The document is imported into DB2 using the XMLClobfromFile UDF.
2. DB2 converts the encoding to SJIS. When the document is imported, the document stored in the database might be corrupted because characters represented in UTF-8, might not have a representation in SJIS.

Scenario 5:

This scenario is a configuration with a Windows NT limitation. On Windows NT, operating system locales cannot be set to UTF-8, however, DB2 allows the client to set the code page to UTF-8 using `db2set DB2CODEPAGE=1208`. In this scenario, the client and server are on the same machine. The client is UTF-8, but the server cannot be set to UTF-8; its code page is 1252. The document is encoded as 1252 and the encoding declaration is ibm-1252. The database code page is UTF-8.

1. The document is imported from the server by a storage UDF and converted from 1252 to 1208.
2. The document is exported from DB2 using the Content() UDF by client.
3. DB2 converts the document from UTF-8 to 1252, even though client might expect 1208 because the client is on the same system as the server and is set to 1208.

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: (assumption: 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 querying the result table, have the client set the environment variable environment variable DB2CODEPAGE 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 Windows NT:** 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 XMLxxxfromFile 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.

Appendix D. The XML Extender limits

The XML Extender DAD file, stored procedures, and tables have the following limits.

Table 58. XML Extender limits

Value	Limit
Table in a decomposition XML collection	1024 rows from each decomposed XML document
Stored procedure parameters:	
XML document CLOB	1 MB[2]
Document Access Definition (DAD) CLOB	100KB[2]
collectionName	30
colName	30
dbName	18
defaultView	128
rootID	128
resultTabName	128
tablespace	128
tbName	128[1]
db2xml.DTD_REF table columns	
AUTHOR	128
CREATOR	128
UPDATOR	128
DTDID	128
CLOB	100 KB

Notes:

1. If *tbName* is qualified by a schema name, the entire name (including the separator character) must be no longer than 128 characters.
2. This size can be changed. See "Increasing the CLOB limit" on page 202 to learn how.

Names can undergo expansion when DB2 converts them from client code page to database code page. A name may fit within the size limit at the client, but exceed the limit when the stored procedure gets the converted name. See

© Copyright IBM Corp. 1999, 2000

the “National Language Support Application Development” section in the “Programming in Complex Environments” chapter of the *DB2 Application Development Guide* for more information.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other 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

© Copyright IBM Corp. 1999, 2000

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.

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 Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

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.

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.

COPYRIGHT LICENSE:

This information contains 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.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, or other countries, or both:

DB2	Net.Data
DB2 Extenders	OS/2
DB2 Universal Database	OS/390
IBM	OS/400
IMS	VTAM

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation 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, and service names 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.

© Copyright IBM Corp. 1999, 2000

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 is designed to be used by XSLT. Every location path can be expressed using the syntax defined for XPath.

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.

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 44
- planning 44
- XML collections 53, 55
- XML columns 53, 55

access method
- choosing an 44
- introduction 6
- planning an 44
- XML collections 10
- XML column 6

adding
- nodes 87, 93, 101
- side tables 74, 75

administration
- db2cmd command 65
- dxxadm command 151
- stored procedures 201
- support tables
  - DTD_REF 225
  - XML_USAGE 226
- tasks 65
- tools 5, 44
- wizard 65

administration stored procedures
- dxxDisableCollection() 210
- dxxDisableColumn() 208
- dxxDisableDB() 205
- dxxEnableCollection() 209
- dxxEnableColumn() 206
- dxxEnableDB() 204

administration support tables
- DTD_REF 225
- XML_USAGE 225

administration wizard
- Disable a Column window 82
- Enable a Column window 77
- introduction to the 65
- logging in 67
- setting up 65
- Side table window 74
- specifying address 67
- specifying JDBC driver 67
- specifying user ID and password 67
- starting 65

