Integrating XML with DB2 XML Extender and DB2 Text Extender

Integrate XML with IBM WebSphere and VisualAge for Java
Compose and decompose XML documents
Store and access XML documents with SQL

Luis Ennser
Christophe Delporte
Mikio Oba
K. M. Sunil

ibm.com/redbooks
Integrating XML with DB2 XML Extender and DB2 Text Extender

December 2000
First Edition (December 2000)


Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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

© Copyright International Business Machines Corporation 2000. All rights reserved.
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Take Note!
Before using this information and the product it supports, be sure to read the general information in Appendix F, “Special notices” on page 299.
## Contents

Figures ................................................................................. ix

Tables ................................................................................. xiii

Preface ....................................................................................... xv

The team that wrote this redbook ........................................ xv

Comments welcome .............................................................. xvii

### Part 1. The theory of XML .................................................. 1

**Chapter 1. Introduction to XML** ........................................ 3

1.1 What is XML? .............................................................. 3

1.2 XML business benefits ................................................ 4

1.2.1 Information sharing ............................................... 4

1.2.2 XML inside a single application .............................. 5

1.2.3 Content delivery .................................................... 5

1.3 Technological benefits of XML ..................................... 6

1.3.1 An example of using XML ..................................... 6

1.3.2 Major benefits ...................................................... 6

1.4 XML concepts ............................................................ 8

1.4.1 Document validity and well-formedness ................... 9

1.4.2 Document type definition ....................................... 10

1.4.3 Namespaces ....................................................... 16

1.4.4 DTD versus XML schemas ................................... 17

1.5 XPath ........................................................................... 18

1.6 eXtensible stylesheet language (XSL) ............................ 20

1.6.1 Cascading Style Sheets ......................................... 20

1.6.2 XSL = fo: + XSLT ............................................... 21

1.6.3 XSL transformations ............................................. 23

1.7 Real-life uses of XML ................................................ 23

1.7.1 Resource description frameWork (RDF) .................... 23

1.7.2 SABRE and Wireless Markup Language ................ 24

1.8 Processing XML using Java .......................................... 24

1.8.1 XML applications ............................................... 25

1.8.2 SAX ............................................................... 25

1.8.3 DOM ............................................................ 29

1.8.4 SAX or DOM? ................................................ 31

**Chapter 2. DB2 XML Extender** ......................................... 33

2.1 Introduction ............................................................. 33

2.1.1 Storage and access methods .................................. 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.2</td>
<td>Document Type Definition (DTD) repository</td>
<td>35</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Document Access Definition</td>
<td>35</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Location path</td>
<td>36</td>
</tr>
<tr>
<td>2.2</td>
<td>Installation</td>
<td>36</td>
</tr>
<tr>
<td>2.3</td>
<td>Enabling the database to XML</td>
<td>42</td>
</tr>
<tr>
<td>2.4</td>
<td>XML Columns method</td>
<td>44</td>
</tr>
<tr>
<td>2.4.1</td>
<td>XML example used as illustration</td>
<td>44</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Table creation</td>
<td>46</td>
</tr>
<tr>
<td>2.4.3</td>
<td>DAD creation</td>
<td>49</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Enabling the XML Column</td>
<td>57</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Insert, delete, extract, and update functions</td>
<td>60</td>
</tr>
<tr>
<td>2.4.6</td>
<td>Disabling the XML Column</td>
<td>65</td>
</tr>
<tr>
<td>2.5</td>
<td>XML Collections method</td>
<td>67</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Scenario to Illustrate composing an XML document</td>
<td>68</td>
</tr>
<tr>
<td>2.5.2</td>
<td>DAD with SQL_stmt</td>
<td>69</td>
</tr>
<tr>
<td>2.5.3</td>
<td>DAD with RDB_node</td>
<td>84</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Composition stored procedures</td>
<td>90</td>
</tr>
<tr>
<td>2.5.5</td>
<td>Dynamically overriding values in the DAD file</td>
<td>91</td>
</tr>
<tr>
<td>2.5.6</td>
<td>Enabling XML Collections</td>
<td>93</td>
</tr>
<tr>
<td>2.5.7</td>
<td>Decomposing XML documents using RDB_node mapping</td>
<td>95</td>
</tr>
<tr>
<td>2.5.8</td>
<td>Decomposition stored procedures</td>
<td>99</td>
</tr>
<tr>
<td>2.5.9</td>
<td>Disabling XML Collections</td>
<td>100</td>
</tr>
<tr>
<td>2.6</td>
<td>Disabling the database for XML</td>
<td>102</td>
</tr>
<tr>
<td>3.1</td>
<td>DB2 Text Extender installation</td>
<td>103</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Installation requirements</td>
<td>103</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Installing the DB2 Extenders</td>
<td>104</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Creating a DB2 Text Extender instance</td>
<td>105</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Dropping a Text Extender instance</td>
<td>106</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Checking the installation</td>
<td>106</td>
</tr>
<tr>
<td>3.2</td>
<td>Text Extender indexing capabilities</td>
<td>108</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Linguistic indexes</td>
<td>109</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Precise indexes</td>
<td>110</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Ngram indexes</td>
<td>110</td>
</tr>
<tr>
<td>3.3</td>
<td>Text Extender environment variables</td>
<td>111</td>
</tr>
<tr>
<td>3.4</td>
<td>Text extender use</td>
<td>113</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Starting and stopping the DB2 Text Extender server</td>
<td>113</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Database enablement for Text Extender</td>
<td>113</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Table enablement for Text Extender</td>
<td>116</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Column enablement for Text Extender</td>
<td>116</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Querying the text documents</td>
<td>119</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Structural search on XML documents</td>
<td>122</td>
</tr>
</tbody>
</table>
6.6.2 The JSP files: SupMainMenu and SupB2B .................................................. 203
6.6.3 The servlets used on the supplier side ............................................................. 205

Chapter 7. XMLapp: implementation with XML Collections .......................... 207
7.1 Application overview ............................................................................................. 207
7.2 Database architecture ............................................................................................ 207
7.3 XML application implementation ......................................................................... 209
    7.3.1 Logon and user profile management .............................................................. 209
    7.3.2 Search engine ................................................................................................... 221
    7.3.3 Implementation of shopping basket and ordering system .................................. 224
    7.3.4 Order tracking ................................................................................................. 235
    7.3.5 Supplier .............................................................................................................. 241

Chapter 8. XMLapp: installing XML Columns version in WAS .................... 249
8.1 Files contained in SG246130 Columns directory .............................................. 249
8.2 Installing the databases ......................................................................................... 250
8.3 Configuring WebSphere Application Server 3.02 .............................................. 252
    8.3.1 Using XMLConfig tool ......................................................................................... 252
    8.3.2 Manual configuration .......................................................................................... 253
    8.3.3 Install the Java classes and JSP files ................................................................. 259
    8.3.4 Install the XML and XSL files .............................................................................. 259
8.4 Running the application ......................................................................................... 261

Chapter 9. XMLapp: installing XML Collections version in WAS ............. 263
9.1 The files contained in SG246130 Collections directory ................................... 263
9.2 Installing the databases ......................................................................................... 264
9.3 Running the application ......................................................................................... 268
9.4 Application logging ................................................................................................. 268

Appendix A. XMLapp: installing XML Columns version in VAJ .............. 269
A.1 The content of the zip file ...................................................................................... 269
A.2 Installing the databases .......................................................................................... 270
A.3 Configuring VisualAge for Java 3.02 Enterprise Edition .................................... 271
    A.3.1 Creation of a new project ..................................................................................... 272
    A.3.2 Defining a datasource ......................................................................................... 273
    A.3.3 Configuring multiple Web applications ............................................................. 274
A.4 Completing the configuration of the xmlapp application............................... 276
A.5 Running the application .......................................................................................... 277
A.6 Accessing your IBM WebSphere Test Environment from another PC ............ 277

Appendix B. XML extender diagnostic trace ..................................................... 279
B.1 Start tracing: .......................................................................................................... 279
B.2 Stop tracing .............................................................................................................. 279
Figures

1. The Team: Mikio, Christophe, Luis and Sunil xvi
2. A simple CSS stylesheet 21
3. SAX application components 26
4. DOM interface hierarchy 30
5. Sample DOM tree 30
6. The DB2 XML Extender Product Selection screen 37
7. Choose Destination Location 38
8. XML Extender Administration Logon to a local DB2 server 40
9. Switching from database with the GUI Administration Wizard 41
10. XML Extender Administration Logon to a remote DB2 server 42
11. XML Extender Wizard: enabling a database 43
12. Person.xml illustration 45
13. GUI XML Extender: Work with XML Columns 47
14. GUI XML Extender: Add an XML Column (1 of 2) 48
15. GUI XML Extender: Add an XML Column (2 of 2) 49
16. GUI XML Extender: importing a DTD (1 of 2) 51
17. GUI XML Extender: importing a DTD (2 of 2) 51
18. GUI XML Extender: editing a DAD file 54
19. GUI XML Extender: creating a new DAD file (1 of 4) 55
20. GUI XML Extender: creating a new DAD file (2 of 4) 55
21. GUI XML Extender: creating a new DAD file (3 of 4) 56
22. GUI XML Extender: creating a new DAD File (4 of 4) 57
23. GUI XML Extender: enabling an XML Column 60
24. GUI XML Extender: disabling an XML Column (1 of 2) 66
25. GUI XML Extender: disabling an XML Column (2 of 2) 67
26. Person1.dtd 76
27. GUI XML Extender: Select a task 77
28. Specify a DAD 78
29. Select Validation 78
30. Specify text 79
31. Specify SQL Statement 80
32. Sample Map SQL window 81
33. Map SQL window 83
34. Persona.dtd 90
35. Syntax for XML expression 92
36. Admin Launchpad 94
37. Enable Collection 94
38. Collection details 95
39. XML document for decomposing 98
40. Admin Launchpad 100
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>84. List of all the users</td>
<td>217</td>
</tr>
<tr>
<td>85. Search results</td>
<td>224</td>
</tr>
<tr>
<td>86. Order basket</td>
<td>227</td>
</tr>
<tr>
<td>87. Order details</td>
<td>241</td>
</tr>
<tr>
<td>88. Supplier orders</td>
<td>243</td>
</tr>
<tr>
<td>89. Supplier order details</td>
<td>245</td>
</tr>
<tr>
<td>90. Web application definition</td>
<td>254</td>
</tr>
<tr>
<td>91. DAXSL servlet definition</td>
<td>255</td>
</tr>
<tr>
<td>92. DAXSL properties</td>
<td>255</td>
</tr>
<tr>
<td>93. MIME filter definition</td>
<td>256</td>
</tr>
<tr>
<td>94. Installing the JSP enabler</td>
<td>257</td>
</tr>
<tr>
<td>95. Directory view for XMLapp</td>
<td>260</td>
</tr>
<tr>
<td>96. Directory view of XMLExtApp</td>
<td>266</td>
</tr>
<tr>
<td>97. Administrative console</td>
<td>267</td>
</tr>
<tr>
<td>98. SERunner and ServletEngine</td>
<td>275</td>
</tr>
<tr>
<td>99. Screen shot of welcome page</td>
<td>283</td>
</tr>
<tr>
<td>100. Creation of dxxServlet</td>
<td>286</td>
</tr>
<tr>
<td>101. Screen shot of resulting page</td>
<td>288</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>DTD symbols</td>
</tr>
<tr>
<td>2</td>
<td>Attribute types</td>
</tr>
<tr>
<td>3</td>
<td>Default Value for Attributes</td>
</tr>
<tr>
<td>4</td>
<td>Columns created in the side tables</td>
</tr>
<tr>
<td>5</td>
<td>Extracting UDFs summary</td>
</tr>
<tr>
<td>6</td>
<td>Customer</td>
</tr>
<tr>
<td>7</td>
<td>Customer_Details</td>
</tr>
<tr>
<td>8</td>
<td>Mapping XML Document to Database</td>
</tr>
<tr>
<td>9</td>
<td>DB2 Text Extender commands</td>
</tr>
<tr>
<td>10</td>
<td>Transformation from UCS-2 to UTF-8</td>
</tr>
<tr>
<td>11</td>
<td>Chapters covering XMLapp</td>
</tr>
<tr>
<td>12</td>
<td>Application modules</td>
</tr>
<tr>
<td>13</td>
<td>Products and versions used</td>
</tr>
<tr>
<td>14</td>
<td>SG246130 Columns directory</td>
</tr>
<tr>
<td>15</td>
<td>Defining servlets used in xmlapp Web application</td>
</tr>
<tr>
<td>16</td>
<td>Products and Versions used</td>
</tr>
<tr>
<td>17</td>
<td>SG246130 collections directory</td>
</tr>
<tr>
<td>18</td>
<td>Products and versions used</td>
</tr>
<tr>
<td>19</td>
<td>SG246130 VAJ directory</td>
</tr>
<tr>
<td>20</td>
<td>Parameters for dxxServlet</td>
</tr>
<tr>
<td>21</td>
<td>Supported languages and code sets by DB2 UDB</td>
</tr>
<tr>
<td>22</td>
<td>Contents of SG246130.ZIP</td>
</tr>
</tbody>
</table>
Preface

This IBM Redbook shows how to use XML technology efficiently in business applications, and explains how to integrate it with DB2 Universal Database, DB2 XML Extender and Text Extender, and WebSphere Application Server. This book will help developers to set up the environment and to create and process XML documents that can be stored and recovered using SQL.

In Part 1, we introduce the theory of XML, including some advantages of its use. We show its positioning in real-life situations and explain how to process XML using Java. Following this, we cover DB2 XML Extender, from its installation to enabling/disabling the database for XML. Then we cover DB2 Text Extender, also from its installation to the actual use of data. Finally, we offer some considerations on Codepage and National Language Support.

In Part 2, we describe our XMLApp application, including two different implementations using XML Columns and XML Collections. We explain how to install both implementations in WebSphere Application Server, and how to install one of them in VisualAge for Java.

Anyone wanting to use XML technology to build better business solutions will benefit from reading this book.

Note: This book was written from the perspective of the Windows NT operating system, and some commands shown may not be available on other systems. However, all sections in this book, except some operating system specific commands and operations, should be applicable for database systems on non-NT operating systems such as OS/390, Solaris, and AIX.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization San Jose Center.

Luis Ennser is a consultant with the International Technical Support Organization, San Jose Center. He holds a bachelor's degree in Mechanical Engineering from IME-Instituto Militar de Engenharia, Rio de Janeiro, Brazil. Luis has 16 years of experience in Application Development and Networking. He writes extensively and teaches IBM classes worldwide on all areas of XML, WebSphere, and Java. Before joining the ITSO, he worked as an e-business Solution Architect in Brazil.
Christophe Delporte is a Software Engineer with Information Technology Services in IBM Belgium/Luxembourg. He has three years of experience with DB2 and programming languages. He holds a degree in mining engineering and geology from the Polytechnic Faculty of Mons in Belgium. His areas of expertise include e-business technologies such as XML, Java, WebSphere, DB2 and the AS/400 platform.

Mikio Oba is an I/T Engineer with ISE (IBM Japan Systems Engineering Co., Ltd.), providing technical support for DB2 UDB. He has two years of experience in DB2 UDB on UNIX and Windows platforms. He holds a master's degree in Engineering (Aeronautics) from the University of Tokyo. His areas of expertise include DB2 UDB and XML.

K. M. Sunil is a Software Engineer with IBM Global Services, India. He has two years of experience in application development on AIX and Windows platforms. He holds a degree in Information Science and Engineering from Bangalore University. His areas of expertise include RDBMS, Java, C, and C++.

Figure 1. The Team: Mikio, Christophe, Luis and Sunil
Thanks to the following people for their invaluable contributions to this project:

**IBM Silicon Valley Laboratory, San Jose, California:**
Susan Malaika
John Funderburk
Albert Maier
Dung Nguyen
Reto Preisig (visiting scientist from UBS AG, Switzerland)
Angel Reyda
Thomas Sharkey
Singgh Tanumhardjo
Gary Wilmot
Julie Wood
Rita Wu
Jane Xu

**IBM Germany:**
Michael Haide
Robert Kern
Juergen Metter

**IBM International Technical Support Organization, San Jose, California:**
Emma Jacobs
Yvonne Lyon
Claudia Traver

---

**Comments welcome**

**Your comments are important to us!**

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 321 to the fax number shown on the form.
- Use the online evaluation form found at [ibm.com/redbooks]
- Send your comments in an Internet note to redbook@us.ibm.com
Part 1. The theory of XML
Integrating XML with DB2 XML Extender and DB2 Text Extender
Chapter 1. Introduction to XML

This chapter provides an introduction to XML. We give a brief overview of XML, explain its business and technological benefits, and discuss basic XML concepts. At the end of this chapter we look at some examples of XML applications.

1.1 What is XML?

XML stands for eXtensible Markup Language. XML is a meta-markup language and is used for creating your own markup languages. Using XML, you can define the tags for your markup language. XML tags are used to describe the contents of the document. This means that any type of data can be defined easily using XML.

XML is universal not only by its range of applications but also by its ease of use: Its text-based nature makes it easy to create tools, and it is also an open, license-free, cross-platform standard, which means anyone can create, develop, and use tools for XML. What also makes XML universal is its power. Data is transmitted and stored in computers in many different ways: Originally it was stored in flat-files, with fixed-length or delimited formats, and then it moved into databases, and often into complex binary formats. XML is a structured data format, which allows it to store complex data, whether it is originally textual, binary, or object-oriented. To this day, very few data-driven technologies have managed to address all these different aspects in one package — except XML.

A brief comparison of XML and HTML

XML and HTML both descend from common roots. However, because XML is still new, most data on the Web is stored in HTML format. The amount of data currently stored on the Web is hard to imagine — for the latest survey (by OCLC Research) see the following Web site:

http://wcp.oclc.org

This survey indicates that there are 300 million pages on the Web. In fact, the amount of data available may be much higher, because counting pages ignores the fact that one dynamically generated page can act as a gateway to a large database.

The problem with data available in HTML format is that it is formatted for people to view, and not for computers to use. HTML consists of a pre-defined set of tags, the purpose of which are known. This makes it a language that is easy to learn and accessible, but makes it hard to re-use the data.
This is where XML enters the picture. As its name indicates, XML is extensible, which means that you can define your own set of tags and make it possible for others (people or programs) to know and understand these tags. This makes XML much more flexible than HTML. In fact, because XML tags represent the logical structure of the data, they can be interpreted and used in various ways by different applications.

Much of the value of the Web comes from re-using data. For example, one of the great success stories of the Web are the search engines. They work on the basis of a universal communications method (HTTP), and a universal markup language (HTML), to catalog Web pages. However, search engines work on very limited information, because only a tiny part of an HTML document is designed to be used by a search engine. Imagine how much more powerful search engines could be if the data that they search was stored in a simple, structured, re-usable format.

1.2 XML business benefits

To date, XML has three main applications:

- Sharing of information between computer systems and between businesses
- Storage and transmission of information within a single application
- Content delivery — delivering information to users

The early usage of XML has been in the first two of these areas, where the benefits of XML are easiest to achieve.

1.2.1 Information sharing

The benefits of using XML to share information between computer systems and businesses are probably the greatest and easiest to achieve: XML allows businesses to define data formats in XML, and to easily build tools which read data, write data, and transform data between XML and other formats. This has allowed a number of businesses and industry consortiums to build standard XML data formats. Areas such as Electronic Data Interchange (EDI), inter-bank payments, supply-chain, trading, and document management are all the subject of ongoing XML-based standardization by industry consortiums.

By using XML, the standard can be published, extended, and easily used by new applications. Data transformation tools exist that can convert between existing formats and XML, and new tools are emerging. The ability to link
enterprise applications, known as Enterprise Application Integration (EAI), is a key focus for many companies, and has produced cost savings and increased revenue for many enterprise customers. In particular, many businesses aim to improve Customer Relationship Management (CRM) by creating a single logical view of each customer across multiple existing systems. XML is an important technology to create this single customer view.

Furthermore, because XML makes it easy to relate structure to content, XML subsets can be defined with specific industries or applications in mind. For example, XML has been used to define standard data formats for the banking industry. In the same manner, a standard could be developed specifically for flight booking systems, thereby allowing airlines to easily exchange information.

1.2.2 XML inside a single application

The business benefits of using XML within a single application are less compelling, but they are very easy to achieve, and so we have seen a number of applications that use XML internally. The benefits of this approach are speed of development and future extensibility.

XML is a very powerful and flexible language for describing the complexities of the real world. Because XML can describe complex data, it means that it can be a powerful tool to creating new applications. The extensibility of XML means that the application can grow and develop without major changes. Finally, when the application needs to connect to other applications, XML makes an excellent way of linking the application with others.

1.2.3 Content delivery

XML has a number of key benefits for content delivery. The main benefits are the ability to support different users and channels, and to build more efficient applications. Channels are information delivery mechanisms — for example, Digital TV, phone, Web, and multimedia kiosk. Supporting different channels is an important step in delivering e-business applications to customers through their chosen medium. XML is a key technology for this.