attribute_node 55, 63

authorization requirements 43

available operating systems 3

B

B-tree indexing 49

bibliography xiv

binding stored procedures 203

C

calling stored procedures 202

casting
- default, functions 113
- parameter markers 129

cleaning up, getting started 39

client code page 269

CLOB limit, increasing for stored procedures 202

code pages
- client 269
- considerations 269
- data loss 277
- database 269
- DB2 assumptions 270
- document encoding consistency 269, 270, 277
- encoding declaration 272
- exporting documents 271
- importing documents 270
- legal encoding declarations 272
- preventing inconsistent documents 277, 278
- server 269
- supported encoding declarations 272
- terminology 269
- UDFs and stored procedures 270, 271
- Windows NT UTF-8 limitation 277, 278
- XML Extender assumptions 270

codes
- messages and 233
- SQLSTATE 228

column data
- available UDTs 47
- managing XML documents as 113
- storing XML documents as 72

column type, for decomposition 63

command options
- disable_collection 161

command options *(continued)*
- disable_column 157
- disable_db 153
- enable_collection 159
- enable_column 155
- enable_db 152

composing XML documents 32

composite key
- for decomposition 62
- XML collections 62

composition
- dxxGenXML() 132
- dxxRetrieveXML() 132, 134
- of XML documents 38
- overriding the DAD file 135
- stored procedures
  - dxxGenXML() 38, 212
  - dxxRetrieveXML() 216
- XML collection 131

conditions
- optional 58, 62
- RDB_node mapping 61
- SQL mapping 58, 60

Content() function
- for retrieval 118
- retrieval functions using 171
- XMLCLOB to an external server file 176
- XMLFile to a CLOB 172
- XMLVarchar to an external server file 174

creating
- a database 19, 31
- DAD 72
- db2xml schema 69, 109
- indexes 25, 81
- nodes 87, 93, 101
- side tables 74, 75
- UDFs 69, 109
- UDTs 69, 109
- XML collections 32
- XML columns 20
- XML table 76

D

DAD
- attribute_node 55
- creating for XML collections 32
  - from the command shell 87, 93, 101

DAD (continued)
RDB_node mapping 90, 98
SQL mapping 84
creating for XML columns 21
from the command shell 74
using the administration wizard 72
defining side tables 17
definition of 10, 12
DTD for the 253
editing for XML collections
from the command shell 87, 93, 101
editing for XML columns
from the command shell 74
using the administration wizard 72
element_node 55, 61
examples of 262
RDB_node mapping 265
SQL mapping 263
file for XML column 77
for XML collections 32
for XML columns 53, 54
introduction to the 6
mapping scheme 32, 83
node definitions
attribute_node 55
element_node 55
RDB_node 61
root_node 55
text_node 55
overriding the 135
planning for the 53, 54
tutorial 30
XML collections 53
XML column 21, 53
RDB_node 61
root element_node 61
root_node 55
samples of 262
size limit 53, 54, 279
text_node 55
data loss, inconsistent encodings 277
data types
XMLCLOB 163
XMLFile 163
XMLVarchar 163
database
code page 269
creating 19, 31
enabling for XML 19, 31, 69, 108
relational 57
DB2
and XML documents 3
composing XML documents 10
decomposing XML documents 10
integrating XML documents 5
storing untagged XML data 10
storing XML documents 6
db2cmd command 65
db2xml 7, 225, 226
decomposition
collection table limit 279
composite key 62
DB2 table sizes 63, 140
dxxInsertXML() 141, 142
dxxShredXML() 141
of XML collections 139
specifying the column type for 63
specifying the orderBy attribute 62
specifying the primary key for 62
stored procedures
dxxInsertXML() 223
dxxShredXML() 221
default casting function
managing XML column data 113
retrieval 117, 171
storage 115, 166
update 122, 195
default view, side tables 48
deleting
nodes 87, 93, 101
side tables 75
diagnosis information
messages and codes 233
SQLSTATE codes 228
stored procedure return codes 228
tracing 246
UDF return codes 227
Disable a Column window 82
disable_collection command 161
disable_column command 157
disable_db command 153
disabling
administration command 151
databases for XML, stored procedure 205
disable_collection command 161
disable_column command 157
disable_db command 153
stored procedure 205, 208, 210
disabling (continued)
XML collections
from the command shell 108
stored procedure 210
using the administration wizard 107
XML columns
from the command shell 82
stored procedure 208
using the administration wizard 81
document access definition (DAD)
creating 72
creating for XML collections
from the command shell 87, 93, 101
RDB_node mapping 90, 98
SQL mapping 84
creating for XML columns
from the command shell 74
using the administration wizard 72
DTD for the 253
editing 72
editing for XML collections
from the command shell 87, 93, 101
editing for XML columns
from the command shell 74
using the administration wizard 72
file for XML column 77
mapping scheme 83
planning for the 53, 54
XML collections 53
XML columns 53
document encoding declaration 269
document type definition 70
documentation, including in Information Center xi
DTD
availability 4
for getting started lessons 15, 28
for the DAD 253
for validation 47
inserting 20
inserting from the command shell 71
planning 15, 28
publication 4
repository
DTD_REF 6, 225
storing in 70
structured searches 47
using multiple 47, 53