For example, a customer and a supplier both need to access the same on-line product catalogue. Although the information is the same, the visual emphasis will differ, depending on who the user is: The customer will be more interested in looking for information on functionality, pricing, and availability, while the supplier will want to have easy access to catalog maintenance and inventory information. All this information might be stored in a single XML document and be displayed differently by the application.
Using XML for content delivery has been limited by the availability of XML-enabled browsers. Microsoft Internet Explorer 5.0 was the first browser to support XML directly, and until a large majority of browsers are XML ready, many Web masters will not adopt XML. As people become more familiar with this technology, this particular strength of XML is likely to be exploited more often.

1.3 Technological benefits of XML

In order to see the technological benefits of XML, let us consider an example.

1.3.1 An example of using XML

Many libraries offer their catalogs over the Web. Usually, there is a simple Web form where you enter a title, name, or subject, and you are presented with some search results. If you wanted to search several libraries, you would need to go to each of their Web sites. It would be useful and convenient if there could be a single Web page which could search many libraries. To build that today would require extracting the data (title, author, ISBN, and so forth) from each search results page. However, each search results page is formatted differently, and the data is mixed in with presentation information. To collate the results of many searches would require complex programming for each libraries Web site, separating the data from the presentation.

Suppose, instead, that there were a single format for returning search results from libraries: Let us call it A Book Catalog Markup Language or ABCML. ABCML would define tags for the author, title, and so on, thus making it easy for a computer to extract the data. Building the meta-catalog suddenly becomes easier.

ABCML would also help the libraries, because they could re-use each other's software. Also, when a new book came in, the publisher could provide the book's catalog information in ABCML, and save the librarian the effort of typing it into the catalog.

1.3.2 Major benefits

So ABCML would help us build a meta-catalog, and also help the libraries re-use existing data. Those arguments may not be enough to persuade a particular library's IT director to rewrite his online catalog. However, there are a number of further technical benefits to using XML. As well as re-using data, these benefits include: separating data from display, extensibility, and adding semantic content to the data.
1.3.2.1 Re-use of data
We have seen the benefit of re-using data: The librarian could re-use the publisher’s data, because it was in a common format, and we could re-use the data when we built our meta-catalog. Of course there are many common file formats in the world of computing that have allowed data re-use. These have usually been proprietary and application specific. XML is neither of those.

1.3.2.2 Separation of data and display
What are the benefits of separating data and presentation? First, without this separation, we could not achieve the simple re-use of the data. Second, the presentation changes. If you look at the Web sites of 5 years ago, and the Web sites of today, they are radically different. If you look at any successful Web site, you will probably see at least one redesign a year. That is not simply because it is trendy — successful Web sites analyze and react to feedback from users, and redesign the site to be more productive and intuitive. Let us return to the library Web site — the Web site gets a redesign, but the underlying data remains in place — so it makes sense to separate the data output from the Web site design.

There is still another even more compelling reason to separate data and display: the rise of pervasive computing. Pervasive computing means that computing devices become integrated into common everyday appliances: mobile phones, televisions, printers, and palm computers. Each of these appliances may have a different display technology, and require different instructions on how to display the data. The same search of the library catalog should be viewable on a mobile phone or a high-resolution PC.

1.3.2.3 Extensibility
HTML has been a constantly evolving standard since it emerged, and one of the problems it has faced is that it has often been extended by companies wishing to go beyond HTML. Browser suppliers regularly add non-standard extensions to HTML. Similarly, Web server manufacturers build “server-side” extensions to HTML: These include NCSA includes, Microsoft Active Server Pages, Java Server Pages, and many others. This has led to many confusing variants of the HTML standard, causing difficulties for Web developers, tool vendors, and ultimately for end-users.

As the name implies, eXtensible Markup Language was designed from the beginning to allow extensions. If we go back to our example of the library, when they first indexed books, the Web did not exist. Probably the library catalog has no references to Web sites in it. Nowadays, many books have a companion Web site, and the librarian may wish to reference it. If XML were
used to develop the catalog, then this could easily be accomplished. Importantly, with XML, old software is not disrupted by the addition of new information.

1.3.2.4 Semantic information
The final major benefit of XML is that it builds semantic information into the document. Semantic information (or meaning) is what allows readers to decide whether the book is about the color Brown, or written by Brown. An HTML-based Web search engine cannot do that, because it cannot distinguish between the title and author in the document — there isn’t enough semantic information in the HTML page. With XML, the document includes self-descriptive terms that show the meaning of the document.

1.3.2.5 Other benefits
The other main benefits of XML are that it is human-readable, tree-based, and easy to program. As time goes on, a large number of XML tools are emerging from both existing software vendors and XML startup companies. It is human-readable, because it is based on text and simple tagging. This means that it can be understood by humans, and it can be written using nothing more than a text-editor or word-processor. This is important in the sense that programmers can interpret the data faster when writing new applications, but once they start running, no one reads the data (only the applications). The tree-based structure of XML is much more powerful than fixed-length data formats. Because objects are tree structures as well, XML is ideally suited to working with object-oriented programming. In particular, many people believe that there is an excellent affinity between Java and XML.

Finally, XML is easy to program, because there are already standards for XML parsers. XML parsers are the subsystems that read the XML and allow programmers to work with XML. Because XML parsers are available to be re-used in new computer systems, many programmers are starting to use XML, even if they do not need any of the previously mentioned benefits.

1.4 XML concepts
In this section we try to cover a few basic concepts of XML, however this is not intended to be an XML reference manual. Since most people are familiar with HTML, we often make comparisons between HTML and XML. While people’s familiarity with HTML will hopefully simplify their task in understanding XML, it is also important to emphasize the differences between the two languages.
1.4.1 Document validity and well-formedness

XML is a metalanguage, which means that it is a language for describing markup languages. This is done by defining a set of tags for each markup language. XML does not predefine any tags. It allows you to create your own tags. However the process of defining tags and creating documents using the tag is not arbitrary. Therefore, there are a few rules that XML tags and documents should adhere to, in order to ensure that they are usable by any XML application.

XML has tighter constraints as compared to HTML, which tolerate minor structural irregularities in the documents they are parsing, such as unclosed tags. A well formed XML document should start with an XML declaration and should have a root element which contains all other elements. XML parsers will not accept documents that contain start tags, such as `<AUTHOR>`, without their corresponding end tags, in this example `</AUTHOR>`. This differs from HTML, which can be parsed even without any explicit end tags. On the other hand, XML does accept empty tags such as `<AUTHOR/>`. Well-formedness constraints also deal with attribute names, which should be unique within an element, and attribute values, which must not contain the character “<”. A document is said to be well-formed when it conforms to these constraints, which are referred to as the well-formedness constraints (WFC) defined in the XML 1.0 Recommendation (refer to http://www.w3.org/XML/ for more information).

The notion of validity applies to XML documents which have a Document Type Definition (DTD) associated with them. A Document Type Definition specifies the structure of the XML document by providing with a list of elements, attributes, notations and entities that a document can contain. When an XML document has a DTD associated with it a validating parser will read the DTD and checks whether the document adheres to the rules specified in the DTD. For example, for a document to be valid, all tags and attributes appearing in the document must have corresponding declarations in the DTD, and all elements appearing within other elements must respect the content model defined in the DTD.

It is worth noting that validity and well-formedness are two different aspects of an XML document. While well-formedness insures that XML parsers will be able to read the document, validity determines whether an XML document adheres to a DTD or schema. An XML application will check for and reject documents that are not well-formed before checking whether these documents comply with its validity constraints (VC). After a system is tested, validity checking can be turned off to improve performance.
1.4.2 Document type definition

In this section we briefly describe what a document type definition (DTD) is.

1.4.2.1 What is a DTD?
Since it is not the intent of this book to serve as an XML reference manual, we will not describe all the syntax elements of a DTD here. However, it is essential to understand the purpose and use of a DTD, and that is what we will focus on in this section.

The DTD specifies the structure of an XML document, thereby allowing XML parsers to understand and interpret the document's contents. The DTD contains the list of tags which are allowed within the XML document and their types and attributes. More specifically, the DTD defines how elements relate to one another within the document's tree structure, and specifies which attributes may be used with which elements. Therefore, it also constrains the element types that can be included in the document and determines its conformance: An XML document which conforms to its DTD is said to be valid.

A DTD can be either be stored in a separate file or embedded within the same XML file. XML documents referencing a DTD will contain the <!DOCTYPE> declaration which either contains the entire DTD declaration, or specifies the location of an external DTD, as shown in the following example:

```xml
<!DOCTYPE LibraryCatalogue SYSTEM "library.dtd">
```

An XML document is not required to have a DTD. However, with most applications, it will prove beneficial or even necessary to build a DTD which conveys efficiently the meaning behind the XML file's contents. DTDs provide parsers with clear instructions on what to check for when they are determining the validity of an XML document.

Having the logical definition of an XML file stored separately allows for the resulting DTD to be shared across organizations, industries, or the Web. When building XML applications, it is probably a good idea to look for existing DTDs that might suit your purpose. At the time of writing, well-recognized, universal standards have yet to emerge as more current industry initiatives are still in the drafting stages. However, as XML becomes more popular, more commercially-oriented or industry-oriented applications will likely appear and standards will emerge.
For more information on the latest emerging XML standards, the following sites may prove a good starting point:

http://www.schema.net
http://www.oasis-open.org

1.4.2.2 DTD example

The DTD has its own syntax, but is similar to XML in that it also uses markup tags. The following sample shows a simple internal DTD:

```xml
<?xml version = "1.0"?>
<!DOCTYPE authors [
  <!ELEMENT authors(author)+>
  <!ELEMENT author(firstname, lastname, title)>
  <!ELEMENT firstname(#PCDATA)>
  <!ELEMENT lastname(#PCDATA)>
  <!ELEMENT title(#PCDATA)>
]>...
[ insert XML data here]...
```

In the above example, the DOCTYPE statement represents the Document Type Declaration and the statements included within the square brackets make up the Document Type Definition. Both terms share the same acronym (DTD), which can be confusing, but it is usually clear from the context which of the two meanings is being referred to.

A well-formed XML document must contain a root element. The DOCTYPE name specified in the declaration must match that root element, in this case authors:

```xml
<!DOCTYPE authors [ ...
  <!ELEMENT authors(author)> ]>
```

The second line constitutes an element declaration, and the * indicates that the authors element can contain one or more author elements, which in turn are declared like this:

```xml
<!ELEMENT author(firstname, lastname, title)>
```

Similarly, the author element contains several elements: firstname, lastname, title. However, only one instance of each is allowed in this case:

```xml
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname(#PCDATA)>
<!ELEMENT title(#PCDATA)>
```
These last three elements contain only text and are therefore defined as *parse character data* or *PCDATA*. The adjunction of the # character marks the PCDATA type as a reserved word, and it cannot be used for names created by the author.

As mentioned earlier in this chapter, the DTD can either be stored within the XML document which it validates, or in an external file. The following is an example of a Document Type Declaration specifying an external DTD:

```xml
<?xml version = "1.0"?>
<!DOCTYPE authors SYSTEM "authors.dtd">
```

The use of the SYSTEM keyword indicates to the parser that this is an external declaration and that the set of rules for this XML document can be found in the specified file.

### 1.4.2.3 What’s in a DTD?

A Document Type Definition can contain different types of declarations. A list of these different types follows:

- **Elements** constitute the basic building blocks of an XML file and are declared like this:

  ```xml
  <!ELEMENT elementName(allowed element contents)>
  ```

  Example:

  ```xml
  <!ELEMENT greeting (#PCDATA)>
  <greeting>Hello, World!</greeting>
  ```

  Table 1 lists all the declaration attributes allowed inside an element declaration.
Attributes, as their name indicates, are attributes of an element which must be declared in the same DTD:

```xml
<!ATTLIST elementName attributeName attributeType attributeDefault>
```

Example:

```xml
<!ELEMENT BOOK(#PCDATA)>
<!ATTLIST BOOK
  ID ID #REQUIRED
  TYPE (Hardcover | Paperback) "Hardcover"
  STORELOC CDATA #FIXED "5th Avenue"
  COMMENT CDATA #IMPLIED
```
Table 2 provides a description of the various attribute types which can be used in attribute declarations, and lists these attribute types.

**Table 2. Attribute types**

<table>
<thead>
<tr>
<th>Attribute type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDATA</td>
<td>Can contain any kind of character data.</td>
</tr>
<tr>
<td>ID</td>
<td>Must have unique values within the element. In the example below, TYPEID is of the ID type, and so requires unique values within the range of BOOK elements:</td>
</tr>
<tr>
<td></td>
<td>&lt;BOOK TYPEID=&quot;ch1&quot;&gt;See Spot Run&lt;/BOOK&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;BOOK TYPEID=&quot;ch2&quot;&gt;Jack and Jill&lt;/BOOK&gt;</td>
</tr>
<tr>
<td>IDREF</td>
<td>The value of an ID type attribute of an element in the document.</td>
</tr>
<tr>
<td>IDREFS</td>
<td>Multiple ID's of elements separated by whitespace.</td>
</tr>
<tr>
<td>(enumerated)</td>
<td>Attributes can have a specified list of acceptable values.</td>
</tr>
<tr>
<td>ENTITY</td>
<td>The name of an entity declared in the DTD.</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>The attribute is optional.</td>
</tr>
<tr>
<td>NMTOKEN</td>
<td>The attribute is fixed (the syntax is of the type &quot;#FIXED Value&quot;).</td>
</tr>
<tr>
<td>NOTATION</td>
<td>The name of a notation declared in the DTD.</td>
</tr>
<tr>
<td>NMTOKENS</td>
<td>Multiple XML names separated by whitespace.</td>
</tr>
</tbody>
</table>

Table 3 lists all the default attribute values.

**Table 3. Default Value for Attributes**

<table>
<thead>
<tr>
<th>Attribute value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#REQUIRED</td>
<td>The attribute is required.</td>
</tr>
<tr>
<td>#IMPLIED</td>
<td>The attribute is optional.</td>
</tr>
<tr>
<td>#FIXED</td>
<td>The attribute is fixed (the syntax is of the type &quot;#FIXED Value&quot;).</td>
</tr>
</tbody>
</table>

- **Entities** are used to represent one or more characters in an XML document and act as constants, the value of which can be changed without the need to edit corresponding XML documents:

```
<!ENTITY entityName "character string represented">
```
Example:

```xml
<!ENTITY prodstyler "ACME Calendar">
```

(XML file:)
Thank you for choosing &prodstyler; as your primary scheduling program

(rendered:)
Thank you for choosing ACME Calendar as your primary scheduling program

- **Parameter entities** are entities which are used within the DTD itself. Their declaration differs by the inclusion of the `%` character:

```xml
<!ENTITY % entityName "character string represented">
```

Example:

```xml
<!ENTITY % commonAtts
    "ID ID #REQUIRED
    MAKE CDATS #IMPLIED
    MODEL CDATA #IMPLIED">

<!ELEMENT CAR (#PCDATA)>
<!ATTLIST CAR %commonAtts>

<!ELEMENT COMPUTER (#PCDATA)>
<!ATTLIST COMPUTER %commonAtts>
```

- **Notations** are used to refer to data from an outside (non-XML) source. They provide a basic means by which non-textual information can be handled within a document:

```xml
<!NOTATION name ExternalID>
```

Example:

```xml
<!NOTATION jpeg SYSTEM "jpeg.exe">
<!NOTATION gif SYSTEM "gif.exe">

<!ELEMENT person (#PCDATA)>
<!ATTLIST person
    picformat NOTATION (jpeg | gif) #REQUIRED>
```

(XML file:)
<person picformat="jpeg">Kelly Brown</person>
• **Comments** can be inserted inside the DTD by using the following notation:

<!-- insert comment text here -->

**Example:**

<!-- XML Comments Example -->
<!-- Author: Christophe Chuvan -->
<!-- Last Modified Date: 05/10/99 -->

### 1.4.3 Namespaces

Namespaces are useful when there is a need for elements and attributes of the same name to take on a different meaning depending on the context in which they are used.

For instance, a tag called `<TITLE>` takes on a different meaning, depending on whether it is applied to a person or a book. If both entities (a person and a book) need to be defined in the same document, for example, in a library entry which associates a book with its author, we need some mechanism to distinguish between the two and apply the correct semantic description to the `<TITLE>` tag whenever it is used in the document. Namespaces provide the mechanism that allows us to write XML documents which contain information relevant to many software modules. Consider this example:

```xml
<?xml version="1.0"?>
<library-entry xmlns:authr="authors.dtd"
               xmlns:bk="books.dtd">
  <bk:book>
    <bk:title>XML Sample</bk:title>
    <bk:pages>210</bk:pages>
    <bk:isbn>1-868640-34-2</bk:isbn>
    <authr:author>
      <authr:firstname>Joe</authr:firstname>
      <authr:lastname>Bloggs</authr:lastname>
      <authr:title>Mr</authr:title>
    </authr:author>
  </bk:book>
</library-entry>
```

As we can see, the `<TITLE>` tag is used twice, but in a different context, once within the `<AUTHOR>` element and once within the `<BOOK>` element. Note the use of the `xmlns` keyword in the namespace declaration. Interestingly, the XML Recommendation does not specify whether a namespace declaration should point to a valid URI (Uniform Resource Identifier), only that it should be unique and persistent.
In the previous example, in order to illustrate the relationship of each element to a given namespace, we chose to specify the relevant namespace prefix before each element. However, it is assumed that once a prefix is applied to an element name, it applies to all descendants of that element unless it is over-ridden by another prefix. The extent to which a namespace prefix applies to elements in a document is defined as the namespace scope. If we were to use scoping, the above example would then look like this:

```xml
<?xml version"1.0"?>
<library-entry xmlns:authr=\"authors.dtd\"
               xmlns:bk=\"books.dtd\">
  <bk:book>
    <title>XML & WebSphere</title>
    <pages>210</pages>
    <isbn>1-868640-34-2</isbn>
    <authr:author>
      <firstname>Joe</firstname>
      <lastname>Bloggs</lastname>
      <title>Mr</title>
    </authr:author>
  </bk:book>
</library-entry>
```

In this example, it is clear that all elements within the `<BOOK>` element are associated with the `bk` namespace, except for the elements within the `<AUTHOR>` element which belong to the `authr` namespace.

### 1.4.4 DTD versus XML schemas

The DTD provides a relatively easy-to-use way to describe the syntax and semantics of XML documents. However, to achieve this simplicity, a compromise was made when porting DTD support over from SGML to XML, which resulted in the expected simplification, but also in limitations that prevented the DTD from performing a high degree of semantic checking.

For example, a DTD allows for limited conditional checking by specifying allowed values, but there is no support for more complex semantic rules. For instance, it is currently impossible to check that an element which should contain a date actually does contain a date. There are also limitations when it comes to defining complex relationships between data elements and their usage, especially when XML documents also use namespaces which might define elements conflicting with DTD declarations.
Therefore, there is a need for a way to specify more complex semantic rules and provide type-checking within an XML document. XML schemas, while still in the drafting stage, aim to provide such functionality and also introduce new semantic capabilities such as support for namespaces and type-checking.

Whether XML schemas provide a viable alternative to the DTD or come in the form of extensions to the DTDs functionality, they will need to provide a high degree of compatibility with existing structures. Also, because DTD support is part of the current XML Recommendation, the schema designers cannot overlook the fact that many applications and data structures will have been developed with the DTD in mind. For more information on XML schemas, refer to the specification documents from the W3C:

XML Schema Part 1: Structures
http://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes
http://www.w3.org/TR/xmlschema-2/

1.5 XPath

The name XPath comes from its use as notations in URIs for navigating through XML documents. The aim of an XPath is to address parts of an XML document. XPath answers the needs common to both XPointer and XSL transformations. XPath uses a compact syntax, and it operates on the logical structure underlying XML to facilitate usage of XPath within URIs and XML attribute values. XPath supports XML namespaces because XPath models an XML document as a tree of nodes (root nodes, element nodes, attribute nodes, text nodes, namespace nodes, processing instruction nodes, and comment nodes). The basic syntactic construct in XPath is the expression. An object is obtained by evaluating an expression, which has one of the following four basic types:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean
- Number
- String

XPath uses path notation to define locations within a document. The paths starting with a “/” signifies an absolute path. A simple example of this follows.
Let us consider an XML document (Library.xml) that describes a Library System. This document will be used for XPath and XPointer examples.

```xml
<?xml version="1.0"?>
<!DOCTYPE LIBRARY SYSTEM "library.dtd">
<LIBRARY>
    <BOOK ID="B1.1">
        <TITLE>xml</TITLE>
        <COPIES>5</COPIES>
    </BOOK>
    <BOOK ID="B2.1">
        <TITLE>WebSphere</TITLE>
        <COPIES>10</COPIES>
    </BOOK>
    <BOOK ID="B3.2">
        <TITLE>great novel</TITLE>
        <COPIES>10</COPIES>
    </BOOK>
    <BOOK ID="B5.5">
        <TITLE>good story</TITLE>
        <COPIES>10</COPIES>
    </BOOK>
</LIBRARY>
```

The path `/child::book/child::copies` selects all copies element children of book which are defined under the document's root. The above path can also be written as `/book/copies`.