DTD (continued)
validating with a 47
DTD_REF table 70
column limits 279
inserting a DTD 71
schema 225
DTDID 71, 225, 226
DXX_SEQNO for multiple occurrence 48
dxxadm
disable_collection command 161
disable_column command 157
disable_db 109
disable_db command 153
enable_collection command 159
enable_column command 155
enable_db 69
enable_db command 152
introduction to 151
syntax 151
dxxDisableCollection() stored procedure 210
dxxDisableColumn() stored procedure 208
dxxDisableDB() stored procedure 205
dxxEnableCollection() stored procedure 209
dxxEnableColumn() stored procedure 206
dxxEnableDB() stored procedure 204
dxxGenXML() 38
dxxGenXML() stored procedure 132, 212
dxxInsertXML() stored procedure 141, 223
dxxRetrieveXML() stored procedure 132, 216
DXXROOT_ID 23, 50
dxxShredXML() stored procedure 141, 221
dxxtrc command 246, 248, 249
dynamically overriding the DAD file, composition 135

E

editing
side tables 74, 75
XML table 76
element_node 55, 62
Enable a Column window 77
enable_collection command 159
enable_column command 155
enable_db command
creating XML_USAGE table 225, 226
option 152
enabling
administration command 151
database tasks 69, 109
databases for XML 19, 31
from the command shell 69, 109
stored procedure 204
using the administration wizard 69, 109
enable_collection command 159
enable_column command 155
enable_db command 152
stored procedure 204, 206, 209
XML collections
from the command shell 106
requirements 140
stored procedure 209
using the administration wizard 106
XML columns
for Text Extender 127
from the command shell 23, 78
stored procedure 206
using the administration wizard 77
encoding
conversion by DB2 270, 271, 277
declaration considerations 272
declarations 269
legal, declarations 272
of a document 269
supported declarations 272
XML documents 269
existing DB2 data 10
eXtensible Markup Language (XML) 4
Extensive Stylesheet Language Transformation 51
extractChar() function 185
extractChars() function 185
extractCLOB() function 188
extractCLOBs() function 188
extractDate() function 190
extractDates() function 190
extractDouble() function 182
extractDoubles() function 182
extracting document fragments 188
extracting functions
description of 165
extractChar() 185
extracting functions (continued)
extractChars() 185
extractCLOB() 188
extractCLOBs() 188
extractDate() 190
extractDates() 190
extractDouble() 182
extractDoubles() 182
extractInteger() 179
extractIntegers() 179
extractReal() 184
extractReals() 184
extractSmallint() 181
extractSmallints() 181
extractTime() 191
extractTimes() 191
extractTimestamp() 193
extractTimestamps() 193
extractVarchar() 186
extractVarchars() 186
fragments of XML documents 188
introduction to 178
table of 121
extractInteger() function 179
extractIntegers() function 179
extractReal() function 184
extractReals() function 184
extractSmallint() function 181
extractSmallints() function 181
extractTime() function 191
extractTimes() function 191
extractTimestamp() function 193
extractTimestamps() function 193
extractVarchar() function 186
extractVarchars() function 186