The XPath location step makes the selection of document part based on the basis and the predicate. The basis performs a selection based on Axis Name and Node Test. Then the predicate performs additional selections based on the outcome of the selection from the basis. A simple example of this is as follows:

The path `/child::book[position()-1]` selects the first book element under root. This location step can also be written as `/book[1]` if you wish.

For example, the path `/book/author/@*` would have selected all the author elements' attributes.

The path `/book/author[@type='old']` would have selected all the author elements with type attribute equal to "old".
1.6 eXtensible stylesheet language (XSL)

XSL is the language defined by the W3C to add formatting information to XML data. Stylesheets allow data to be formatted based on the structure of the document, so one stylesheet can be used with many similar XML documents.

XSL is based on two existing standards, Cascading Style Sheets (CSS) and Document Style Semantics and Specification Language (DSSSL).

CSS is the stylesheet language for HTML 4.0, and as such, is well supported in Web design tools, such as IBM WebSphere Studio. XSL is mainly based on CSS, and so a short description of CSS is provided below. CSS can also be used as a formatting language for XML, but it is less powerful than XSL. Because CSS was designed for the Web, it is excellent for defining the presentation of data for Web browsers.

XML aims to support any possible display, and printed output has a number of challenges that browsers do not face. DSSSL is the stylesheet language of SGML and has mainly been used for printed output. Therefore, elements of DSSSL that go beyond CSS have been incorporated into XSL. More information on DSSSL can be found at the following Web address:

http://www.jclark.com/dsssl/

1.6.1 Cascading Style Sheets

Cascading Style Sheets were designed to help separate presentation from data with HTML. The reason that they are called Cascading Style Sheets is because HTML, like XML, has a tree structure, and styles which are applied to the root of a tree cascade down to the branches and leaves. CSS allows the Web developer to define styles that apply:

- To any given type of element (for example, all paragraphs)
- To a class of elements (for example, all paragraphs which contain code samples)
- To a particular element (for example, the third paragraph)

This is achieved by specifying classes and ids in the HTML, and applying styles to them.

A very simple stylesheet is presented in Figure 2. This stylesheet defines a standard font and colors for all text in the BODY of the HTML file. It defines a specific class of text which is twice the normal size, bold and capitalized (.largeClass), and finally it specifies that a particular element labelled THISID should be displayed in fuschia-colored cursive text.
The benefits of CSS are well-understood: Web developers can easily change the layout and presentation of a whole site by editing a single stylesheet. CSS can be used with XML if the display engine supports it. So far, the only browser being shipped that supports this feature is Microsoft Internet Explorer 5.0.

```css
BODY{
    font-family : Verdana,sans-serif;
    font-weight : normal;
    color : black;
    background-color : white;
    text-decoration : none;
}
.largeClass{
    font-size : 200%;
    font-weight : bolder;
    text-transform : capitalize;
}
#THISID{
    font-family : cursive;
    color : fuchsia;
}
```

*Figure 2. A simple CSS stylesheet*

CSS can be used within a document, or referenced in a separate stylesheet, which is the more common approach. For more information on CSS, see: http://www.w3.org/Style/CSS

### 1.6.2 XSL = fo: + XSLT

Although XSL has derived much from CSS, the approach of XSL is much more powerful, and has major differences from CSS. XSL is defined by a working draft of the W3C at the following URL:

http://www.w3.org/TR/WD-xsl

XSL actually consists of two different standards, the *transformation language*, and the *formatting objects*.

The transformation language is called XSLT, and is defined as a working draft of the W3C at the following URL:

http://www.w3.org/TR/WD-xslt
XSLT defines a common language for transforming one XML document into another. It defines how to create a result tree from a source tree.

The formatting objects (FO) define how to display the result tree. This is the part of XSL which is most closely related to CSS. Formatting objects are referred to as fo: in the XSL code. The main difference between CSS and XSL FO is that XSL is based on an XML format, properly defined with a DTD, and CSS is not.

Because XSL defines an extra step in presenting data, it can do much more powerful presentation tasks than CSS can. CSS always retains the order of the source tree, whereas XSL can re-order the data. A simple example might be where two stylesheets can be used to display the same data, one ordered by name, and the other ordered by location.

At the time of writing, there are only experimental implementations of XSL formatting objects, although as XML browsers become more advanced, this functionality is expected to become mainstream. Because XSL formatting objects are still mainly theoretical, we chose not to cover them in detail in this book.

So, if XSL FO is not implemented, how can you actually display XML? The most common way of doing this is to use an XSL stylesheet that transforms the XML source document into an HTML result document. Then, any HTML display engine can be used to display the result tree. Similarly, alternative stylesheets can be built that transform the source tree into other directly displayable markup languages: for example, Wireless Markup Language for mobile phones, and Precision Graphics Markup Language for printed output.

There is also a method of specifying alternate stylesheets for one document, and with the correct software these can be chosen based on the display engine, so an Internet Explorer user will see a different result than a Nokia mobile phone user might see.

Most XSL implementations that target HTML result in a combination of HTML and CSS. This breaks up the presentation of XML into two stages: Structure and layout are controlled by the XSLT and resulting HTML, and the visual style and formatting are controlled by the CSS stylesheet.

An important point is that XSL does not require a DTD for either the source or result document. This means that it can be flexibly applied in many situations which would be impossible otherwise.
1.6.3 XSL transformations

XSLT works on two principles: pattern matching and templates. Remember that XSLT transforms a source tree into a result tree. The XSLT processor takes two inputs; the XSL stylesheet and the source tree; and produces one output — the result tree. The stylesheet associate patterns with templates. The patterns identify parts of the source tree. When a match is made, the associated template is applied.

XSL stylesheets can also filter the input data, based on logical tests, and re-order the data. The templates can also contain arbitrary tags, formatting instructions or data. The result of this is that an XSL stylesheet can perform very powerful transformations on the input data. For example, a stylesheet could be used to create an HTML display of a list of bank accounts, sorted by balance, with overdrawn accounts colored red, and large balances colored green. The same data could be used with another stylesheet which graphically represented the data by transforming it into Structured Vector Graphics (SVG), which is an XML format for drawing graphics.

1.7 Real-life uses of XML

In this section we present two examples of how XML is being used in real-life to bring benefits to people, businesses, and organizations. For more examples, see the XML section in the IBM developerWorks site at:

http://www.ibm.com/developer/xml

Or, see the XML.org catalog site at:

http://www.xml.org/xmlorg_registry/index.shtml

1.7.1 Resource description frameWork (RDF)

The Resource Description Framework (RDF) is an XML application used to encode metadata which can be used to describe any internet resource such as a web site and its content. An RDF description also referred to as metadata can include the author of the resource, the date of creation, key word, information about the content, and so on. Metadata has many uses on the Web, including organizing, searching, filtering, and personalizing Web sites. In order for metadata to have these benefits, however, Websites, search engines, and directories must agree to use a standard format for metadata. The Resource Description Framework is a W3C-recommended XML application for encoding, exchanging, and re-using structured metadata.
1.7.2 SABRE and Wireless Markup Language

The SABRE Group is one of the major distributors of international travel services, offering electronic travel bookings through travel agents and on the Web worldwide. They are transforming their travel information into XML using a Java application, and then allowing mobile phone users worldwide to look up, reserve, and purchase travel from a mobile phone. The XML is automatically translated from XML into Wireless Markup Language, which is a standard for building applications on mobile phones. The benefits of XML to this application are its extensibility, the speed of development, and the ability to build a standard repository in XML, and translate as needed into a particular environment, in this case the mobile phone.

1.8 Processing XML using Java

In a perfect world, computer applications would just exchange XML documents. In real life, applications often have to be able to support multiple client types, all with different capabilities. The dominant client type for Web application servers is currently a browser (usually Netscape or Internet Explorer), but it will not be like that forever. We might have cellular phones and other front-end devices, all with different XML capabilities.

The most recent edition of any browser that might have XML support cannot be a prerequisite for using an XML based Web application. We also do not want to send the same XML document to every client, because some users of the application might be authorized to see more data than others. We must have the ability to process XML documents and generate the kind of response to the client that is adequate for the client type.

On the server side, the Web application server usually connects to a back-end data store like a relational database that does not natively support data interchange using XML. We need to be able to extract the necessary information from an XML document and pass that information to the database, as well as transform the information coming from the database to XML. To fulfill both the client and the server requirements, we need an XML processor.

While the XML document format is the most natural form of data exchange in the internet, Java is the most natural language to be used in Internet applications and application servers. This is because of Java’s nature: object-oriented and distributed.
One technical advantage of Java over other languages is its built-in support for Unicode. With other languages, XML processing has to be done using tricks or by developing additional libraries to support Unicode in that language environment. IBM does have a C++ implementation of an XML parser, as well as supporting Unicode libraries, but, to summarize, Java is an excellent language in implementing XML processors and other XML related tools and applications.

1.8.1 XML applications

At the heart of every XML application is an XML processor that parses the well-formed XML document, so that the document elements can be retrieved and transformed into data that can be understood by the application and task in hand. The other responsibility of the parser is to check the syntax and structure (validity and well-formedness) of the document.

Anyone has the freedom to implement a parser that can read and print an XML document. The XML 1.0 Recommendation defines how an XML processor should behave when reading and printing a document, but the API to be used is not defined. However, there are standards that define how XML documents should be accessed and manipulated. Currently, the following two APIs used are widely used:

- Simple API for XML
- Document Object Model

1.8.2 SAX

Simple API for XML (SAX) was developed by David Megginson and a number of people on the xml-dev mailing list on the Web, because a need was recognized for simple, common way of processing XML documents. As such, SAX 1.0 is not a W3C recommendation, but it is the de-facto standard for interfacing with an XML parser, with many commonly available Java parsers supporting it.

SAX is an event-driven lightweight API for accessing XML documents and extracting information from them. It cannot be used to manipulate the internal structures of XML documents. As the document is parsed, the application using SAX receives information about the various parsing events. The logical structure of an application using SAX API with the parser is shown in Figure 3.
The SAX driver can be implemented by the XML parser vendor, or as an add-on to the parser. That makes the application using the parser via SAX independent of the parser.

1.8.2.1 SAX classes and interfaces
The SAX 1.0 API defines two sets of interfaces, one of which is meant to be implemented by XML parsers, and one by XML applications. The interfaces that parsers have to implement are:

- Parser
- AttributeList
- Locator (optional)

The first thing an XML application has to do is to register SAX event handlers to a parser object that implements the `Parser` interface. As the XML document is processed by a parser, SAX notifies the application whenever an event occurs. The events that can be captured depend on the registered event handlers, the interfaces of which are:

- DocumentHandler
- DTDHandler
- ErrorHandler

The most important and commonly used interface is DocumentHandler, because it can be used to track basic document-related events like the start and end of elements and character data. The events occur in the order that is directly related to the order of elements that are found in the tree-formed XML document that is being parsed.
DTDHandler notifies the application about unparsed external entity declarations or when a notation declaration is encountered. ErrorHandler notifies the application whenever an error occurs while parsing the XML document.

The SAX specification also provides a HandlerBase class, which implements all interfaces and provides default behavior. Instead of implementing the appropriate interfaces, an XML application can extend the HandlerBase class and override just the methods that need to be customized.

The Java implementation of SAX is organized in two Java packages:

- `org.xml.sax`
- `org.xml.sax.helpers`

The first of the above-mentioned packages contains the SAX core implementation classes, interfaces and exceptions. The second one contains convenience classes and a Java-specific class (ParserFactory) for dynamically loading SAX parsers.

The implementation can be downloaded from [http://www.megginson.com/SAX/](http://www.megginson.com/SAX/). The same location also contains full descriptions (in JavaDoc format) of all classes and interfaces defined in SAX 1.0.

### 1.8.2.2 SAX example

For a Java application to be able to use SAX, we need a class that implements an interface most suitable for the job. The following code fragment shows the relevant methods of DocumentHandler that are implemented to track start and end of elements and the whole document. It also prints out the actual data within the elements:

```java
public class MyDocHandler implements org.xml.sax.DocumentHandler {
    ...
    public void characters(char[] arg1, int start, int length) throws org.xml.sax.SAXException {
        System.out.println(new String(arg1, arg2, arg3));
    }
    public void startDocument() throws org.xml.sax.SAXException {
        System.out.println("Start of document");
    }
    public void endDocument() throws org.xml.sax.SAXException {
        System.out.println("End of document");
    }
    public void startElement(String name, org.xml.sax.AttributeList arg2)
        throws org.xml.sax.SAXException {
        System.out.println("Start of element "+ name);
    }
}
```
public void endElement(String name) throws org.xml.sax.SAXException {
    System.out.println("End of element " + name);
}

The application that uses the DocumentHandler implementation above is simple. IBM's XML for Java implements the Parser interface in SAXParser class, which the following example uses:

...  
Parser parser = ParserFactory.makeParser("com.ibm.xml.parsers.SAXParser");  
SampleDocumentHandler hndlr = new SampleDocumentHandler();  
parser.setDocumentHandler(hndlr);  
parser.parse(anXMLFileURL);  
...

Given the following XML document as input:

```xml
<?xml version="1.0"?>
<personnel>
    <person id="jedi1">
        <name>
            <lastname>Skywalker</lastname>
            <firstname>Luke</firstname>
        </name>
    </person>
</personnel>
```

The output of the SAX application looks like this:

Start of document
Start of element personnel
Start of element person
Start of element name
Skywalker
End of element lastname
Start of element firstname
Luke
End of element firstname
End of element name
End of element person
End of element personnel
End of document
1.8.3 DOM

While XML is a language to describe tree-structured data, the Document Object Model (DOM) defines a set of interfaces to access tree-structured XML documents. DOM specifies how XML and HTML documents can be represented as objects. Unlike SAX, DOM also allows creating and manipulating the contents of XML documents. Basically, the DOM interfaces are platform and language neutral. The current W3C DOM Level 1 Specification provides bindings for Java, OMG IDL, and ECMAScript (formerly JavaScript).

DOM originated from the need to dynamically render HTML content (DHTML). The current DOM Level 1 Recommendation has two parts: Core and HTML. Core contains fourteen interfaces, seven of which are applicable to both HTML and XML documents. Six remaining interfaces are specific to XML. DOM HTML defines additional convenience methods that are useful for client side scripting.

1.8.3.1 DOM hierarchy

The DOM API is a set of interfaces that must be implemented by a DOM implementation such as IBM's XML for Java. The interfaces, being originally described in IDL, form a hierarchy (see Figure 4).

The root of the inheritance tree is Node, that defines the necessary methods to navigate and manipulate the tree-structure of XML documents. The methods include getting, deleting, and modifying the children of a node, as well as inserting new children to it. Document represents the whole documents, and the interface define methods for creating elements, attributes, comments, and so on. Attributes of a Node are manipulated using the methods of the Element interface. DocumentFragment allows extracting parts of a document.

Note that while a DOM application reads an XML document and an object representation if formed, that representation remains only in memory. Changing a DOM object in memory does not automatically modify the original file. That is something an application program has to do for itself.
The W3C DOM Level 1 Recommendation can be found at:
http://www.w3.org/DOM/

1.8.3.2 DOM example
When the simple XML document we used in our SAX example (see listing on page 28) is processed using DOM, the resulting object tree will look like the one in Figure 5. The shaded rectangles represent character data, and the others represent elements.
Reading an XML document using DOM is relatively easy, provided that a good parser is available. Among other things, IBM’s XML Parser for Java provides a robust and very complete implementation of the W3C DOM API. The following code fragment shows a simplified example of how to read and manipulate an XML document using the DOMParser class:

```java
DOMParser parser = new DOMParser();
parser.parse(uri);
Document document = parser.getDocument();
print(document); // implemented in our own code
Node n = document.getLastChild().getFirstChild();
n.setNodeValue("ZAP! You're history!");
print(document);
```

1.8.3.3 DOM Level 2
Some important facilities that are missing from the DOM Level 1 Recommendation are being defined in DOM Level 2, which is currently a W3C Specification working draft (July 19, 1999). The added functionality in Level 2 contains interfaces for creating a document, importing a node from one document to another, supporting XML Namespaces, associating stylesheets with a document, the Cascading Style Sheets object model, the Range object model, filters, and iterators, and the Events object model.

1.8.4 SAX or DOM?
There are certainly applications that could use either SAX or DOM to get the necessary functionality needed when processing XML documents. However, these two approaches to XML processing each have their strengths and weaknesses.

1.8.4.1 SAX advantages and disadvantages
SAX provides a standardized and commonly used interface to XML parsers. It is ideal for processing large documents whose content and structure does not need to be changed. Because the parser only tells about the events that the application is interested in, the application is typically small, and has a small memory footprint. This also means that SAX is fast and efficient, and a good choice for application areas such as filtering and searching, where only certain elements are extracted from a possibly very large document.

Because the events must be handled as they occur, it is impossible for a SAX application, for example, to traverse backwards in the document that is under processing. It is also beyond SAX’s capabilities to create or modify the contents and internal structure of an XML document.
1.8.4.2 DOM advantages and disadvantages

Because every element of an XML document is represented as a DOM object to the application using the DOM API, it is possible to make modifications to the original XML document. Deleting a DOM node means deleting the corresponding XML element and so on. This makes DOM a good choice for XML applications that want to manipulate XML documents, or to create new ones.

DOM is not originally an event driven API like SAX, even though the DOM Level 2 draft specifies events. To extract even a small piece of data from an XML document, the whole DOM tree has to be created. There is no way of creating lightweight applications using DOM. If the original XML document is large, the DOM application that manipulates the document requires a lot of memory. In practice, DOM is mostly used only when creating or manipulating XML documents is a requirement.
Chapter 2. DB2 XML Extender

In this chapter we provide a detailed description of IBM DB2 XML Extender, which helps you integrate the power of IBM DB2 Universal Database (DB2 UDB) with the flexibility of XML.

2.1 Introduction

DB2 XML Extender provides the ability to store and access XML documents, and also to compose XML documents from the existing relational data or to decompose XML documents into relational data. It does so by providing a set of new data types, functions, and stored procedures. With DB2 XML Extender, you can use SQL as the main access method to your XML documents, which makes your task much easier. It also facilitates your Administration tasks by providing you with a GUI-based wizard, a set of administrative stored procedures and an Administration command.

Even though XML solves many of the problems by providing a standard format for data interchange, there are other problems, such as storing the XML documents in a centralized repository, as well as the ability to quickly search for information or to trigger automatic data interchange when a particular action occurs. These kinds of issues can be addressed only by a database management system.

This is where DB2 XML Extender can help you. For example, the following applications can benefit from using DB2 XML Extender:

**Electronic Data Interchange (EDI) applications:** These applications are typically Business-to-Business (B2B) applications which interchange data primarily using XML. However, due to the different ways these applications use XML data, some might need to keep the native XML formatted documents, while others might prefer to map the data into relational tables. In the former case, the whole XML document can be stored in the database and searches on known elements or attributes can be performed. In the latter case, XML documents can be mapped to columns in the relational database.

**Web Information Retrieval Applications:** These applications are Business-to-Consumer (B2C) applications that are often used in interactive Web sites. The XML documents are usually not large in size, but have structured information. In some cases, it is required to store entire XML documents into database and use SQL to do fast search on desired XML elements or attributes with rich data types. Range search for rich data types is often important in applications that retrieve information interactively. In
other cases, application builders want to draw data from existing business tables, form XML documents, and provide them on the web site for viewing. In both cases, the XML documents are generally in read-only format.

**XML Content Manager:** Applications that need to provide advanced content management functions to the user can use DB2 XML Extender as their underlying physical storage and provide fast search with indexing. XML Extender can provide efficient management for large documents by partitioning the XML documents into multiple pieces and doing updates in place. Applications for content management can provide XML repository services using DB2 XML Extender. These applications can be built on top of object-oriented databases (OODBs) as well.

### 2.1.1 Storage and access methods

With DB2 XML Extender you can either store the entire XML document in DB2 as a user-defined data type or map the XML content into columns in DB2 tables. DB2 XML Extender can also store the XML document as an external file in the file system with a pointer to that file stored in the DB2 database so that you can keep working with your legacy flat files but benefit from the advantages offered by DB2 at the same time. DB2 XML Extender provides powerful capabilities to search the XML elements and attributes which can also be combined with the structural and full text search capabilities of DB2 Text Extender.

DB2 XML Extender provides two options you can use for integrating XML documents into DB2:

- **XML Columns:** This method allows you to store XML documents in DB2. The documents are inserted into columns which are enabled for XML, and can be updated, retrieved, and searched. Element and attribute data can be mapped to DB2 tables called *side tables*, which can then be indexed for fast searches.

- **XML Collections:** This method allows you to map XML document structures to DB2 tables, so that you can compose XML documents from existing DB2 data, or decompose XML documents into DB2 data.