F

FROM clause 60
functions
Content(): from XMLCLOB to file 176
Content(): from XMLFILE to CLOB 172
Content(): from XMLVARCHAR to file 174
default casting 114, 116, 121
extractChar() 185
extractChars() 185
extractCLOB() 188
extractCLOBs() 188
extractDate() 190
extractDates() 190
extractDouble() 182
extractDoubles() 182
extracting 165, 178

functions *(continued)*
  extractInteger() 179
  extractIntegers() 179
  extractReal() 184
  extractReals() 184
  extractSmallint() 181
  extractSmallints() 181
  extractTime() 191
  extractTimes() 191
  extractTimestamp() 193
  extractTimestamps() 193
  extractVarchar() 186
  extractVarchars() 186
  for XML columns 165
  limitations when invoking from JDBC 129
  limits 279
  retrieval 116
    description 165
    from external storage to memory pointer 171
    from internal storage to external server file 171
    introduction 171
  storage 114, 165, 166
  summary table of 166
  update 121, 165, 195
  XMLCLOB to an external server file 176
  XMLCLOBFromFile() 168
  XMLFile to a CLOB 172
  XMLFileFromCLOB() 170
  XMLFileFromVarchar() 169
  XMLVarchar to an external server file 174
  XMLVarcharFromFile() 167

G

getting started lessons
  cleaning up 39
  collection tables 27
  composing the XML document 38
  creating DAD files 21, 30, 32, 34
  creating indexes 25
  creating the database 19, 31
  creating the XML collection 32
  creating the XML column 20
  defining side tables 17
  enabling the database 19, 31
  inserting the DTD 20
  introduction 13
  overview 13
  planning 15, 28
  searching the XML document 26
getting started lessons *(continued)*
  storing the XML document 25
getting started scripts 18, 30

H

highlighting conventions xi

I

importing the DTD 70
include files for stored procedures 201
indexes, for side tables 25, 81
indexing
  considerations 50
  multiple indexes 50
  ROOT ID 50
  side tables 49
  Text Extender structural-text 50
  with side tables 17, 49
  with Text Extender 49
  with XML columns 49
  XML columns 49
  XML documents 49
  XML documents with multiple occurrence 50
Information Center, including this book in xi
installing
  installation directory DXX_INSTALL 10, 12
  the XML Extender 43
invoking the administration wizard 67

J

JDBC, limitations when invoking functions 166
JDBC, limitations when invoking UDFs 129
JDBC address, for wizard 67
JDBC driver, for wizard 67
join conditions
  RDB_node mapping 61
  SQL mapping 60

L

limits
  stored procedure parameters 131, 201, 225
  The XML Extender 279
location path
  introduction to 8, 50
  restrictions 52
  simple 52
  syntax 51
  XPath 9, 51
  XSL 9, 51
location path *(continued)*
  XSLT 9, 51
logging in, for wizard 67

M

maintaining document structure 6
management
  column data 113
  retrieving column data 116
  searching XML documents 123
  storing column data 114
  updating column data 121
  XML collection data 131
  XML column data 113
mapping scheme
  creating the DAD for the 32, 83
  determining RDB_node mapping 58
  determining SQL mapping 58
  editing the DAD for the 83
  figure of DAD for the 46
  for XML collections 46
  for XML columns 46
  FROM clause 60
  introduction 10
  ORDER BY clause 61
  RDB_node mapping requirements 61, 62
  requirements 59
  SELECT clause 59
  SQL mapping requirements 59
  SQL mapping scheme 59
  SQL_stmt 57
  WHERE clause 61
messages and codes 233
multiple DTDs
  XML collections 53
  XML columns 47
multiple occurrence
  affecting table size 63, 140
  deleting elements and attributes 145
  DXX_SEQNO 48
  indexing XML documents with 50
  order of elements and attributes 140
  orderBy attribute 62
  preserving the order of elements and attributes 146
  recomposing documents with 62
  searching elements and attributes 126
  updating collections 144
  updating elements and attributes 123, 144, 198

multiple occurrence *(continued)*
updating XML documents 123, 198
multiple_occurrence attribute 35

N

nodes
add new 87, 93, 101
attribute_node 55
creating 87, 93, 101
DAD configuration 34, 88, 94, 102
deleting 87, 93, 101
element_node 55
RDB_node 61
removing 87, 93, 101
root, creating 87
root_node 55
text_node 55
notices 281

O

operating systems supported 3
ORDER BY clause 61
orderBy attribute
for decomposition 62
for multiple occurrence 62
XML collections 62
overloaded function, Content() 171
overrideType
No override 135
SQL override 135
XML override 135
overriding the DAD file 135

P

parameter markers in functions 129, 166
performance
default views of side tables 48
indexing side tables 49
searching XML documents 49
stopping the trace 249
planning
a mapping scheme 57
choosing a storage method 44
choosing an access and storage method 44
choosing an access method 44
choosing to validate XML data 47, 53
determining column UDT 16
determining document structure 28
determining DTD 15, 28
for the DAD 53, 54
planning *(continued)*
for XML collections 54
for XML columns 53
getting started lessons 15, 28
mapping XML document and database 17, 29
side tables 48
the XML collections mapping scheme 57
to index XML columns 49
validating with multiple DTDs 47, 53
primary key for decomposition 62
primary key for side tables 23, 48, 50
problem determination 227
processing instructions 56, 97

R

RDB_node mapping
composite key for decomposition 62
conditions 61
creating a DAD 90, 98
decomposition requirements 62
determining for XML collections 58
requirements 61
specifying column type for decomposition 63
related information xiv
relational tables 131
removing
nodes 87, 93, 101
side tables 75
repository, DTD 70
restrictions for the location path 52
retrieval functions
Content() 171
description of 165
from external storage to memory pointer 171
from internal storage to external server file 171
introduction to 171
XMLCLOB to an external server file 176
XMLFile to a CLOB 172
XMLVarchar to an external server file 174
retrieval of data
attribute values 119
element contents 119
entire document 117
return codes
stored procedures 228
return codes *(continued)*
UDF 227
ROOT ID
default view of side tables 48
indexing 50
indexing considerations 50
specifying 23, 79
root_node 55

S

sample files 18, 30
schema
db2xml 69
DTD_REF table 71, 225, 226
name for stored procedures 11
name for user data types 7
name for user defined functions 8
searching
direct query on side tables 125
document structure 124
from a joined view 125
multiple occurrence 126
side tables 26
Text Extender structural text 126
with extracting UDFs 125
XML collection 146
XML documents 26, 123
SELECT clause 59
server code page 269
setting up
administration wizard 65
the XML Extender 43
setting up XML collections
adding the DAD 83
creating the DAD 83
disabling 107
editing the DAD 83
enabling 105
setting up XML columns
altering an XML table 76
creating an XML table 76
creating the DAD 72
disabling 81
editing an XML table 76
editing the DAD 72
enabling 77
side tables
add new 74, 75
create 74
default view 48
definition of 10
deleting 74
DXX_SEQNO 48
DXXROOT_ID 23
editing 74, 75