With the XML Columns method, once the document is stored, you have a whole range of possibilities for working with the XML document:

- You can *extract* and *search* XML elements and attributes in the XML document directly or for improved performances you can store in the *side tables* the elements and attributes you are likely to access more often. DB2 Text Extender can be used to extend your search capabilities to perform structural and full text search into the XML document.
You have the option to update the content of an XML element or the value of an XML attribute.

The XML document can also by retrieved and used by your XML application.

With the XML Collections method, you have these possibilities:

- You can populate the columns of one or more DB2 tables starting from an XML document. The tags in the XML document and the XML document itself is not stored in DB2; only the values of the elements and attributes you are interested in are stored in DB2. The mapping between elements and attributes in the XML document and the DB2 tables is given by the DAD file.

- You can create an XML document using the data in columns of one or more DB2 tables. DB2 XML Extender composes an XML document using the data stored in columns of DB2 tables. Here again, the mapping between data in the DB2 tables and the composed XML document is done using the DAD file.

- You can work with the data shredded (or decomposed) from the XML document as you would work with any relational data in DB2.

2.1.2 Document Type Definition (DTD) repository

DB2 XML Extender provides an XML DTD repository. When a database is enabled for XML, a DTD reference table called DTD_REF is created. Each row in this table contains a DTD with additional metadata information about it. You can insert your own DTDs into this table. The DTDs in this table are used to validate the XML documents.

2.1.3 Document Access Definition

DB2 XML Extender also provides you with a mapping scheme called Document Access Definition (DAD), which is a file used to map the XML document to relational data. The DAD is an XML formatted document which allows you to associate XML document structure to a DB2 database when using either XML Columns or XML Collections. The structure of DAD files is different when using an XML Column or an XML Collection.

DAD files are managed using the XML_USAGE table that is created when the database is enabled.
2.1.4 Location path

A location path is a sequence of XML tags separated by a forward slash (/) that identifies an XML element or attribute. Location paths are used in the following situations within DB2 XML Extender and DB2 Text Extender:

- They are given as input to extracting UDFs to identify elements and attributes to be extracted.
- They are used 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.
- They are used by the Text Extender for structural-text search.

The following is the location path syntax supported by DB2 XML Extender:

- /: Represents the XML root element.
- /tag1: Represents the element tag1 under root.
- /tag1/tag2/.../tagn: Represents an element with the name tagn as the child of the descending chain from root, tag1, tag2, through tagn-1.
- //tagn: Represents any element with the name tagn, where double slashes (//) denote zero or more arbitrary tags.
- /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.
- /tag1/tag2/@attr1: Represents the attribute attr1 of an element with the name tag2, which is a child of element tag1 under root.
- /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.
- /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.

Now that the major concepts have been covered, we continue by explaining how to install DB2 XML Extender; then we describe its use in detail.

2.2 Installation

DB2 XML Extender Version V7.1 requires DB2 Universal Database V6.1 or higher and is available on AIX, Sun Solaris, Linux, Windows NT and Windows 2000. DB2 XML Extender contains the following components: XML Extender.
Server, XML Extender run-time client, XML Extender software developer's kit (SDK), samples and online documentation.

To install DB2 XML Extender on Windows NT, you need to follow these steps:

1. Load the DB2 XML Extender CD.
2. Read the readme file under D:\db2xml\yourPlatform, where D:\ is the drive used for the CD and your platform is either AIX, NT or Solaris, depending on your platform.
3. To install DB2 XML Extender on Windows NT, execute the setup.exe file located in the directory D:\db2xml\nt.
4. Choose the desired setup language.
5. Select the product you want to install: DB2 XML Extender Server or DB2 XML Extender Client. Select **XML Extender Server** if you are installing DB2 XML Extender on a server platform. This option also installs the **XML Extender Client** component. Select **XML Extender Client** if you are installing DB2 XML Extender in a client environment. Figure 6 shows the Product Selection screen.

![Product Selection](image)

*Figure 6. The DB2 XML Extender Product Selection screen*

6. Select one of the three installation types: Typical, Compact, or Custom. A Typical installation is recommended, unless you do not have enough space on your machine.
7. Leave the default destination location \C:\dxx as shown in Figure 7, or choose another folder to install DB2 XML Extender.

![Choose Destination Location](image)

*Figure 7. Choose Destination Location*

8. Leave the default Program Folder DB2 XML Extender or choose another one.

9. Shut down and restart the system once the selected components have been installed.

When the installation is complete and the workstation restarted, you should check the installation. To check the installation, you must create a database bind it to XML Extender and CLI and run the installation check program.

1. Create a database. You can use the DB2 Control Center or use the `CREATE DATABASE MYDB` command in the Command Center or a Command Line Processor window, where MYDB is the name of the database you want to use for XML Extender.

2. After creating the database, bind it as follows:
   a. In a DB2 Command Window, connect to the database by entering: `db2 connect to mydb`
   b. Change to the BND subdirectory in the directory where you installed XML Extender. For example, if you installed XML Extender in C:\DXX, change to C:\DXX\BND with the command: `cd \d C:\DXX\BND`
c. Bind the database to XML Extender by entering: `db2 bind @dxxbind.lst`

d. Change to the BND sub directory in the directory where you installed DB2 with the command: `cd /d %DB2PATH%\bnd`

e. Bind the database to CLI by entering: `db2 bind @db2cli.lst`

f. Disconnect from the database by entering: `db2 terminate`

3. After binding the database, change back to the base directory where you installed XML Extender; for example, `C:\DXX` by typing: `cd /d C:\DXX`

4. Run the installation check program, dxxinstallchk.cmd by entering:

   `dxxinstallchk mydb > out 2>&1` where `mydb` is your database name and `out` is the name of an output file. The output should be similar to the content of the verification file, dxxinstallchk.vfy. For instance, with the user `xmlext` and the database `xmltest`, we get the following output file:

   ```
   DXXA002I Connecting to database xmltest.
   DXXA005I Enabling database xmltest. Please wait.
   DXXA006I The database xmltest was enabled successfully.
   Database Connection Information
   Database server = DB2/NT 7.1.0
   SQL authorization ID = XMLEXT
   Local database alias = XMLTEST
   DB20000I The SQL command completed successfully.
   DXXA002I Connecting to database xmltest.
   DXXA005I Enabling column Order. Please Wait.
   DXXA022I Column Order enabled.
   DXXA002I Connecting to database xmltest.
   DXXA034I XML Extender has successfully disabled column Order.
   DB20000I The SQL command completed successfully.
   DXXA002I Connecting to database xmltest.
   DXXA035I XML Extender is disabling database xmltest. Please wait.
   DXXA036I XML Extender has successfully disabled database xmltest.
   DB20000I The TERMINATE command completed successfully.
   ```

Now that the DB2 XML Extender has been successfully installed, let’s see how we can make the most of it in the next section.

The last part of the installation is the GUI Administration Wizard of DB2 XML Extender. This GUI Administration tool is a Java wizard that can be launched from the DB2 Control Center and allows you to:

- Connect to the database
- Enable or disable a database for XML Extender
- Enable or disable an XML Column
• Enable or disable an XML Collection
• Create or edit a DAD file

It requires you to have a IBM JDK and the JFC Swing classes on our computer. These classes are automatically provided with DB2 and the easiest way is probably to use them. Therefore, you should add in your classpath environment variable the two files: C:\SQLLIB\java\jdk\lib\classes.zip (containing the IBM JDK1.1.7) and C:\SQLLIB\java\swingall.jar (containing the Swing classes), where C:\SQLLIB in the installation directory of DB2.

Once the classpath has been adapted, we can start the GUI Wizard by selecting Start > Programs > DB2 XML Extender > XML Extender Administration Wizard. If the logon screen appears, as indicated in Figure 8, the installation and setup of the Wizard are successful.

Note: To go further, you need to logon to a DB2 Server, either local or remote, by filling its URL, the JDBC Driver you want to use, a userid, and a password. If the DB2 Server is local or has been cataloged with the CAE, you do not need to mention the host and port in the address, and can use the JDBC driver COM.ibm.db2.jdbc.app.DB2Driver.

Figure 8. XML Extender Administration Logon to a local DB2 server
Once you are connected to one DB2 server, you can switch to another one, as shown in Figure 9. You must have bound this other database to DB2 XML Extender and to DB2 CLI.

Figure 9. Switching from database with the GUI Administration Wizard
You can also connect to a remote DB2 server by mentioning the host name and port, the default being 6789. The driver to use in this case is the JDBC network driver COM.ibm.jdbc.db2.net.DB2DRIVER as indicated in Figure 10.

Before you can actually work with XML documents in our database, you must enable the database to XML. This is described in the next section.

### 2.3 Enabling the database to XML

In this section and in the following ones, many of the tasks we complete can be done with either:

- The Administration command
- The Administration Wizard
- The Administration stored procedures

We systematically describe the Administration commands, often illustrated with the GUI Wizard, and only indicate which stored procedure performs the same function. You can refer to the *IBM DB2 Universal Database XML Extender Administration and Programming*, for more details.

The XML enablement of the database is the first step that is required to store XML information in a database. Enabling a database creates:
The db2xml schema (also assigns the needed privileges).

- The user-defined types (UDT) XMLVARCHAR, XMLCLOB, XMLFILE, and many user-defined functions (UDF).

- The necessary control tables required by DB2 XML Extender, for example, DTD_REF and XML_USAGE.

The Administration command `dxxadm` provided by DB2 XML Extender has various command options, depending on the action you want to execute. To enable a database, the command is: `dxxadm enable_db mydb`, where `mydb` is the name of the database you want to enable.

The GUI Administration Wizard can also be used to enable a database for XML, as shown in Figure 11.

The stored procedure `dxxEnableDB()` fulfills the same function.

![XML Extender Administration LaunchPad](image)

**Figure 11. XML Extender Wizard: enabling a database**

After having enabled our database, you can work with XML documents, either using the XML Columns method, or the XML Collections method. Ways to use these methods are described in the following sections.
2.4 XML Columns method

Using the XML Columns method allows you to store the entire XML document, as it is, in a column.

We recommend choosing the XML Columns method if one or more of the following criteria can be met:

- The XML documents already exist — for example, you want to archive documents such as newspaper articles, orders, and so on.
- The XML documents are read-often and update-rarely.
- The performance of the update is not critical.
- You want to store the intact XML documents.
- You want to keep the XML documents externally from DB2 on local file systems.
- You know what elements or attributes will be frequently searched. To perform efficient searches on these documents, you can decide to create indexes in side tables on the elements or attributes that you need to access more often.

We first present the XML document we will use for illustration purposes throughout this section.

2.4.1 XML example used as illustration

In this section, which discusses the XML Columns method, and in the next section, which discusses the XML Collections method, we illustrate with a very simple XML example. This sample XML document, named person.xml (as rendered by Internet Explorer 5) is shown in Figure 12.
The DTD that validates our sample document follows:

```xml
<?xml version="1.0" ?>
<!DOCTYPE person (View Source for full doctype... )>
  - <person id="1">
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <country>US</country>
  - <phone>
    <type>office</type>
    <number>12345678</number>
  </phone>
  - <phone>
    <type>home</type>
    <number>34567890</number>
  </phone>
</person>
```

Later in this section, we illustrate our description of the functionality of XML Columns by applying them to this sample XML document, person.xml.
2.4.2 Table creation

Once the database has been enabled for XML, the next step in storing an XML document in an XML Column is to decide in which table you will write your XML document. You can create a new table with an XML Column or just alter an existing table to add an XML Column.

DB2 XML Extender provides you with three new user-defined types located in the db2xml schema to store your XML documents as column data:

- **XMLVarchar**: You can store an XML document in the database, with a maximum size of 3 KB.
- **XMLCLOB**: The XML document is also stored in the database, but its maximum size is 2 GB.
- **XMLFILE**: This UDT allows you to keep the document on the local file.

For instance, the following SQL statement would create a table, person1, containing one XML Column of type XMLCLOB:

```sql
create table person1(id integer not null primary key, person db2xml.xmlclob not logged);
```

If you prefer to add an XML Column to an existing table, you could alter this table. The following statement adds an XML Column of type xmlvarchar to the table, person2:

```sql
alter table person2 add person db2xml.xmlvarchar;
```

The GUI XML Administration Wizard also offers you the possibility to add an XML Column to an existing table.
The first step in working with the GUI XML Extender Administration Wizard, after being connected to the DB2 Server, is to click the **Work with XML Columns** button, as shown in Figure 13.

![GUI XML Extender: Work with XML Columns](image)

*Figure 13. GUI XML Extender: Work with XML Columns*
Now select **Add an XML Column** and click the **Next** button, as shown in Figure 14.

*Figure 14. GUI XML Extender: Add an XML Column (1 of 2)*
On the next display, as shown in Figure 15, you can select the table you want to alter, the type of XML Column you want to add, and its name. Click the **Finish** button to alter the table.

Figure 15. GUI XML Extender: Add an XML Column (2 of 2)

A pop-up window confirms that the XML Column was successfully added.

### 2.4.3 DAD creation

The Document Access Definition file, or DAD file, itself an XML document, specifies how the XML documents that you store in the database are to be handled.

In the case of XML Columns, the DAD file is only needed if you want to validate your XML documents before storing them, or if you want to index elements or attributes in side tables. The side tables are additional tables created by DB2 XML Extender to improve performance when searching elements or attributes in an XML Column.

Before creating the DAD file, you should:

- Decide which elements or attributes you often want to search in your XML documents
- For each element or attribute that you want to index in a side table, define:
- The location path to represent it: Use the XPath data model to map XML structure (the element and attribute) to the relational tables (the columns).

- The data type your element or attribute should be converted to. In your XML documents, all your elements content and attributes value are considered character data. But in your side tables, you can use any DB2 data types.

- Consider whether they have multiple occurrences or not. This must match the declaration in the DTD validating your XML documents.

- For elements and attributes which are not multiple occurring but are at the same level of the XML document structure, you can extract them into columns in the same side table.

- For each multiple occurring element or attribute, you need to create a new side table if you want to extract their value for indexing.

- Decide whether you want the validation to occur or not. This decision can be based on the following considerations:
  - The validation has a small performance impact
  - You may not want to validate XML documents that you know are valid
  - The validation by DB2 XML Extender can only occur at the time the XML documents are stored into the XML table and not afterwards.

Before you can validate an XML document, you need to register your DTD within DB2 XML Extender. To do that, you must store your DTD file in the DTD_REF table created in DB2XML schema during the enablement of the database and associate this file with an unique identifier: the DTD ID. You have two possibilities to insert a DTD in the DTD_REF table:

- Use an INSERT SQL statement to insert the DTD file as a XMLCLOB data type in the DTD_REF with the user-defined function XMLClobFromFile provided by DB2 XML Extender. (This function will be discussed in more detail in 2.4.5.1, “The storage UDFs” on page 61.)

  For example, if you want to insert the DTD, person.dtd, located on your workstation in the directory, C:\SG246130\code, and associate this DTD with the unique identifier: C:\SG246130\code\person.dtd, the SQL statement would be:

```sql
insert into db2xml.dtd_ref(dtdid, content, usage_count, author, creator, updater) values('C:\SG246130\code\person.dtd',
    db2xml.XMLClobFromFile('C:\SG246130\code\person.dtd'),0, 'xmlext',
    'xmlext', 'xmlext');
```
• Use the GUI XML Extender Administration Wizard. After being connected, you first need to click the **Insert a DTD** button, as shown in Figure 16.

![Figure 16. GUI XML Extender: importing a DTD (1 of 2)](image)

On the next screen, shown in Figure 17, you must enter the location of the DTD file and the DTD ID you want. Then click **Finish**.

![Figure 17. GUI XML Extender: importing a DTD (2 of 2)](image)
You receive a pop-up window confirming you that the DTD file was successfully added in the DTD_REF table.

Now that the DTD file has been registered, we can create the DAD file for our simple illustration. The coding is followed by explanatory notes:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\doc\dtd\dad.dtd">
<DAD>
  <dtid>C:\SG246130\code\person.dtd</dtid>
  <validation>YES</validation>
  <Xcolumn>
    <table name="person_names">
      <column name="fname" type="varchar(50)"
            path="/person/firstName"
            multi_occurrence="NO"/>
      <column name="lname" type="varchar(50)"
            path="/person/lastName"
            multi_occurrence="NO"/>
    </table>
    <table name="person_phone_number">
      <column name="pnumber" type="varchar(20)"
            path="/person/phone/number"
            multi_occurrence="YES"/>
    </table>
    <table name="person_phone_type">
      <column name="ptype" type="varchar(20)"
            path="/person/phone/type"
            multi_occurrence="YES"/>
    </table>
  </Xcolumn>
</DAD>
```

Notes:

1. A DAD file is an XML document.
A DAD file must conform to the DAD.DTD provided with DB2 XML Extender in the dtd subdirectory of its installation directory.

The DAD file starts and ends with a <DAD> tag.

A DAD file contains 3 tags within the root element: <dtdid>, <validation> and <Xcolumn> or <Xcollection>. The dtdid tag refers to the identifier of the DTD file stored in the DTD_REF table. It is only useful if we decide to validate the XML document.

We want to validate the XML document as indicated by the YES content of the validation element. If we do not want to validate the document, the value should be NO.

We are using the XML Columns method.

Specify a <table> tag for each side table you want to create.

For each side table, specify a <column> tag for each column you want the side table to contain. Each <column> has 4 attributes: name, path, type and multi_occurrence. The name attribute is the name of the column that is created in the side table.

The path attribute indicates the location path in the XML document for each element or attribute to be indexed. It conforms to the XPath notation.

The type attribute indicates the data type in the side table for each indexed element or attribute.

The multi_occurrence attributes indicates whether the element or attribute referred to by the path attribute can occur more than once in the XML document. The possible values are YES or NO. If the value is NO, than you can mention more than one <column> tag in the side table. Otherwise, you can only mention one <column> in a side table you shown in 12.

A DAD file can also be created or edited with the GUI XML Extender Administration Wizard.

The first step is to click **Edit DAD**, as shown in Figure 18.
Then you can either choose an existing DAD file that you want to edit, or leave the **File Name** field blank if you want to create a new DAD file. If you intend to create a new DAD file, you can also decide which method you want to use, either the XML Columns method, or the XML Collections method, as shown Figure 19.
After clicking the Next button, you decide if you want validation or not. If you do want validation, select the DTD ID, as indicated in Figure 20.

Figure 20. GUI XML Extender: creating a new DAD file (2 of 4)
After choosing the DTD ID, click **Next**. On the next screen, you define the tables and their column attributes as indicated in Figure 21.

![GUI XML Extender: creating a new DAD file (3 of 4)](image)

**Figure 21.** GUI XML Extender: creating a new DAD file (3 of 4)
After defining all the side tables you need, click the Next button. The last step consists in defining the name you want for this newly created DAD file, as shown in Figure 22.

Now click Finish and the DAD file is automatically created for you by the Wizard.

We have seen how we could create a DAD file, but there is more to be done. We must also link it to the XML Column created in our XML table. This what we do in the next section by enabling the XML Column.

2.4.4 Enabling the XML Column

Enabling an XML Column in a table is not mandatory. If you have not created a DAD file, you do not enable the XML Column, and you can still store an XML document in an XML Column. However, this document will not be validated, and no side table will be created.

If you created a DAD file for an XML Column, it is necessary that you tell DB2 XML Extender to which XML Column and in which table this file relates. When you enable an XML Column, DB2 XML Extender does the following:

- It parses the DAD file.
- It creates the side tables with the desired columns corresponding to the elements and attributes from the XML document.
- It creates the triggers on the user table or XML table containing the XML Column to synchronize with side tables.
- It adds a new entry in the db2xml.xml_usage table created during the XML enablement of the database. This new entry keeps the relation between the user table, the XML Column in this table, the DTD ID and the DAD file. This DAD file is stored as a CLOB in the XML_USAGE table.
If you decided to validate the XML document, the usage_count column in the dtd_ref table is increased for the concerned row. You cannot delete the entry in the dtd_ref table when the usage_count for this DTD is not zero.

As we only have one Administration command for DB2 XML Extender, the command to enable an XML Column remains dxxadm, but this time with the option enable_column. The required parameters are:

1. The name of the database
2. The name of the user table containing the XML Column
3. The name of the XML Column in this table
4. The DAD file to be associated with this XML Column

There are also some optional arguments:

- Tablespace (-t): the table space to store the side tables. You mention this option only if you want to put the side tables into a specified tablespace.
- Default_view (-v): the name of the default view joining the user table containing the XML Column and side tables, if you want one.
- Root_id (-r): the name of the single primary key in the user table that is added to the side tables. The root_id is the way to tie the side tables together with the user table. We recommend that you use your own defined primary key as root_id. But if it is not specified or if your application table has no primary key, the DXXROOT_ID column is added automatically to the user table containing the XML Column for storing a unique ID created at insertion time. All side tables will then have this same column DXXROOT_ID.

For our simple illustration, the command to enable the XML Column is:

dxxadm enable_column xmltest person1 person c:\SG246130\code\personColumn.dad -r id

As a consequence of this command, three side tables are created by DB2 XML Extender: person_names, person_phone_number, person_phone_type — as mentioned in the DAD file.
The columns created for these three tables are displayed in Table 4 and are explained in the notes following it.

Table 4. Columns created in the side tables

<table>
<thead>
<tr>
<th>person_names</th>
<th>person_phone_number</th>
<th>person_phone_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>fname</td>
<td>dxx_seqno</td>
<td>dxx_seqno</td>
</tr>
<tr>
<td>lname</td>
<td>pnumber</td>
<td>ptype</td>
</tr>
</tbody>
</table>

Notes:

1. The id column has been added as consequence of the root_id optional argument used with the `dxxadm enable_column` command. The value contained here is the value copied from the id column in the user table containing the XML Column (person1 in our illustration).

2. When the attributes or elements are multiply-occurring, a sequence number is automatically provided to identify each occurrence. This allows for joining the various side tables.

You can use the GUI XML Extender Administration Wizard rather than the Administration command to enable an XML Column.

The first step is to click the **Work with XML Columns** button as shown previously on Figure 13 on page 47.

The next screen offers you the choice to Enable an XML Column as previously shown in Figure 14 on page 48. Select it and click **Next**.
Your final step is to give the name of the user table, the XML Column, the DAD file, and any desired optional parameters, as indicated in Figure 23.

![GUI XML Extender: enabling an XML Column](image)

Press **Finish** when the required fields are filled in, and wait for the pop-up window confirming the enablement of the XML Column.

Note that the stored procedure `dxxEnableColumn` is also available if you want to enable an XML Column from inside a program.

### 2.4.5 Insert, delete, extract, and update functions

Now that you have created the DAD file and enabled the XML Column, you can start working with our XML documents. You can:

- Insert them in our user table
- Search them
- Update them
- Retrieve them

Corresponding to these four ways of treating the XML documents, DB2 XML Extender provides four types of user-defined functions (UDFs). For a detailed description concerning the syntax and arguments of these functions, refer to *IBM DB2 Universal Database XML Extender Administration and*
Programming. In the following subsections, we describe these functions and illustrate some of them using our simple example.

2.4.5.1 The storage UDFs

The storage UDFs to import XML data into DB2. We have four storage UDFs depending on the UDT in which we store the XML document:

- XMLVarcharFromFile(): Used to import an XML Document from a file in the file system to an XMLVarchar column in the user table
- XMLCLOBFromFile(): Used to import an XML Document from a file to an XMLCLOB column in the user table
- XMLFileFromVarchar(): Used to read an XML Document from memory as a VARCHAR, write it to a file system and insert the file name and path as an XMLFile type in an XML Column
- XMLFileFromCLOB(): Used to read an XML Document from memory in a CLOB format, write it to a file system, and insert the file name and path as an XMLFile type in an XML Column

In our illustration, we insert the XML document person.xml located in the directory C:\SG246130\code into the table, person1, whose column person was enabled in the previous section:

```sql
insert into person1
values(1, db2xml.xmlclobfromfile('c:\SG246130\code\person.xml'));
```

The insert is successful, meaning that the XML document person.xml was valid as we choose for validation in the DAD file used to enable the XML Column.

It is important to emphasize that DB2 XML Extender populates the side tables at the insertion time with the triggers added during the enablement of the column. This means that if your column already contained XML documents before being enabled for DB2 XML Extender, the content of these XML documents will not be reflected in the side tables.

2.4.5.2 The extraction UDFs

There are two ways to extract XML elements or attributes from an XML document stored in a DB2 column:

- Using the side tables is the easiest way if you specified in the DAD file that you wanted them to be placed in the side tables.

  In our illustration, for instance, we can find the office telephone number of Mr. Smith with the following SQL statement, using the information generated in the side tables:
```sql
select pnumber
from person_names t1, person_phone_number t2, person_phone_type t3
where t1.id = t2.id
and t1.id = t3.id
and t2.dxx_seqno = t3.dxx_seqno
and t1.lname='Smith'
and t3.ptype='office';
```

- If the elements or attributes are not in the side tables, then you must use the extracting UDFs that allow you to extract XML elements or attributes directly from an XML document stored in DB2 Column. These functions allow for search and read, depending whether the UDF is placed in the select or in the where clause of the select statement. All these extracting UDFs are based on the location path (XPath notation) to locate the desired element or attribute.

The extracting UDFs are divided into these groups:

- The scalar extracting UDFs: These allow you to find an element or attribute within an XML document. This element or attribute must have only one occurrence in the whole XML document. The UDFs return a scalar SQL data type.

- The table extracting UDFs: These give the possibility to find a multiple occurring element or attribute within an XML document and return a DB2 table having multiple rows of the considered SQL data type.

Each extracting UDF expects two input parameters:

- The XML document to be searched (XMLFile, XMLVarchar, or XMLCLOB)

- The location path expressed in XPath notation to identify the element or attribute

The extracting UDFs convert the string of an element or attribute in the XML document to one of the following SQL data types:

- CHAR
- VARCHAR
- CLOB
- INTEGER
- SMALLINT
- DOUBLE
- REAL
- DATE
- TIME
- TIMESTAMP
Table 5 summarizes the existing extracting functions. Refer to IBM DB2 Universal Database XML Extender Administration and Programming, for a more detailed description of each of these functions.

Table 5. Extracting UDFs summary

<table>
<thead>
<tr>
<th>Return type</th>
<th>Scalar UDF</th>
<th>Table UDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>extractInteger</td>
<td>extractIntegers</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>extractSmallint</td>
<td>extractSmallints</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>extractDouble</td>
<td>extractDoubles</td>
</tr>
<tr>
<td>REAL</td>
<td>extractReal</td>
<td>extractReals</td>
</tr>
<tr>
<td>CHAR</td>
<td>extractChar</td>
<td>extractChars</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>extractVarchar</td>
<td>extractVarchars</td>
</tr>
<tr>
<td>CLOB</td>
<td>extractCLOB</td>
<td>extractCLOBs</td>
</tr>
<tr>
<td>DATE</td>
<td>extractDate</td>
<td>extractDates</td>
</tr>
<tr>
<td>TIME</td>
<td>extractTime</td>
<td>extractTimes</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>extractTimestamp</td>
<td>extractTimestamps</td>
</tr>
</tbody>
</table>

In our illustration, we can for example extract the information from the <country> element which is not duplicated in any of the side tables. The following SQL statement extracts the country of Mr. Smith:

```
select db2xml.extractVarchar(person, '/person/country')
from person1 t1, person_names t2
where t1.id = t2.id
and t2.lname='Smith'
```

The extracting function can also be placed in the where clause. For example, the following coding could be used to search through all the XML documents to locate people whose country is ‘US’:

```
select lname, fname
from person1 t1, person_names t2
where t1.id = t2.id
and db2xml.extractVarchar(person, '/person/country') = 'US';
```

### 2.4.5.3 The update UDFs

There are two ways to update an XML document:

- Use the update SQL statement to replace the XML document by another one located on the file system:
update person1 set person =
db2xml.XMLCLOBFromFile('c:\SG246130\code\personbis.xml');

The side tables are immediately changed to stay in synchronization with the content of the XML document.

- Use the Update UDF function. This function allows you to replace the XML document stored in an XML Column by only changing an element or attribute value without needing the updated XML document on the file system. This function uses the location path to locate the attribute or element whose value must be changed.

For example, the following SQL statement changes the value of the country element of all XML documents stored in the XML Column, person, in the table, person1:

update person1 set person =
db2xml.Update(person,'/person/country','UK');

We strongly recommend that you do not update the side tables directly, because doing so will cause data inconsistency problems. You must update the original XML documents stored in the XML Columns.

### 2.4.5.4 The retrieval UDF

There are various ways of retrieving the XML document you stored in the database. If you stored it as an XMLCLOB or XMLVarchar, for example, you can cast it to a CLOB or VARCHAR type. These cast functions are provided in the db2xml schema.

Still, DB2 XML Extender provides the overloaded retrieval UDF Content that allows you to retrieve or export data from DB2. There are three implementations of this function, depending on the type of parameters:

- The Content UDF to export an XML document from an XMLVarchar format to an external file server
- The Content UDF to export an XML document from an XMLCLOB format to an external file server
- The Content UDF to export an XML document from an XMLFile format to a CLOB locator

For more details on this overloaded UDF, refer to *IBM DB2 Universal Database XML Extender Administration and Programming*.

The following SQL statement illustrates, on the basis of our small example, how to retrieve an XML document from a XMLCLOB column and how to export it on an external file system (in our case, on c:\SG246130):
select db2xml.Content(person, 'c:\SG246130\retrievedperson.xml') from
person1 where id = 1;

You can check on drive d: in the directory SG246130 and will find the XML
document retrievedperson.xml in this directory.

2.4.6 Disabling the XML Column

At the present time, disabling and re-enabling the XML Column in the only
way to change a DAD file. For example, suppose that you had originally
decided not to validate the XML document, but now you have changed your
mind, and you want the validation to take place. The developers of DB2 XML
Extender are aware of this problem and intend to develop a solution, so that
the XML Column need not be disabled and re-enabled for minor changes in
the DAD.

Another reason that you might want to disable the XML Column is simply to
return to the original configuration, because you no longer need to store XML
documents in that XML Column.

When you disable an enabled column, DB2 XML Extender takes the following
actions for you:

- It drops the previously created triggers for the specified column on the
  user table containing the XML Column.
- It drops the created side tables and all their content.
- It deletes the entry added in the XML_USAGE table.
- If you required validation to take place in the DAD file, the usage_count
column in the DTD_REF table is decreased of one unity for the concerned
  DTD ID.

To disable an XML Column, the command option to use with the dxxadm
Administration command is disable_column. The required parameters are:

1. The name of the database
2. The name of the user table containing the XML Column
3. The name of the XML Column in this table that you want to disable

In our simple illustration, the Administration command would be as follows to
disable the column, person, in the table, person1:

dxxadm disable_column xmltest person1 person
You can execute the same task with the DB2 XML Extender Administration Wizard. The first step is to click the **Work with XML Columns** button as previously shown in Figure 13 on page 47.

Then you select the option **Disable an XML Column** and click the **Next** button as indicated in Figure 24.

![Figure 24. GUI XML Extender: disabling an XML Column (1 of 2)](image)

Your next step is to select the table and the column that you want to disable, and click the **Finish** button, as shown in Figure 25.
A pop-up window appear, confirming you that the XML Column has been successfully disabled.

You also have the possibility to disable an XML Column by using the Administration stored procedure dxxDisableColumn if you want to incorporate this operation in one of your programs.

### 2.5 XML Collections method

Using the XML Collections method allows you to map XML document structures to DB2 tables, so that you can compose XML documents from existing DB2 data or decompose XML documents into DB2 tables.

We recommend using 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 are interested in 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.

2.5.1 Scenario to Illustrate composing an XML document

We are going to compose an XML document from existing DB2 data. The existing data for the XML document is shown in Table 6 and Table 7 below. The column names in italics are columns that are intended to be part of the XML document structure. All the data is stored in a database called XMLTEST, previously created to illustrate XML columns.

Table 6. Customer

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Country</td>
<td>VARCHAR(30)</td>
</tr>
<tr>
<td>LName</td>
<td>VARCHAR(30)</td>
</tr>
<tr>
<td>FName</td>
<td>VARCHAR(30)</td>
</tr>
</tbody>
</table>

Table 6 has been created using the following DDL statement:

```sql
CREATE TABLE Customer (ID INTEGER NOT NULL PRIMARY KEY, Country VARCHAR(30), LName VARCHAR(30), FName VARCHAR(30))
```

Table 7. Customer_Details

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Phone_Number</td>
<td>VARCHAR(20)</td>
</tr>
<tr>
<td>Phone_Type</td>
<td>VARCHAR(20)</td>
</tr>
</tbody>
</table>

Table 7 has been created using the following DDL statement:

```sql
CREATE TABLE Customer_Details (ID INTEGER, Phone_Number VARCHAR(20), Phone_Type VARCHAR(20), CONSTRAINT CUST_DETAIL_FK FOREIGN KEY (ID) REFERENCES Customer (ID))
```

Records were inserted into the Customer table using following SQL statements:

```sql
INSERT INTO Customer values(1,'US','Smith','Bill');
```
INSERT INTO Customer values(2,'INDIA','Gupta','Raj');

Records were inserted into the Customer_Details table using following SQL statements:

INSERT INTO Customer_Details values(1,'408-231-300','office')
INSERT INTO Customer_Details values(1,'408-200-301','home')
INSERT INTO Customer_Details values(2,'91-80-5262355','office')
INSERT INTO Customer_Details values(2,'91-80-527117','home')

We use the same DTD that we used in the scenario for the XML Columns method, as follows. The XML document we compose will have document structure as specified by this DTD:

```xml
<?xml version="1.0"?>
<!ELEMENT person (firstName,lastName,phone*)>
<!ATTLIST person id CDATA #REQUIRED >
<!ELEMENT firstName (#PCDATA )>
<!ELEMENT lastName (#PCDATA )>
<!ELEMENT country (#PCDATA )>
<!ELEMENT phone (type,number)> 
<!ELEMENT type (#PCDATA )>
<!ELEMENT number (#PCDATA )>
```

In Table 8 the DB2 column names are qualified by the table names. This mapping was used to create the DAD file shown in 2.5.2, “DAD with SQL_stmt” on page 69 that defines the relationship between relational data and the XML document structure.

<table>
<thead>
<tr>
<th>XML Element/Attribute</th>
<th>DB2 Column Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Customer.ID</td>
</tr>
<tr>
<td>firstName</td>
<td>Customer.FNAME</td>
</tr>
<tr>
<td>lastName</td>
<td>Customer.LNAME</td>
</tr>
<tr>
<td>country</td>
<td>Customer.COUNTRY</td>
</tr>
<tr>
<td>number</td>
<td>Customer.Phone_Number</td>
</tr>
<tr>
<td>type</td>
<td>Customer.Phone_Type</td>
</tr>
</tbody>
</table>

2.5.2 DAD with SQL_stmt

The DAD file used to compose the XML document with SQL mapping follows:
2.5.2.1 Preparing the DAD file
To create the DAD file shown previously:
1. Open a text editor and create a file, person.dad.

2. Create the DAD header:

   ```xml
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
   ```

3. Insert the `<DAD></DAD>` tag. This element will contain all the other elements.

4. Insert the `<validation></validation>` tag to indicate whether DB2 XML Extender validates the XML document using the DTD in the DTD repository table. In this example, you are not validating, so you would insert the following code:

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

   However, if you intended to validate, you would insert the following code.

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

5. You use the `<Xcollection></Xcollection>` tags to specify the access and storage method as XML collection.

6. You use the `<SQL_stmt></SQL_stmt>` to specify the SQL used for mapping the relational data to the XML document as follows. The `SQL_stmt` tag should be within the `Xcollection` tag.

   ```sql
   SELECT c.ID, COUNTRY, LNAME, FNAME, p.PHONE_ID, PHONE_NUMBER, PHONE_TYPE from
   CUSTOMER C, table (SELECT substr(char(timestamp(generate_unique())),16)
   as PHONE_ID, ID, PHONE_NUMBER, PHONE_TYPE from customer_details) P where
   P.ID=C.ID order by ID, PHONE_ID
   ```

Follow these guidelines to create the SQL statement for mapping the relational data to the XML document:

- Columns are specified in top-down order, by the hierarchy of the XML document structure.
- The columns for an entity are grouped together, and each group has an object ID column: ID, Phone_ID.
- The object ID column is the first column in each group. For example, C.ID precedes the columns related to the person element and P.PHONE_ID precedes columns for the phone element.
- The `CUSTOMER_DETAIL` table does not have a single key column, and therefore, the `generate_unique` DB2 built-in function is used to generate the `PHONE_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.

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

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

8. Add the `<doctype></doctype>` tag which will contain the path to the DTD against which the composed document will be validated:

```
<doctype>!DOCTYPE person SYSTEM "C:\dxx\samples\dtd\person.dtd"></doctype>
```

In this example we stored the DTD under C:\dxx\samples\dtd directory.

9. Add the `<root_node></root_node>` tag to define the root element. All the elements and attributes that make up the XML document are specified within the root_node.

10. Map the XML document structure to the DB2 relational table structure using the following three types of nodes:

```
  element_node
  Specifies that 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.
```

Follow these steps for each type of node. We use the DAD file in 2.5.2, “DAD with SQL_stmt” on page 69 to explain each step.

a. For each element in the XML document, define an `<element_node>` tag with the name attribute set to the element's name, as follows:

```
<root_node>
  <element_node name="person">
    <element_node name="firstName">
    </element_node>
    <element_node name="lastName">
    </element_node>
    <element_node name="phone" multi_occurrence="YES">
      <element_node name="type">
      </element_node>
      <element_node name="number">
      </element_node>
  </element_node>
</root_node>
```
b. For each attribute in the XML document define an <attribute_node> tag with the name attribute set to the attribute's name. The attributes are nested in their element_node. The example code follows:

```xml
<root_node>
  <element_node name="person">
    <attribute_node name="id">
    </attribute_node>
    <element_node name="firstName">
    </element_node>
  </element_node>
  <element_node name="lastName">
    <text_node>
    </text_node>
  </element_node>
  <element_node name="phone" multi_occurrence="YES">
    <element_node name="type">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="number">
      <text_node>
      </text_node>
    </element_node>
  </element_node>
</root_node>
```

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

```xml
<root_node>
  <element_node name="person">
    <attribute_node name="id">
    </attribute_node>
    <element_node name="firstName">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="lastName">
      <text_node>
      </text_node>
    </element_node>
    <element_node name="phone" multi_occurrence="YES">
      <element_node name="type">
        <text_node>
        </text_node>
      </element_node>
      <element_node name="number">
        <text_node>
        </text_node>
      </element_node>
    </element_node>
  </element_node>
</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. The code for this follows:
<root_node>
  <element_node name="person">
    <attribute_node name="id">
      <column name="ID"/>
    </attribute_node>
    <element_node name="firstName">
      <text_node>
        <column name="FName"/>
      </text_node>
    </element_node>
    <element_node name="lastName">
      <text_node>
        <column name="LName"/>
      </text_node>
    </element_node>
    <element_node name="phone" multi_occurrence="YES">
      <element_node name="type">
        <text_node>
          <column name="Phone_Type"/>
        </text_node>
      </element_node>
      <element_node name="number">
        <text_node>
          <column name="Phone_Number"/>
        </text_node>
      </element_node>
    </element_node>
  </element_node>
  </root_node>

11. Place an ending <root_node> tag after the last <element_node> tag.
12. Place an ending </Xcollection> tag after the </root_node> tag.
13. Place an ending </DAD> tag after the </Xcollection> tag.
14. Save the file as person.dad

2.5.2.2 Composing the XML document
To compose the XML document you will use the dxxGenXML() stored procedure. (For a detailed description of XML Extender stored procedures, please refer to IBM DB2 Universal Database XML Extender Administration and Programming.) There is a sample Java program shipped with XML Extender which can be used to compose the document.
To compose the document:

1. Compile the Java program Generate.java under INSTALL_DIR/samples/jdbc using javac Generate.java command. INSTALL_DIR is the directory where XML Extender is installed (for example: C:\dxx).

2. Create a table called RESULT_TAB with the following command:

   ```
   create table result_tab(doc varchar(3000))
   ```

3. Run Generate using the following command:

   ```
   java Generate xmltest C:\dxx\samples\dad\person.dad result_tab
   ```

   Where:

   XMLTEST is the database containing the relational data.
   C:\dxx\samples\dad\person.dad is the DAD file used to compose the XML document.
   RESULT_TAB is the table used to store the composed XML document.

4. Use the following SQL statements to extract the XML documents from RESULT_TAB table to files:

   ```
   select db2xml.content(db2xml.XMLVarchar(doc), 'DOC_DIR\person1.xml')
   from result_tab
   where db2xml.extractVarchar(db2xml.XMLVarchar(doc), '/person/@id') = '1'

   select db2xml.content(db2xml.XMLVarchar(doc), 'DOC_DIR\person2.xml')
   from result_tab
   where db2xml.extractVarchar(db2xml.XMLVarchar(doc), '/person/@id') = '2'
   ```

   Where:

   DOC_DIR is the directory where documents have to be extracted (for example, C:\xml\doc).
Figure 26 shows the XML document that we composed using DAD with SQL mapping.

```xml
<?xml version="1.0"?>
<!DOCTYPE person (View Source for full doctype...)>
  <person id="1">
    <firstName>Bill</firstName>
    <lastName>Smith</lastName>
  </person>
  <phone>
    <type>office</type>
    <number>408-231-300</number>
  </phone>
  <phone>
    <type>home</type>
    <number>408-200-301</number>
  </phone>
</person>
```

Figure 26. Person1.dtd

2.5.2.3 Creating DAD file using Administration Wizard

To create a DAD file for composition using SQL mappings:

1. Start the Administration Wizard by selecting Start > Programs > DB2 XML Extender > XML Extender Administration Wizard.

2. Connect to the DB2 database by completing the fields in Launchpad Logon window.
3. In the Launchpad window shown in Figure 27 click the **Edit DAD** button. Click **Next**.