side tables *(continued)*
getting started lessons 17
indexing 17, 49
planning 48
removing 74, 75
ROOT ID 23
searching 17, 26, 123
specifying ROOT ID 79
updating 122
sizes, limits 279
software requirements 43
SQL mapping
creating a DAD 34, 84
determining for XML collections 58
FROM clause 60
ORDER BY clause 61
requirements 59
SELECT clause 59
SQL mapping scheme 59
WHERE clause 61
SQL override 135
SQL_stmt
FROM clause 60
ORDER_BY clause 61
SELECT clause 59
WHERE clause 61
SQLSTATE codes 228
starting
administration wizard 65
the administration wizard 67
the XML Extender 43
storage functions
description of 165
introduction to 166
storage UDF table 115
XMLCLOBFromFile() 168
XMLFileFromCLOB() 170
XMLFileFromVarchar() 169
XMLVarcharFromFile() 167
storage method
choosing a 44
introduction 6
planning a 44
XML collections 10
XML column 6
storage UDFs 115, 122
stored procedures
administration 201, 203
dxxDisableCollection() 210
dxxDisableColumn() 208
dxxDisableDB() 205
dxxEnableCollection() 209
dxxEnableColumn() 206
dxxEnableDB() 204
stored procedures *(continued)*
binding 203
calling 202
code page considerations 270, 271, 277
composition 201, 211
dxxGenXML() 212
dxxRetrieveXML() 216
decomposition 201, 220
dxxInsertXML() 223
dxxShredXML() 221
dxxDisableCollection() 210
dxxDisableColumn() 208
dxxDisableDB() 205
dxxEnableCollection() 209
dxxEnableColumn() 206
dxxEnableDB() 204
dxxGenXML() 38, 132, 212
dxxInsertXML() 141, 223
dxxRetrieveXML() 132, 216
dxxShredXML() 141, 221
include files 201
return codes 228
update 201
storing the DTD 70
structure
hierarchical 29
of DTD 29
of mapping 17, 29
of XML document 29
relational tables 17, 29
stylesheets 56, 97
syntax diagram
disable_collection command 161
disable_column command 157
disable_db command 153
dxxadm 151
enable_collection command 159
enable_column command 155
enable_db command 152
extractChar() function 185
extractChars() function 185
extractCLOB() function 188
extractCLOBs() function 188
extractDate() function 190
extractDates() function 190
extractDouble() function 182
extractDoubles() function 182
extractInteger() function 179
extractIntegers() function 179
extractReal() function 184
extractReals() function 184
extractSmallint() function 181
extractSmallints() function 181
extractTime() function 191
syntax diagram *(continued)*
extractTimes() function 191
extractTimestamp() function 193
extractTimestamps() function 193
extractVarchar() function 186
extractVarchars() function 186
how to read xii
location path 51
Update() function 195
XMLCLOB to an external server file Content() function 176
XMLCLOBFromFile() function 168
XMLFile to a CLOB Content() function 172
XMLFileFromCLOB() function 170
XMLFileFromVarchar() function 169
XMLVarchar to an external server file Content() function 174
XMLVarcharFromFile() function 167

T

table of UDFs 166
tables sizes, for decomposition 63, 140
terminology 10, 11
Text Extender
enabling for search 127
enabling XML columns for 127
searching with 127
supported operating systems 126
text_node 55, 63
tracing
dxxtrc command 246
starting 248
stopping 249
trademarks 283
transfer of documents between client and server, considerations 269

U

UDFs
code page considerations 270, 271, 277
definition of 10
extractChar() 185
extractChars() 185
extractCLOB() 188
extractCLOBs() 188
extractDate() 190
extractDates() 190

UDFs (continued)
extractDouble() 182
extractDoubles() 182
extracting functions 178
extractInteger() 179
extractIntegers() 179
extractReal() 184
extractReals() 184
extractSmallint() 181
extractSmallints() 181
extractTime() 191
extractTimes() 191
extractTimestamp() 193
extractTimestamps() 193
extractVarchar() 186
extractVarchars() 186
for XML columns 165
from external storage to memory pointer 171
from internal storage to external server file 171
limitations when invoking from JDBC 166
retrieval functions 171
return codes 227
searching with 125
summary table of 166
Update() 122, 195
XMLCLOB to an external server file 176
XMLCLOBFromFile() 168
XMLFile to a CLOB 172
XMLFileFromCLOB() 170
XMLFileFromVarchar() 169
XMLVarchar to an external server file 174
XMLVarcharFromFile() 167
UDTs
definitions 10, 113
introduction to 7
summary table of 47
XML table 77
XMLCLOB 47
XMLFILE 47
XMLVARCHAR 47
Update() function 122, 195
update function
description of 165
document replacement behavior 196
introduction to 195
updating
how the Update() UDF replaces XML documents 196
side tables 122
updating (continued)
XML column data 121
attributes 122
entire document 122
multiple occurrence 123, 198
specific elements 122
user-defined functions (UDFs)
for XML columns 165
searching with 125
summary table of 166
Update() 122, 195
user-defined types (UDTs) 113
user ID and password, for wizard 67

V

validate XML data
considerations 47, 53
deciding to 47, 53
DTD requirements 47, 53
validating
DTD 70
performance impact 47, 54
using a DTD 47
XML data 47

W

WHERE clause 61
Windows NT UTF-8 limitation, code pages 277, 278

X

XML 4
XML applications 4
XML collections
composition 131
creating 32
creating a DAD
from the command shell 87, 93, 101
RDB_node mapping 90, 98
SQL mapping 84
DAD, planning for 53
decomposition 139
defining 71
definition of 6, 12
determining a mapping scheme for 57
disabling 107
from the command shell 108
using the administration wizard 107
DTD for validation 70
editing the DAD
from the command shell 87, 93, 101
XML collections (continued)
enabling 105
from the command shell 106
using the administration wizard 106
introduction to 10
managing XML collection data 131
mapping scheme 57
creating the DAD 83
editing the DAD 83
mapping schemes 58
RDB_node mapping 58
scenarios 45
searching 146
setting up 83
SQL mapping 58
storage and access methods 6, 10
validation 70
when to use 45
XML columns
adding 23
configuring 72
creating 20
creating the DAD 21
from the command shell 74
using the administration wizard 72
DAD, planning for 53
default view of side tables 48
defining 71
definition of 6, 10
disabling
from the command shell 82
using the administration wizard 81
editing the DAD
from the command shell 74
using the administration wizard 72
enabling 23
from the command shell 78
using the administration wizard 77
figure of side tables 49
indexing 49
introduction to 6
location path 50
maintaining document structure 6
managing XML documents 113
preparing the DAD 21
retrieving data
attribute values 119

XML columns *(continued)*
element contents 119
entire document 117
sample DAD file 263
scenarios 45
setting up 72
side tables 24
storage and access methods 6
storing data 114
the DAD for 53
UDFs 165
updating XML data
attributes 122
entire document 122
specific elements 122
view columns 24
view side tables 24
when to use 45
with side tables 49
XML type 23
XML documents
B-tree indexing 49
code page assumptions 270
code page consistency 269, 270, 271, 277
composing 32, 132
creating indexes 25, 81
decomposition 139, 141
default casting functions 25
deleting 128
encoding declarations 272
exporting, code page conversion 271
importing, code page conversion 270, 277
indexing 49
introduction to 3
legal encoding declarations 272
mapping to tables 17, 29
searching 26, 123
direct query on side tables 125
document structure 124
from a joined view 125
multiple occurrence 126
Text Extender structural text 126
with extracting UDFs 125
stored in DB2 3
storing 25
supported encoding declarations 272
XML DTD repository
DTD Reference Table (DTD_REF) 6
XML DTD repository *(continued)*
introduction to 6
XML Extender
available operating systems 3
capabilities 6
features 6
functions 165
installing 43
introduction to 3
XML Extender stored procedures 201
XML override 135
XML Path Language 9, 51
XML repository 44
XML Stylesheet Language Transformation 9
XML table
creating 76
definition of 10
editing 76
XML_USAGE table 226
XMLCLOB to an external server file function 176
XMLClobFromFile() function 168
XMLFile to a CLOB function 172
XMLFileFromCLOB() function 170
XMLFileFromVarchar() function 169
XMLVarchar to an external server file function 174
XMLVarcharFromFile() function 167
XPath 9, 51
XSLT 9, 51, 58

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. This guide suggests information that you can gather to help DB2 Customer Support to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

http://www.ibm.com/software/data/
The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more.

http://www.ibm.com/software/data/db2/library/
The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information.

Note: This information may be in English only.

http://www.elink.ibmlink.ibm.com/pbl/pbl/
The International Publications ordering Web site provides information on how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products, including DB2.

© Copyright IBM Corp. 1999, 2000

ftp.software.ibm.com
Log on as anonymous. In the directory `/ps/products/db2`, you can find demos, fixes, information, and tools relating to DB2 and many other products.

comp.databases.ibm-db2, bit.listserv.db2-l
These Internet newsgroups are available for users to discuss their experiences with DB2 products.

On Compuserve: GO IBMDB2
Enter this command to access the IBM DB2 Family forums. All DB2 products are supported through these forums.

For information on how to contact IBM outside of the United States, refer to Appendix A of the *IBM Software Support Handbook*. To access this document, go to the following Web page: http://www.ibm.com/support/, and then select the IBM Software Support Handbook link near the bottom of the page.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.