![GUI XML Extender: Select a Task](image)

*Figure 27. GUI XML Extender: Select a task*
4. In the first step of Work with DAD Files Wizard, leave the File name field empty. Select XML collection SQL mapping from the type list box as shown in Figure 28. Click Next.

![Figure 28. Specify a DAD](image)

5. In step 2 of 6 select whether to validate the XML document or not. We selected Do not validate, as shown in Figure 29. Click Next.

![Figure 29. Select Validation](image)
6. In step 3 of 6 type `<?xml version="1.0"?>` in the Prolog field and
`<!DOCTYPE person SYSTEM "C:\dxx\samples\dtd\person.dtd"` in the
Doctype field as shown in Figure 30.

Figure 30. Specify text
7. In step 4 of 6, type the SQL statement in the **SQL Statement** field as shown in Figure 31 and click **Test SQL** to validate the SQL statement. Click **Next**.

![Figure 31. Specify SQL Statement](image-url)
8. In step 5 of 6, map the elements and attributes in the XML document to the columns in DB2. Follow the instructions provided to do the mapping. A sample image of the Map SQL Wizard is shown in Figure 32.

![Sample Map SQL window](image)

**Figure 32. Sample Map SQL window**

To add the root node:

a. Select the **Root** icon.

b. Click **New Element** to define a new node.

c. In the Details box, specify **Node type** as **Element**.

d. Enter the name of the top level node in the **Node name** field.

e. Click **Add** to create the new node.

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

a. Click 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:

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. The version of DB2 XML Extender we were using had a bug because of which you had to go back to the text or attribute node you just added, specify the column name again and click the Change button for the column name to be actually selected.

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 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 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.

Once you complete your mapping, the Map SQL window should look like the one shown in Figure 33.
9. Type the name of an output file for the modified DAD file in the **File name** field of the **Specify a DAD window**.

10. Click **Finish** to return to the Launchpad window.

The coding for the DAD file created using the GUI wizard follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "../dtd/dad.dtd">
<DAD>
  <validation>NO</validation>
  <Xcollection>
    <SQL_stmt>SELECT c.ID, COUNTRY, LNAME, FNAME, p.PHONE_ID, PHONE_NUMBER, PHONE_TYPE from CUSTOMER C, table (SELECT substr(char(timestamp(generate_unique())),16) as PHONE_ID, ID, PHONE_NUMBER, PHONE_TYPE from customer_details) P where P.ID=C.ID order by ID, PHONE_ID</SQL_stmt>
    <prolog><?xml version="1.0"?></prolog>
    <doctype>!DOCTYPE person SYSTEM "C:\dxx\samples\dtd\person.dtd"</doctype>
    <root_node>
      <element_node name="person">
        <attribute_node name="id">
          <column name="ID"/>
        </attribute_node>
      </element_node>
    </root_node>
  </Xcollection>
</DAD>
```
2.5.3 DAD with RDB_node

Here is the coding for the DAD file we used to compose the XML document with RDB_node mapping follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
  <validation>NO</validation>
  <Xcollection>
    <prolog><?xml version="1.0"?></prolog>
    <doctype>!DOCTYPE person SYSTEM "C:\dxx\samples\dtd\person.dtd"></doctype>
    <root_node>
      <element_node name="person">
        <RDB_node>
          <table name="customer">
          </table>
          <table name="customer_details">
            <condition>
              customer.id=customer_details.ID
            </condition>
          </table>
        </RDB_node>
      </element_node>
    </root_node>
  </Xcollection>
</DAD>
```
2.5.3.1 Preparing the DAD file having RDB_node mapping

To create the DAD file shown previously:

1. Open a text editor and create a file person_rdb.dad.

2. Create the DAD Header as follows:

   ```xml
   <?xml version="1.0"?>
   <!DOCTYPE DAD SYSTEM "c:\doc\dtd\dad.dtd">
   ```

3. Insert `<DAD></DAD>` tag. This element will contain all the other elements.

4. Insert the `<validation></validation>` tag to indicate whether DB2 XML Extender validates the XML document using the DTD in the DTD repository table. In this example we are not validating, so we insert the following code:

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

   If you intended to validate, you would insert the following code:

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

5. You use the `<Xcollection></Xcollection>` tags to specify the access and storage method as Xcollection.

6. When using RDB_node mapping, do not use the SQL_stmt element in the Xcollection element of the DAD file. Instead use the RDB_node element for the element_node, text_node and attribute_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:

     ```xml
     <element_node name ="person">
     <RDB_node>
     <table name="customer"/>
     <table name="customer_details"/>
     <condition>
     customer.id=customer_details.ID
     </condition>
     </RDB_node>
     ```

     If you are decomposing an XML document using 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. For a composite key, the key attribute is specified by the names of key columns separated by a space.

- 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 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.

**RDB_node for the attribute_node**

You need to define an RDB_node for each attribute_node to specify from which table and which column and query condition to use to get the data. In the DAD file we

- Specify the table name and column name having the required data. In our DAD file, the table for the attribute node `id` of element `person` is `customer` and the column is `id`. This is specified in the DAD file as follows:

```
<attribute_node name="id">
  <RDB_node>
    <table name="customer"/>
    <column name="id"/>
  </RDB_node>
</attribute_node>
```

**RDB_node for the text_node**

You need to define an RDB_node for each text_node to specify from which table and which column to get the data. Also you can optionally specify a query condition to get the data. In the DAD file we

- Specify the name of the table containing the column data. In our DAD file, the table for text_node of element `firstName` is `customer` and column name is `FName`. This is specified in the DAD file as follows:

```
<element_node name="firstName">
  <text_node>
    <RDB_node>
      <table name="customer"/>
      <column name="FName"/>
    </RDB_node>
  </text_node>
</element_node>
```
- You can optionally specify a condition if you want the XML documents to be composed using a query condition. A sample code segment follows:

```xml
<element_node name="firstName">
  <text_node>
    <RDB_node>
      <table name="customer"/>
      <column name="FName"/>
      <condition>
        FName like '%R%'
      </condition>
    </RDB_node>
  </text_node>
</element_node>
```

7. Place an ending </root_node> tag after the last </element_node> tag.

8. Place an ending </Xcollection> tag after the </root_node> tag.

9. Place an ending </DAD> tag after the </Xcollection> tag.

10. Save the file as person_rdb.dad

---

**Note**

XML Extender has a limitation when using RDB_node mapping because of which you cannot have sub-elements of an element mapping to columns in different table.

### 2.5.3.2 Composing the XML document

To compose the XML document you will use the dxxGenXML() stored procedure. There is a sample Java program shipped with XML Extender which can be used to compose the document.

Skip steps 1 and 2 if you have already done them in 2.5.2.2, “Composing the XML document” on page 74.

To compose the document:

1. Compile the Java program Generate.java under INSTALL_DIR/samples/jdbc using javac Generate.java command. INSTALL_DIR is the directory where XML Extender is installed (for example: C:\dxx).
2. Create a table called RESULT_TAB with the following command:

```sql
create table result_tab(doc varchar(3000))
```

3. Run Generate using the following command:

```java
java Generate xmltest C:\dx\samples\dad\person_rdb.dad result_tab
```

Where:
- XMLTEST is the database containing the relational data.
- C:\dx\samples\dad\person_rdb.dad is the DAD file used to compose the XML document.
- RESULT_TAB is the table used to store the composed XML document.

4. Use the following SQL statements to extract the XML documents from RESULT_TAB table to files:

```sql
select db2xml.content(db2xml.XMLVarchar(doc), 'DOC_DIR\persona.xml')
from result_tab
where db2xml.extractVarchar(db2xml.XMLVarchar(doc), '/person/@id') = '1'

select db2xml.content(db2xml.XMLVarchar(doc), 'DOC_DIR\personb.xml')
from result_tab
where db2xml.extractVarchar(db2xml.XMLVarchar(doc), '/person/@id') = '2'
```

Where:
- DOC_DIR is the directory where documents have to be extracted (for example C:\xml/doc).
2.5.4 Composition stored procedures

dxxGenXML(): The declaration for the dxxGenXML stored procedure follows:

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

This stored procedure is used to compose XML documents from the tables specified in the DAD file which is passed as input parameter. Each XML document is stored as a row in the result table. It also lets you specify the maximum number of rows to be inserted in the result table.

dxxRetrieveXML(): The declaration for the dxxRetrieveXML stored procedure follows:

```sql
```

Figure 34 shows the XML document that we composed using DAD with RDB_node mapping.
This stored procedure is used to compose XML documents from the tables specified in the DAD file which corresponds to the collection name passed as input parameter. To use this stored procedure you should have an enabled collection. You can enable collections by following instructions in 2.5.6, “Enabling XML Collections” on page 93. Each XML document is stored as a row in the result table. It also lets you specify the maximum number of rows to be inserted in the result table.

2.5.5 Dynamically overriding values in the DAD file

For dynamic queries you can use two parameters in the dxxGenXML stored procedure to override conditions in the DAD file. The declaration for the dxxGenXML stored procedure follows:

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

The parameters you use to override conditions are the `overrideType` and `override` parameters. 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 RDB_node mapping using an XPath-based condition.

In a Java program to specify NO_OVERRIDE set overrideType variable to 0, for SQL_OVERRIDE set it to 1 and for XML_OVERRIDE set it to 2.

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 syntax shown in Figure 35.

Figure 35. Syntax for XML expression

This expression has the following parts:

Simple location path:
This is a simple location path using syntax defined by XPath.

Operators:
A space can be used to separate the operator from the other parts of the expression.

Value:
This can be a numeric value or a single quoted string.

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.

Similarly, you can use override when you are using dxxRetrieveXML().

### 2.5.6 Enabling XML Collections

When you enable an XML Collection, the DAD file is parsed to identify the tables and columns related to the XML document and this information is stored in the XML_USAGE table.

#### 2.5.6.1 Enabling collections from DB2 command shell

To enable a collection from the DB2 command shell, issue the following command:

```
dxxadm enable_collection dbName collection DAD_file -t tablespace
```

In this command:

- **dbName**
  - This is the name of the database.

- **collection**
  - This is the name of the XML Collection. This value is used as a parameter for the XML Collection's stored procedures.

- **DAD_file**
  - This is the name of the file that contains the document access definition (DAD).

- **tablespace**
  - This is an existing table space that contains new DB2 tables that were generated for decomposition. If not specified, the default table space will be used.
2.5.6.2 Enabling XML Collections from Administration Wizard
To enable XML Collections from Administration Wizard, proceed as follows:
1. Start the Administration Wizard and get to the XML Administration Wizard launchpad window shown in Figure 36. Click **Work with XML Collections**.

![Figure 36. Admin Launchpad](image)

2. Select **Enable an XML Collection** as shown in Figure 37 and click **Next**.

![Figure 37. Enable Collection](image)
3. In the window shown in Figure 38 enter the **Collection name** (give any name), select the **DAD file name** by clicking the **Browse** button and optionally specify the **Table space** name which will contain the new DB2 tables. Click **Finish** to enable collection and return to the Administration launchpad window.

![Work with XML Collections Wizard](image)

**Figure 38. Collection details**

### 2.5.7 Decomposing XML documents using RDB_node mapping

You can use DB2 XML Extender to decompose XML documents into a set of relational tables. Decomposition uses RDB_node mapping, you cannot use SQL mapping when decomposing XML documents. To decompose an XML document to a set of relational tables you have to specify the mapping between elements and attributes to the relational tables using a DAD file. The steps to be followed are the same as those given in 2.5.3.1, “Preparing the DAD file having RDB_node mapping” on page 86.

The only differences in the DAD file used for decomposing an XML document, as opposed to composing an XML document using RDB_node, are these:

- You must specify a primary key for each table in the RDB_node for the top element in the DAD file. The primary key can consist of a single column or multiple columns, called 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, as follows:

  `<table name="customer_details" key="id phone_number"/>`


You must specify the column type for each attribute and text node. Column types are specified by adding the attribute type to the column element as shown:

```xml
<attribute_node name="id">
  <RDB_node>
    <table name="customer"/>
    <column name="id" type="integer"/>
    <table name="customer_details"/>
    <column name="id" type="integer"/>
  </RDB_node>
</attribute_node>
```

We used the following DAD file to decompose an XML document:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
  <validation>NO</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE person SYSTEM "C:\dxx\samples\dtd\person.dtd"</doctype>
    <root_node>
      <element_node name ="person">
        <RDB_node>
          <table name="customer" key="id"/>
          <table name="customer_details" key="id phone_number"/>
          <condition>
            customer.id=customer_details.ID
          </condition>
        </RDB_node>
        <attribute_node name="id">
          <RDB_node>
            <table name="customer"/>
            <column name="id" type="integer"/>
          </RDB_node>
        </attribute_node>
        <element_node name="firstName">
          <text_node>
            <RDB_node>
              <table name="customer"/>
              <column name="FName" type="varchar(30)"/>
            </RDB_node>
          </text_node>
        </element_node>
        <element_node name="lastName">
          <text_node>
            <RDB_node>
              <table name="customer"/>
              <column name="LName" type="varchar(30)"/>
            </RDB_node>
          </text_node>
        </element_node>
      </root_node>
    </Xcollection>
</DAD>
```
<RDB_node>
  <table name="customer"/>
  <column name="LName" type="varchar(30)"/>
</RDB_node>
</text_node>
</element_node>
<element_node name="phone" multi_occurrence="YES">
  <element_node name="type">
    <text_node>
      <RDB_node>
        <table name="customer_details"/>
        <column name="Phone_Type" type="varchar(20)"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="number">
    <text_node>
      <RDB_node>
        <table name="customer_details"/>
        <column name="Phone_Number" type="varchar(20)"/>
      </RDB_node>
    </text_node>
  </element_node>
</element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
The XML document we used to decompose is shown in Figure 39.

![XML document for decomposing](image)

**2.5.7.1 Decomposing the XML document**

To decompose the XML document, you will use the `dxxShredXML()` stored procedure. There is a sample Java program shipped with XML Extender which can be used to compose the document.

To decompose the document:

1. Compile the Java program `Shred.java` under `INSTALL_DIR\ samples\jdbc` using `javac Shred.java` command. `INSTALL_DIR` is the directory where XML Extender is installed (for example: C:\dxx).

2. Run `Shred` using the following command:

   ```
   java Shred xmltest C:\doc\samples\dad\person_rdb1.dad d:\data\person_in.xml
   ```
Where:

XMLTEST is the database containing the relational data.

C:\dxx\samples\dad\person_rdb1.dad is the DAD file used to decompose the XML document.

d:\data\person_in.xml is the XML document to decompose.

The following record was inserted into the customer table:

3,,"sharma","Ram"

The following records were inserted into the customer_details table:

3,"91-80-5482355","office"
3,"91-80-579117","home"

2.5.8 Decomposition stored procedures

dxxShredXML(): The declaration for the dxxGenXML stored procedure follows:

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

The dxxShredXML() stored procedure is the pairing stored procedure of dxxGenXML() and is used to decompose XML documents. 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 DB2 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.

```java
DxxInsertXML(): The declaration for the dxxInsertXML stored procedure follows:

```
2.5.9 Disabling XML Collections

2.5.9.1 Disabling XML Collections from DB2 command shell
To disable a collection from the DB2 command shell, give the following command

    dbxadm disable_collection dbName collection

Where

dbName

    The name of the database.

collection

    The name of the XML Collection. This value is used as a parameter for the XML Collection's stored procedures.

2.5.9.2 Disabling XML Collections from Administration Wizard
To disable an XML Collection from the Administration Wizard:

1. Start the Administration Wizard and get to the XML Administration Wizard screen shown in Figure 40. Click Work with XML Collections.

Figure 40. Admin Launchpad
2. Select **Disable an XML Collection** as shown in Figure 41 and click **Next**.

![Figure 41. Disable collection](image)

3. Enter the **Collection name** as shown in Figure 42 and click **Finish**.

![Figure 42. Collection Name](image)
2.6 Disabling the database for XML

When you want to clean up your XML Extender environment, you can disable the database to which you are connected.

Disabling a database for XML removes:

- All the user-defined types (UDT) and user-defined functions (UDF)
- All the control tables required by DB2 XML Extender, such as DTD_REF and XML_USAGE
- The db2xml schema

The command to disable the database is `dxxadm` with the option `disable_db`. For example, to disable our XMLTEST database, the command is:

```
dxxadm disable_db xmltest.
```

This function can also be performed with DB2 XML Extender GUI Wizard and with the Administration stored procedure `dxxDisableDB()`.
Chapter 3. DB2 Text Extender

Because DB2 XML Extender does not perform structural-text search and full text search, these text search functionalities are provided in the DB2 Text Extender product of the DB2 family.

If the content of an XML element or the value of an attribute is a large block of text, you can advantageously use DB2 Text Extender to search through it, either with a full text search or with a structural text search. DB2 Text Extender understands the XML structures and allows application queries to be based on the structure of the document.

The structural search extends the capability of a conventional full-text search by allowing search words to be matched within a specific document section specified by location path. For example, a valid query for DB2 Text Extender might be: Give me all the documents that contain the word ‘XML’ at the structure level or section ‘/rootElement/myElement’.

In order to do a text search on XML-enabled columns or text columns of XML collection tables, the application needs to enable these columns with DB2 Text Extender, just as it would have to do in order to search any other text column with DB2 Text Extender. DB2 XML Extender and DB2 Text Extender are independent, so that if you want to take advantage of both, you need to enable your tables and columns twice, once for each of these products.

In this chapter, we start by describing how to install DB2 Text Extender, then we detail the functionalities offered by DB2 Text Extender, and finally we explain how to work with it. You can immediately skip to 3.2, “Text Extender indexing capabilities” on page 108 if you first want to have a better idea of how useful Text Extender can be before installing it.

3.1 DB2 Text Extender installation

DB2 Text Extender is made up of two parts:

- DB2 Text Extender server: This is the main part of the DB2 Text Extender. You can only have one DB2 Text Extender server instance installed for each DB2 Server instance.

- DB2 Text Extender client: This part contains some DB2 Text Extender utilities installed on the DB2 client, such as a command line interpreter for administration tasks, the user-defined functions, and API functions for text queries. Of course, the user-defined functions and API functions are executed on the server where DB2 Text Extender search engine runs.
3.1.1 Installation requirements
To install DB2 Text Extender Version 7.1, you need DB2 Version 7.1 or above.

3.1.2 Installing the DB2 Extenders
Here we briefly describe the installation steps for the DB2 Extenders product on a Windows platform:

1. Insert the CD-ROM or unzip the downloaded zip file containing the DB2 Extenders.

2. Unless the CD autostart feature is enabled, launch the `x:\db2ext\winnt95\setup` program, where `x:` is the CD-ROM drive letter or the directory where you unzipped the DB2 Extenders software package. This displays the window shown in Figure 43.

![DB2 Extenders installation](image)

Figure 43. DB2 Extenders installation

3. You may check the installation prerequisites by clicking on **Installation Prerequisites**.
4. Click the **Install** button.

5. Choose what you want to install:
   - DB2 Extenders Server (non-EEE): The client and the server for any other edition.
   - DB2 Extenders Client: The client only.

6. Choose the setup type you want:
   - Typical: This installation type includes the DB2 Extenders sample files and SDK, the US English documentation and dictionary.
   - Compact: This installation does not include the DB2 Extenders SDK and the documentation.
   - Custom: This installation lets you select whether you want to install the DB2 Extenders SDK and sample files, the documentation, and any of the available dictionaries.

7. Choose the installation folder and let the components be installed.

8. Once all the components have been copied to your machine, restart your workstation to complete the install.

### 3.1.3 Creating a DB2 Text Extender instance

On the server side, you need to create a DB2 Text Extender instance for each instance of a DB2 server with which you want to use DB2 Text Extender capabilities. A default DB2 Text Extender instance named DB2 and associated with the DB2 instance is created when you install DB2 Text Extender on a Windows platform in a non-partitioned database environment.

If you want to associate a DB2 Extender instance with another DB2 instance, or if you work on another platform than Windows, than you need to create this instance with the `txicrt` Administration command for the DB2 Text Extender server. You execute this command from the operating system prompt or command line. The syntax for this command is slightly different, depending on your workstation platform:

**Note:** In the following two command examples, `DB2INST` is the name of an existing DB2 instance.

- On a Windows workstation, you create a new instance with the `txicrt` command, followed by the name of the DB2 instance you want a DB2 Text
Extender instance to be associated with. Here is a Windows example of this command:

\texttt{txicrt \texttt{DB2INST}}

- On a UNIX workstation, the \texttt{txicrt} command requires two parameters:
  - The user ID of the owner of the new DB2 Text Extender instance you are creating
  - The name of the DB2 instance to be associated with the new DB2 Text Extender instance

Here is a UNIX example of this command:

\texttt{txicrt \texttt{DB2INST DB2INST}}

---

Where to execute the commands?

All the DB2 Text Extender commands that start with `tx` are the Administration commands for the server. They must be run from the operating system prompt.

The other DB2 Text Extender commands are the Administration commands for the client and are to be run from the DB2 Text Extender Command Line Processor. To get this Text Extender Command Line Processor, either execute the command \texttt{db2tx} on the operating system prompt, or select \texttt{Start > Programs > DB2 Extenders > Text Extender > DB2 Text Extender Command Line Processor}.

---

3.1.4 Dropping a Text Extender instance

To drop a DB2 Text Extender instance, use the \texttt{txidrop} command. This command has the same syntax as the \texttt{txicrt} command. This command is illustrated in these two examples:

Windows example: \texttt{txidrop \texttt{DB2INST}}

UNIX example: \texttt{txidrop \texttt{USERID DB2INST}}

---

3.1.5 Checking the installation

To check if DB2 Text Extender was installed successfully, two commands are provided with DB2 Text Extender:
• TXVERIFY: This command creates and enables a database. (Enabling a database is discussed in 3.4.2, “Database enablement for Text Extender” on page 113).

• TXSAMPLE: This command creates a sample table in a database, import text documents into it, and enable the table and columns with the various index types. Enabling a column and the various indexes are discussed in the following sections:
  - 3.2, “Text Extender indexing capabilities” on page 108
  - 3.4.3, “Table enablement for Text Extender” on page 116
  - 3.4.4, “Column enablement for Text Extender” on page 116

To check the installation with these commands, follow these steps:
1. Open a DB2 Command Window and change the current directory to c:\dmb\samples, where c:\dmb is the directory where DB2 Text Extender has been installed.
2. Execute the TXVERIFY.BAT command file and check its output log file txverify.log. This command requires, as argument, the name of a DB2 database.
3. Execute the TXSAMPLE.BAT command file and check its output log file txsample.log. This command also requires, as argument, the name of a DB2 database.
4. If both log files contain no error messages, the installation has completed successfully.
The screenshot on Figure 44 displays the executed commands, with xmltest being the database name used for the testing.

![Image of executed commands]

**Figure 44. Verifying the installation of DB2 Text Extender**

### 3.2 Text Extender indexing capabilities

Now that we have installed DB2 Text Extender, we describe in this section what advantages DB2 Text Extender can bring to you by means of its indexing capabilities of text documents.

There are three types of indexes that allow you for a fast text search using DB2 Text Extender.

The type of index you choose for a text column influences what you can search and naturally what you can find. You can create more than one index (of different types) on the same text column if you want to benefit from the advantages provided by more than one index type.

Schematically, a text index contains a list of the significant terms contained in your documents, and a reference to where the documents are located. All the insignificant terms such as “a” and “the”, referred to as “STOP-WORDS”, are ignored by the indexing process.
Chapter 3. DB2 Text Extender

The indexing is a two-step process:

1. Record all the documents that need to be indexed in a log file (or more precisely, a log table). This step is automatically executed for you by the insert, update, and delete triggers created by DB2 Text Extender for each text column enabled for text search.

2. Periodically, process all the documents recorded in the log and index their significant terms to keep the content of the index synchronized with the content of the database. The period between two such processes is determined by the UPDATEFREQ text configuration setting that we describe in 3.4.2, “Database enablement for Text Extender” on page 113.

The indexing mechanism of DB2 Text Extender requires that you provide certain information about the text document. (These requirements are further described in 3.4.2, “Database enablement for Text Extender” on page 113.) Here are the three characteristics that you must specify:

• The format of the text document, for example, XML, HTML, or ASCII. For the structured documents, DB2 Text Extender allows you to search only in specific sections. This is defined in the document model described in 3.4.6, “Structural search on XML documents” on page 122.

• The language in which a document is written, so that the correct dictionary can be used

• The CCSID of the document

The three types of indexes available with DB2 Text Extender are linguistic, precise, and NGram. These are described in the following three sections.

3.2.1 Linguistic indexes

This indexing method gives you the opportunity to realize searches that we could qualify as “semantic” searches.

This index is based on a dictionary and allows you to find word variations. The words contained in your text documents are reduced to their base form before being placed into the index. For example, this index would store the verb “sleep” even if the term “slept” was contained in your document. Moreover searching for “sleep” would allow you to find all the documents containing any variation of the verb “sleep”, that is “slept”, “sleeping” and so on. After your document has been indexed, if you are searching for the term “slept”, your search argument itself is first normalized and undergoes the same transformation to become “sleep”, and secondly, the index is scanned in search for this normalized argument.
This type of search cannot be case sensitive.

This index method requires the least amount of disk space as all the variations of the same term are considered to be just this term. Of course, the price you have to pay for it is that the indexing itself and the searching can take longer for this type of index than for a precise index, described in the following section.

3.2.2 Precise indexes

This index stores the term exactly the way they are in the text documents. When searching, you only find the documents containing the exact term you are looking for and not any variation of it. In this case, a search for “sleep” would not find the documents containing the word “slept” or any other variation of it and searching for “slept” would ignore anything but “slept”.

The type of indexing allows for a more precise search that can be case sensitive.

The indexing and retrieval is faster, but the indexes require more disk space than what would be necessary for indexing the same text documents with a linguistic index.

3.2.3 NGram indexes

For this type of index, the index creation is not based on a dictionary. The text document is considered as a set of sets of characters. The indexing is character-based rather than being word-based. It allows you to search for character variations.

For example, you are able to find a word even it is misspelled (providing at least the first three characters are right).

This is the only indexing method available for DBCS documents. For the SBCS documents, only the TDS documents (or flat ASCII) can be indexed using this method.
3.3 Text Extender environment variables

It is important that you check that all the required environment variables are set for you to work with DB2 Text Extender.

The two most important DB2 Text Extender environment variables are:

- **DB2INSTANCE**: The DB2 UDB instance name. This variable must have a value.
- **DB2DBDFT**: The default database name. You must add this environment variable if it does not exist on your workstation. To add a new system environment variable on a Windows platform, follow these steps:
  
a. Select Start > Settings > Control Panel > System.
  
b. On the System Properties window, select the Environment tab.
  
c. Change the content of the **Variable**: field in DB2DBDFT.
  
d. Change the content of the **Value**: field in the name of the default database you want.
  
e. Click **Set**.
  
f. Click **Apply** and close the window.

Figure 45 shows what the result should be, if you have been using the default database, XMLTEST.
On the UNIX platform, the other environment variables DB2TX_INSTOWNER, DB2TX_INSTOWNERHOMEDIR, LANG, NLSPATH and DB2ENVLIST must also be defined. Refer to *IBM DB2 Universal Database Text Extender Administration and Programming*, SC26-9930 for more information about these variables.
Now that all the environment variables are set, you can display them with the GET ENVIRONMENT Text Extender command, which must be executed from the DB2 Text Extender Command Line Processor. Figure 46 shows DB2 Text Extender environment variables on a Windows platform.

![Figure 46. Text Extender environment variables](image)

### 3.4 Text extender use

In this section, we describe the successive steps required to index a text document and search through it.

#### 3.4.1 Starting and stopping the DB2 Text Extender server

You start a DB2 Text Extender server with the `txstart` command and stop it with the `txstop` command.

To display its status, use the `txstatus` command.

#### 3.4.2 Database enablement for Text Extender

On the DB2 Text Extender command prompt (`db2tx`), connect to the database you want to enable for Text Extender and issue the `ENABLE DATABASE` command for DB2 Text Extender.

For example: `db2tx => ENABLE DATABASE`

Once a database has been enabled, it can be disabled with the `DISABLE DATABASE` command. This command disables the current database for Text Extender.

When a database is enabled for use with DB2 Text Extender, a set of text configuration settings are created automatically. They can be grouped into three categories: Text, indexing, and processing characteristics.
• **Text characteristics:**
  - **FORMAT:** The format of the text document. The initial value is TDS, or flat ASCII. The other supported formats are HTML, XML, ASCII_SECTIONS, TDS, AMI, FFT, MSWORD, RFT, RTF and WP5.
  - **LANGUAGE:** The language of the text document. The initial value equals the LANGUAGE set for the database. There are currently more than 20 languages supported among which US_ENGLISH, UK_ENGLISH, ARABIC, FRENCH, GERMAN, RUSSIAN and SPANISH.
  - **CCSID:** The CCSID of the text document. The initial value equals CCSID set for the database. EBCDIC, ASCII, DBCS and UNICODE CCSID are supported. Check in *IBM DB2 Universal Database Text Extender Administration and Programming*, SC26-9930 for the whole list of supported CCSIDs.

• **Index characteristics:**
  - **DIRECTORY:** The directory where the indexes are stored. For Windows clients, this value must be set to the name of a directory on the server machine. For Windows systems, the initial value is c:\dmb\instance\instance-name\db2tx\indexes, where c:\dmb\ is the installation directory of DB2 Text Extender.
  - **INDEXTYPE:** The index type to be used. The initial value is NGRAM. The other possible values are PRECISE and LINGUISTIC.
  - **UPDATEFREQ:** The frequency to which the indexes must be updated. The initial value is NONE, meaning that the index is not updated.

• **Processing characteristics:**
  - **UPDATEINDEX:** Indicates whether the newly created index must be updated immediately after its creation or only during the next periodic update. The possible values are UPDATE and NOUPDATE.
  - **COMMITCOUNT:** Determine after how many insert or update statements, DB2 Text Extender issues a DB2 UDB commit. The initial value is 0.
All these text configuration settings and their default values can be displayed with the GET TEXT CONFIGURATION Text Extender command. Figure 47 shows the default text configuration settings on a Windows NT platform.

Figure 47. Default text configuration settings

If you want to change any default values, do so with the DB2 Text Extender command CHANGE TEXT CFG USING, then the settings you want to change, as shown in this example:

```
db2tx => CHANGE TEXT CFG USING INDEXTYPE precise DIRECTORY c:\idxs
         UPDATEFREQ min(20) d(1,2,3,4,5) h(7,12,17) m(00) FORMAT xml
```

This command updates the text configuration settings in the following ways:

- **INDEXTYPE precise**: The default type of any new index is now PRECISE
- **DIRECTORY c:\idxs**: The default directory where any new index is stored is c:\idxs
- **UPDATEFREQ min(20) d(1,2,3,4,5) h(7,12,17) m(00)**: The new default update period for any new index is now from Monday to Friday (d(1,2,3,4,5)), at 07:00, 12:00 and 17:00, if the document log of the index contains at least 20 entries (min(20)).
- **FORMAT xml**: The default format for any document stored in a newly enabled column is XML.

For more details concerning the CHANGE TEXT CONFIGURATION command, refer to *IBM DB2 Universal Database Text Extender Administration and Programming*, SC26-9930.
3.4.3 Table enablement for Text Extender

This step is optional. You only need to enable a table for text search if you want to create a single index for the whole table and thus for all the columns that you will later enable for text search. Otherwise, each enabled column has its own index rather than sharing the same index. When the table has not been enabled before enabling a column, the table is known as a multi-index table.

Creating an index for each text column allows for more flexibility in the characteristics associated with a column index, such as its type, the update period, and the directory where it is stored, as well as a better spread of the resource-consuming activities, by not updating all the indexes at the same time.

Use the `ENABLE TEXT TABLE` command followed by the name of the table you want to enable for your table.

For example:

```
db2tx => ENABLE TEXT TABLE table_name
```

This command creates an index with the default text configuration settings. We can also override them in the command itself. The following example shows how to create a linguistic index:

```
db2tx => ENABLE TEXT TABLE table_name INDEXTYPE linguistic
```

3.4.4 Column enablement for Text Extender

Enabling a text column is a required step before you can search in it, even if you previously enabled your table for text search, as described in 3.4.3, “Table enablement for Text Extender” on page 116.

Use the `ENABLE TEXT COLUMN` command to enable your column. The required parameters are the table name, the column name, and a handle column name. The handle column is a 60-byte Varchar column added by DB2 Text Extender to the table when the column is enabled. This handle contains information such as a unique document ID, the document language, format and CCSID, the index name. When searching among the documents stored in this column, you must specify the column handle associated with the enabled text column and not the text column itself.

When you enable a text column, DB2 Text Extender checks if the table had previously been enabled. If that is the case, no index is created. Otherwise, it creates a new index automatically for you.
Example: `db2tx => ENABLE TEXT COLUMN tabl coll HANDLE hcoll`

This command enables a text column using the default text configuration settings of the database. If you want different configuration settings, you can specify them explicitly in the ENABLE TEXT COLUMN command to override the default ones.

Example: `db2tx => ENABLE TEXT COLUMN tabl coll HANDLE hcoll FORMAT HTML`

There are also various commands that allow you to get information about the indexes created: GET STATUS, GET INDEX STATUS, GET INDEX SETTINGS and GET TEXT INFO.

- GET STATUS displays which tables and text columns have been enabled by DB2 Text Extender. Example: `db2tx => GET STATUS`. The output of this command is shown in Figure 48.

```
db2tx => get status

The database is enabled for DB2 Text Extender

Table XMLEXT.TAB1 is enabled as a multi-index table
TextColumnName    HandleColumnName
COL1              HCOL1
```

**Figure 48. GET STATUS Text Extender command**

- GET INDEX STATUS gives information about the status of a particular index in a table: if it can be updated or searched, the number of documents, any error events. Example: `db2tx => GET INDEX STATUS tabl HANDLE hcoll`. The output of this command is shown in Figure 49.

```
db2tx => get index status tab1 handle hcoll

Node  0
Search status = Search available
Update status = Update available
Reorganization status = Reorganization available
Scheduled documents = 0
Indexed documents = 0
Primary index documents = 0
Secondary index documents = 0
Error events = No error events.
```

**Figure 49. GET INDEX STATUS Text Extender command**
• GET INDEX SETTINGS displays the index and processing characteristics of a given index. Example: `db2tx \=> GET INDEX SETTINGS tab1 HANDLE hcol1`. The output of this command is shown in Figure 50.

```sql
db2tx \=> get index settings tab1 handle hcol1

Current index settings for text column COL1 with handle column HCOL1:
Index type (INDEXTYPE) = NGRAM
Update index option (UPDATEINDEX) = Update
Update frequency (UPDATEFREQ) = NONE
Node 0
Index directory (DIRECTORY) = c:\dmb\INSTANCE\DB2\DB2TX\INDEXES
```

Figure 50. GET INDEX SETTINGS Text Extender command

• GET TEXT INFO displays the text characteristics of an index. Example: `db2tx \=> GET TEXT INFO tab1 HANDLE hcol1`. The output of this command is shown in Figure 51.

```sql
db2tx \=> get text info tab1 handle hcol1

Text information for column COL1 with handle column HCOL1:
Coded character set ID (CCSID) = 1252
Language (LANGUAGE) = US ENGLISH
Format (FORMAT) = HTML
```

Figure 51. GET TEXT INFO Text Extender command

---

---Text column data type---

If the text column you want to enable for DB2 Text Extender has a user-defined type (for example, an XML user-defined type provided by DB2 XML Extender), you must provide a user-defined function responsible for transforming the user-defined type into one of the data types supported by DB2 Text Extender: CHAR, VARCHAR, LONG VARCHAR, CLOB, DBCLOB, GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC.

For the UDTs defined with DB2 XML Extender, these casting functions are provided with the Extender. For example, to convert a XMLCLOB into a CLOB or an XMLVarchar into a varchar, use respectively the functions `DB2XML.CLOB()` and `DB2XML.VARCHAR()` as follows:

```sql
db2tx \=> ENABLE TEXT COLUMN xmltable xmlclobcol FUNCTION DB2XML.CLOB
          HANDLE hxmlcol
```

---
Now that the text column is enabled, the text documents it contains can be searched. We describe how to do this in the following section.

### 3.4.5 Querying the text documents

DB2 Text Extender provides user-defined functions that enable you to include text search subqueries in SQL queries. These functions are contained in the schema `db2tx`.

Update the function path with the `SET CURRENT FUNCTION PATH` command to avoid prefixing each call to a Text Extender function by `db2tx`.

#### 3.4.5.1 DB2 Text Extender UDFs

There are 13 UDFs, but we will limit our description to these three, which are the most useful:

- **CONTAINS(column-handle, search-argument):** This function is used to query the content of an enabled text column referred to by its column-handle. It returns 1 if the text satisfies the search argument, otherwise it returns 0.

- **NO_OF_MATCHES(column-handle, search-argument):** This function returns the number of matches satisfying the search argument found in a text document.

- **RANK(column-handle, search-argument):** This function returns the rank of a document. The rank is an absolute value (of type DOUBLE, ranging between 0 and 1) representing how well the document met the search criteria. This value depends on the number of matches found in the document in relation to the document size.

When large tables are involved and in order to improve the search performance, it is indicated to replace the use of the previous functions by the table function `SEARCH_RESULT` that returns the NO_OF_MATCHES and RANK for each document in the searched table in an intermediate table.

These functions are easy to use. However, a somewhat more difficult part of the job is formulating the search-argument. We explain how to do this in detail in the following section.
3.4.5.2 The search-argument
The search-argument is a character string, and it must be enclosed between simple quotes ('). Following is a list of the main ways that search arguments can be formulated:

- A single argument: "ARG". Each argument is surrounded by double-quotes. You can also add brackets, for example: '("ARG")'.

- A list of arguments combined by OR: You separate each argument by a comma (,) and put all the list of arguments between brackets.
  For example: '("ARG1","ARG2")' will search for any text document containing ARG1 or ARG2.

- A list of arguments combined by OR and AND. You use the operator & for the boolean AND, and the operator | for the boolean OR. The AND operation has a higher priority than OR operation, which means that AND is evaluated before OR. You can impose your priority with the use of brackets. The operators of the same priority are evaluated from left to right.
  For example:
  - '("ARG1" | "ARG2" & "ARG3")' search for any text document containing ARG1 or both ARG2 and ARG3.
  - '(("ARG1" | "ARG2") & "ARG3")' search for any document containing ARG3 and at least one of the two terms ARG1 or ARG2.

- To exclude any document containing a given argument, use the NOT operator.
  Example: '("ARG1" & NOT "ARG2")'
  This example returns all the documents containing the term ARG1 and that do not contain the term ARG2.

- Using wildcard characters to represent optional characters anywhere in a search term. This possibility is very useful to search for variations of a term when using a precise index. You must pay attention when using this with a linguistic index a term containing a wildcard is not normalized to its based form when searching for it.
  You can choose between the two following masking characters:
  - The underscore sign (_): it represents one character
  - The percent sign (%): it represents any number of characters
  For example: '("ARG_", "%MENT")' search for any text document containing a 4-letter word starting by 'ARG' or a word ending by 'MENT'.

Integrating XML with DB2 XML Extender and DB2 Text Extender
• Using an escape character to search for characters that can be interpreted like special characters by the DB2 Text Extender parser, like (_) and (%).

You can define the escape character you want, like this.

Example: ‘("XML$_PAGE" ESCAPE "")’

In this example, we define the character dollar ($) as being the escape character. The search argument will look for any document containing the term XML_PAGE.

• Use the escape character back slash (\) if you want your search argument to contain special characters that could be removed by the SQL parser like a double quote sign (").

Example: ‘\"Hi\"’

This search argument looks for a document containing the term “Hi” with the double quotes included. It will not find a document containing ‘Hi’ with no double-quote.

In the following examples we briefly discuss how to execute structural searches:

• To find terms in the same sentence, use the IN SAME SENTENCE AS keyword.

Example: ‘"ARG1" IN SAME SENTENCE AS "ARG2" AND "ARG3"’

This example searches for any documents containing the terms ARG1, ARG2 and ARG3 in the same sentence.

• To find terms in the same paragraph, use the IN SAME PARAGRAPH AS keyword.

• To find a term contained in a section defined in a document model (as explained in 3.4.6, “Structural search on XML documents” on page 122), use the MODEL and SECTION keywords.

Example: ‘MODEL model1 SECTIONS (root/section1) "ARG1"’

This example looks for all the documents containing the term ARG1 in their section root/section1 as defined in the document model model1.

We now introduce some considerations related to the linguistic indexes:

• To find a term or one of its synonym, use the SYNONYM FORM OF keyword.

Example: ‘SYNONYM FORM OF “customer”’
This search argument searches for any variations of any synonym of customer. The synonyms are defined in the US_ENGLISH dictionaries. If you want to use another dictionary, you need to specify it even if the language of the document is not US_ENGLISH.

Example using a French dictionary: ‘SYNONYM FORM OF FRENCH “client”’

- For more advanced linguistic search topics, refer to IBM DB2 Universal Database Text Extender Administration and Programming, SC26-9930.

Finally, the NGram indexes give the possibility of executing fuzzy searches. A fuzzy search looks for words that are spelled in a similar way to the search term, but accepts some differences. This is helpful in order to find misspelled words. There are 5 degrees of tolerance or accuracy in what is considered as a matching term. The level 1 gives the highest number of matches with the lowest degree of accuracy (about 20 percent of the searched terms correspond to the found terms) and level 5 is the highest degree of accuracy (with around 90 percent of term matching).

3.4.6 Structural search on XML documents

Before being able to structurally query an XML document or any text document, we must define its structure in a document model.

3.4.6.1 Definition of a document model

The document model contains the description of the structured documents like XML or HTML stored as text documents. It defines the markup tags used in the structured documents and their corresponding section names. It also defines which sections are indexed. The definition of these sections makes it possible to search, for example, only in the section “summary” of a text document.

The property INDEXPROPERTY SECTIONS_ENABLED must have been specified when the text column was enabled to text search in order for DB2 Text Extender to support sections for this document.

The sections are divided in two categories:

- The plain-text sections: These have no type.
- The attribute sections: These have a declared type. Using these declared types, it is possible to search for documents whose given attribute section is within a specified range.
Text Extender supports the following attribute types:

- DATE
- TIME
- FLOAT
- INTEGER

All the document models for the server instance are listed in the **document models file**. The default document models file is DESMODEL.INI, which is created automatically with the server instance. This initial file is placed in the directory C:|dmb|instance\DB2|db2tx|txins000, where C:|dmd is the installation directory of the DB2 Extenders and where DB2 is the name of the instance.

At index creation time, this file is copied in the index directory, a directory such as C:|dmb|instance\DB2|db2tx\INDEXES\TXINS000\IX414314\index. Any changes to the initial document models file in the initial directory after the creation of the index has no more influence on this index, as it now uses its own version of the document models file.

Each document model in the document models file is made up of two components: its name and its description. When describing an XML document, the document model must have the same name and case as the root element. For each element or attribute you want to define and use as a section, its complete hierarchy must be included in the model description.

To give you a better understanding of how to build a document model, we use a letter as illustration. We want to be able to search in the following structured XML document: letter.xml. Here is the coding for this document:

```xml
<?xml version="1.0" ?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>
  <company>IBM</company>
  <date day="30" month="September" year="2000" />
  <inside-address>
    <name>
      <first-name>Joe</first-name>
      <last-name>Duluth</last-name>
    </name>
    <street>123 Main Street</street>
    <city>San Francisco</city>
    <state>CA</state>
    <zip>12345</zip>
  </inside-address>
  <subject>Introduction to XML</subject>
</letter>
```
Dear Arthur,

It was a pleasure to meet you at XML World '99 in Ottawa recently. As promised, I'm enclosing a copy of the "Introduction to XML" tutorial.

Please let me know if you have any comments on the tutorial, or if you have suggestions for future topics. We'd really like your feedback!

Sincerely,

Joe Duluth
Senior Engineer

For a better understanding of the structure (although we do not use any validation here), here is the DTD file, letter.dtd, validating this XML document:

```
<!ELEMENT letter (company, date, inside-address, attention?, subject?, salutation, body, closing, signature)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT return-address (street+, city, state, zip)>
<!ELEMENT date EMPTY>
<!ATTLIST date day CDATA #REQUIRED
month CDATA #REQUIRED
year CDATA #REQUIRED>
<!ELEMENT inside-address (name, street+, city, state, zip)>
<!ELEMENT attention (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT salutation (#PCDATA)>
<!ELEMENT body (para+)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT closing (#PCDATA)>
<!ELEMENT signature (name, job-title?)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT name (title?, first-name, last-name)>
```
We now decide that we want to be able to search specifically in some of the sections of this letter, for example, we want to search the content of the company, subject, para and job-title elements. Therefore, we must define the corresponding sections in our document model. For any section, it is also necessary to define all the underlying sections in the hierarchy. This results in the following document models file containing only our document model letter:

```
; list of document models
[MODELS]
modelname = letter

; left side = section name identifier
; right side = section name tag

[letter]
/letter = /letter
/letter/company = /letter/company
/letter/subject = /letter/subject
/letter/body = /letter/body
/letter/body/para = /letter/body/para
/letter/signature = /letter/signature
/letter/signature/job-title = /letter/signature/job-title
```

1 In the document models file, each line starting with a semicolon is a comment.

2 The first part of the document models file groups all the name of the models. It starts with the [models] ‘tag’. In this document model file, we only have one model: letter.

3 The second part of the document models file contains the description of each model named in the first part. Each description starts with the name of the model between square brackets. The tag [letter] indicates the start of the description of the letter document model.
4 The syntax for defining a section is:

sectionname = xpath notation of the considered element or attribute

This means that the name of the section does not have to be the xpath notation of the element or attribute represented by that section. But to avoid any confusion, we have chosen to use the same naming convention, so that when using the section name for querying the content of our XML document, we have no doubt about which element or attribute is actually being queried.

5 Even if the element body, with a xpath notation of /letter/body, does not contain any text, and as such, cannot be searched by our Text Extender, we need to define it in our document model, as it contains the paragraph elements that we want to query.

3.4.6.2 Structural query

In this section, we describe how to structurally query the XML document letter.xml described in the previous section. To illustrate the various possibilities offered by Text Extender, in this example we make use of the external file feature of DB2 Text Extender: It allows external files to be indexed without inserting them physically in a column. For more information about the syntax concerning the external files and XML Extender, please refer to IBM DB2 Universal Database Text Extender Administration and Programming, SC26-9930.

1. Create the table, xmlfile, using the SQL statement:

   ```sql
   create table xmlfile (id integer)
   ```

2. Enable an external file column with the DB2 Text Extender command:

   ```sql
   db2tx => enable text files xmlfile handle hfile format xml indextype linguistic indexproperty sections_enabled documentmodel (letter)
   ```

   This command creates a index handle column in our xmlfile table, and the corresponding index is enabled for working with sections.

3. Insert a row in the xmlfile table using the SQL statement:

   ```sql
   insert into xmlfile values(1, db2tx.init_text_handle(1252, 'XML','US_ENGLISH','c:\SG246130\code\letter.xml'));
   ```

   At this time, DB2 Text Extender has logged the fact that the file letter.xml must be indexed, but this has not yet been done. As we want to test immediately, we must force DB2 Text Extender to update this index.

4. Update the index using the DB2 Text Extender command:

   ```sql
   update index xmlfile handle hfile
   ```
5. Query the XML document through its linguistic index.

- Our first query searches the whole document for the word “enclose”. This word is not precisely contained in the document, but exists as “enclosing”. As we are using a linguistic index, the document should be considered to contain it. The following SQL statement confirms with 1 match found, in the only document contained in the table:

  ```sql
  select db2tx.no_of_matches(hfile, '("enclose")') from xmlfile
  ```

- Our second query searches for the same word “enclose”, not in the whole document, but only in the sections “para”. The following SQL statement indicates us that the word “comment” is indeed encountered in a “para” with 1 match found:

  ```sql
  select db2tx.no_of_matches(hfile, 'model letter sections (/letter/body/para) "comment"') from xmlfile
  ```

- Our last queries focus on the content of the section “subject”. When searching, if it contains the word “enclose”, we get 0 matches found, but 1 match is found if we search for the word “introduction”:

  ```sql
  select db2tx.no_of_matches(hfile, 'model letter sections (/letter/subject) "enclose"') from xmlfile
  ```

  ```sql
  select db2tx.no_of_matches(hfile, 'model letter sections (/letter/subject) "introduction"') from xmlfile
  ```

Of course, we can easily combine the advantages offered by both DB2 XML Extender and DB2 Text Extender by executing a structural search on a document stored in an XML column, such as an XMLCLOB datatype, as follows.

If the text column you want to enable for DB2 Text Extender has a user-defined type, for example, an XML user-defined type provided by DB2 XML Extender, you must provide a user-defined function responsible for transforming the user-defined type into one of the data types supported by DB2 Text Extender: CHAR, VARCHAR, LONG VARCHAR, CLOB, DBCLOB, GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC.

For the UDTs defined with DB2 XML Extender, these casting functions are provided with DB2 Extender. For example, to convert a XMLCLOB into a CLOB or an XMLVarchar into a varchar, use respectively the functions DB2XML.CLOB() and DB2XML.VARCHAR() as follows:

```sql
db2tx => ENABLE TEXT COLUMN tab_nam xmlclob_col_nam FUNCTION DB2XML.CLOB HANDLE handle_nam INDEXPROPERTY SECTIONS_ENABLED DOCUMENTMODEL (letter)
```
Here, tab_nam is the name of the table containing the XML column, xmlclob_col_nam is the name of the XML column, and handle_nam is the name of the handle we associate with this column for text searching.

This illustration concludes our description of the query features of DB2 Text Extender.

### 3.4.7 Working with the indexes

In Table 9 we give a summary of the important Text Extender commands that are related to the indexes. For more information regarding these commands, refer to IBM DB2 Universal Database Text Extender Administration and Programming, SC26-9930.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET STATUS</td>
<td>To get information about the text enabled database, tables and columns</td>
</tr>
<tr>
<td>GET ENVIRONMENT</td>
<td>To get information about the environment variables for DB2 Text Extender</td>
</tr>
<tr>
<td>GET TEXT CONFIGURATION</td>
<td>To get the text configuration settings (text, index and processing characteristics)</td>
</tr>
<tr>
<td>GET INDEX STATUS table</td>
<td>To check any information or message available for an index</td>
</tr>
<tr>
<td>GET INDEX STATUS table HANDLE</td>
<td>Same as above but for one index in a multi-index table</td>
</tr>
<tr>
<td>GET INDEX SETTINGS table</td>
<td>To get the information about the settings of an index (type, update period, directory)</td>
</tr>
<tr>
<td>GET INDEX SETTINGS table HANDLE</td>
<td>Same as above but for one index in a multi-index table</td>
</tr>
<tr>
<td>GET TEXT INFO table</td>
<td>To get the information about the text settings of an index (language, format and CCSID)</td>
</tr>
<tr>
<td>GET TEXT INFO table HANDLE</td>
<td>Same as above but for one index in a multi-index table</td>
</tr>
<tr>
<td>UPDATE INDEX table</td>
<td>To update an index immediately without waiting for periodic indexing to occur</td>
</tr>
<tr>
<td>UPDATE INDEX table HANDLE</td>
<td>Same as UPDATE INDEX but for an index from a multi-index table</td>
</tr>
</tbody>
</table>
CHANGE TEXT CONFIGURATION USING ...
To change any of the default text configuration settings of an enabled database

CHANGE INDEX SETTINGS table UPDATEFREQ freq
To change the update frequency of an index or to stop its periodic indexing

CHANGE INDEX SETTINGS table HANDLE handle UPDATEFREQ freq
Same as above but for one index in a multi-index table

REORGANIZE INDEX table
To reorganize an index

REORGANIZE INDEX table HANDLE handle
Same as above but for one index in a multi-index table

ENABLE DATABASE
Enable the current database for Text Extender

ENABLE TEXT TABLE table
Create an index for the specified table

ENABLE TEXT COLUMN table column HANDLE columnhandle
Create an index for the specified column in a multi-index table
Chapter 4. Code page considerations

This chapter discusses various code page considerations. First we give you some basic information about character set and code pages. We consider how DB2 UDB provides support for the various code pages defined by the National Language Support (NLS) feature. Then we briefly discuss the encoding declaration of XML documents. Finally we show code page settings supported by DB2 XML Extender.

4.1 Character sets and code pages

Computers can understand only a stream of bit patterns. Therefore, if we want to work with characters using a computer, the character must be translated to a bit pattern. This translation is called encoding.

Usually, a bit pattern is represented as a sequence of 1s or 0s. However, it is difficult for us to recognize a number thus represented. We usually think of the bit pattern as binary numbers, and use them after translating them to hexadecimal numbers.

As a result, each character is represented as a hexadecimal number, which is called the code point of the character. A set of characters which are related uniquely to each of the code points in this way is called a coded character set. We look at some of these character sets in the next section.

4.1.1 Character sets and their encoding

In this section we consider several character sets to understand how character sets are defined and used.

**ASCII**

The first characters used on computers were the well-known American National Standard Code for Information Interchange (ASCII) character set defined in 1963. The ASCII characters are shown in Figure 52.

In this table, the hexadecimal representation of a character which locates the column number \( n \) and the row number \( m \) is \( X'\!mn' \). ASCII defines 94 characters and some other control characters which are represented in 7 bits. Control characters are defined from \( X'00' \) to \( X'20' \) and \( X'7F' \), some of which are \( X'20' \) (space), \( X'7F' \) (delete), \( X'0D' \) (carriage return), \( X'0A' \) (line feed), and so on.
As ASCII was defined to be used in the American language environment, it does not include the particular alphabets or signatures of other languages, and therefore it is not convenient to use in other countries.

To resolve this problem, ISO/IEC 646 (ISO 646) was defined. In ISO 646, the characters of code points X'23', X'24', X'40', X'5B' to X'5E', X'60' and X'7B' to X'7E' (the grey cells on the left side of the table shown in Figure 53) can be re-defined for the use of each country.

**Note:** The various versions of ISO 646 are defined for each country. For example, the French version of ISO 646 is shown in Figure 54 on page 134.
However, another problem occurs when the character data is interchanged between countries. As the re-defined characters are different in each country, strings including such characters may have different meanings, or may be meaningless in other countries. For example, “français” is X’66’ X’72’ X’61’ X’6E’ X’5C’ X’61’ X’69’ X’73’ in French ISO 646, but this bit pattern is a representation of “fran\ais” in ASCII. To solve this problem, ISO 2022 was defined.
ISO 646

Remember that in ISO 646, all characters are encoded with 7 bits only. ISO/IEC 2022 (ISO 2022) extends this and defines 8 bits for encoding characters. The basic concept of ISO 2022 is that it extends the coding area of ASCII (X'00' to X'7F') to the area X'80' to X'FF'.

The extended area is assigned for a control character area (X'80' to X'A0') and a character area (X'A1' to X'FE'). Also, it refers to the area from X'21' to X'7E' as "G0", and the area X'A1' to X'FE' as "G1" (shown in Figure 55). Moreover, it defines particular characters called “escape sequences” to designate each of the character sets to G0 or G1.

For example, the escape sequence to designate French ISO 646 to G1 is X'1B' X'29' X'66', so if this escape sequence appears in the bit pattern of a sentence, the bytes in the area X'A1' to X'FE' after this escape sequence are recognized to represent French ISO 646 characters. There are also some extensions of ISO 2022 for Chinese, Japanese, Korean, and so on.
ISO 8859
As 8-bit characters were defined by ISO 2022, a new character set has been defined to represent some European languages. This is ISO/IEC 8859-1 (ISO 8859-1; also called latin-1). This is shown in Figure 56. Designating it to G1 and ASCII to G0, some European languages such as English, French, German, Italian, Dutch, Spanish, Portuguese, Irish, and so on, can be coded, and the character data can be interchanged in these countries. There are other ISO 8859 versions (from ISO 8859-2 to ISO 8859-10) to encode other European languages.

Multi-byte characters
The number of characters is limited to 96 characters in ISO 2022. However, some languages, such as Chinese, Korean, and Japanese, have more than 96 characters. To encode these characters, each must be represented in more than one byte.

For example, in the Japanese character set JIS X0208, each Japanese character is represented in two successive byte elements. That is, a table of 94 times 94 characters is defined, and each character is represented by the column and the row number. The column and the row number is related to the first and the second byte, respectively. Also, ISO 2022 escape sequences have been defined to switch the character set to ASCII or other character sets.
As for Japanese, there are three ways to encode these characters: EUC JP, ISO-2022-JP, and Shift_JIS. These character sets are not compatible.

For more information about ISO/IEC standards, see:

http://www.iso.ch/

ISO 10646 and Unicode
ISO/IEC 10646 (ISO 10646) was defined to encompass the most of the world’s writing systems. It defines 2 bytes fixed length code (Universal Multiple Octet Coded Character Set: UCS-2) and 4 bytes fixed length code (UCS-4). Unicode characters are usually shown as U+xxxx, where xxxx is the hexadecimal code of the character.

Each character of UCS-2 is 2 bytes (16 bits) wide, regardless of the language. While the resulting 65000 code elements are sufficient for encoding most of the characters of the major languages of the world, the Unicode standard also provides an extension mechanism that allows the encoding of as many as one million more characters to completely support...
the characters of all languages over the world. This extension reserves a range of code values (U+D800 to U+D8FF, known as surrogates) for encoding some 32-bit characters as two successive code elements. UCS-2 is identical to Unicode without surrogates, while UCS-4 is identical to Unicode with surrogates.

DB2 UDB supports UCS-2; that is, Unicode without surrogates.

**UTF-8**

With UCS-2 or Unicode encoding, ASCII and control characters are also two bytes long, and the lead byte is zero. This could be a major problem for ASCII-based applications and file systems, because in a UCS-2 string, extraneous NULLs can appear anywhere in the string.

For example, the uppercase “A” is represented by U+0041, while in ASCII this byte stream is represented by X’00’ X’41”, which applications recognize as ‘NULL+A’.

A transformation algorithm, known as UTF-8, can be used to circumvent this problem for programs that depend on ASCII code.

UTF-8 (UCS Transformation Format 8) is an algorithmic transformation that transforms fixed length UCS-2 or UCD-4 characters into variable-length byte strings. In UTF-8, ASCII characters are represented by their usual single-byte codes, but non-ASCII characters in UCS-2 become two or three bytes long. In other words, UTF-8 transforms the UCS-2 characters into a multi-byte code set, for which ASCII characters are invariant.

The number of bytes for UCS-2 characters in represented in UTF-8 format can be determined using Table 10. In this table, each x or y or z represents each bit of the character, that is, 1 or 0.

<table>
<thead>
<tr>
<th>UCS-2 (hex)</th>
<th>UCS-2 (binary)</th>
<th>UTF-8 (binary)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 to 007F</td>
<td>00000000 0xxxxxxx</td>
<td>0xxxxxxx</td>
<td>ASCII</td>
</tr>
<tr>
<td>0080 to 07FF</td>
<td>00000xx xyyyyyy</td>
<td>110xxxxx 10yyyyyy</td>
<td>up to U+07FF</td>
</tr>
<tr>
<td>0800 to FFFF</td>
<td>xxxyyyy yyyyyyy</td>
<td>1110xxx 10yyyyyy 10zzzzzz</td>
<td>other UCS-2</td>
</tr>
</tbody>
</table>

For more information about ISO/IEC standards and Unicode, see:

http://www.iso.ch/
http://www.unicode.org/
4.1.2 Code pages and CCSID

A code page is a specification of code points from a defined encoding structure for each character in a collection of one or more character sets. A character set can be assigned to more than one code page, depending on the way the code point of each character is specified.

CCSID (Coded Character Set IDentifier) is defined in CDRA (Character Data Representation Architecture) to provide the means by which the character associated with a code point can be determined unambiguously. It provides information about the encoding scheme, coded character set, and code page. Therefore, once we specify the CCSID, the code character set and the code page of the environment is determined. For more information about CDRA, see Character Data Representation Architecture Overview, GC09-2207-00, and Character Data Representation Architecture Reference and Registry, SC09-2190-00.

4.2 National Language Support (NLS) provided by DB2 UDB

This section contains information about the NLS provided by DB2, including information about supported countries, languages, and code pages, and how to configure and use DB2 NLS features in your database and application.

Table 21 in Appendix D, “Country codes / code pages supported by DB2 UDB” on page 289 shows the languages and code sets supported by the database servers, and how these values are mapped to country code and code page values that are used by the database manager. Since DB2 XML Extender is now available on the following operating systems, AIX, Sun Solaris, Linux, and Windows NT, the table shows information on these operating systems only.

Detailed information on these topics can be found in the IBM DB2 Universal Database Administration Guide, SBOF-8934.

4.2.1 Deriving code page values

**Database code page**

You can determine the database code page by specifying it (explicitly or by default) at the time of creating the database. The following command creates a database of the name DB805 specifying explicitly code set ISO8859-1 and territory US:

```
db2 create database DB850 using codeset IBM-850 territory US
```
The list of supported code sets and territories is shown in Table 21 on page 290. If the code set and territory is not specified, the default value derived from the operating system locale is set.

If you do not specify the code page value, databases are created in the code page of the application creating them. The way to determine the code page of an application is described in the next section.

**Application code page**
The application code page is derived from the active environment when the database connection is made. DB2 will determine the appropriate code page value from the operating system. For example, the following defines how the active environment is determined in different operating environments.

**UNIX:**
On UNIX based operating systems, the active environment is determined from the locale setting, which includes information about language, territory and code set.

**Windows 32-bit operating systems:**
For all Windows 32-bit operating systems, if the DB2CODEPAGE registry variable is not set, the code page is derived from the ANSI code page setting in the Registry.

Alternatively, it is possible to determine the code page value manually to set the DB2CODEPAGE registry variable. You should be careful to set this variable correctly. Setting the DB2CODEPAGE registry variable to incorrect values may cause unpredictable results.

The following command sets the DB2CODEPAGE variables to 850:

```
db2set DB2CODEPAGE=850
```

Note that when using this method, all variables and files are the specified code page.

**Locales on UNIX and Windows**
Locales are implemented one way on Windows and another way on UNIX based systems. There are two locales on UNIX based systems:

- The environment locale allows you to specify the language, currency symbol, and so on, that you want to use.
• The program locale contains the current language and currency symbol. On Windows, cultural preferences can be set through Regional Settings on the Control Panel. However, there is no environment locale like the one on UNIX based systems.

**How to determine the code page value of the created database**

There are two ways to determine the code page value of the created database. One way is to list the database configuration parameters and see the *Database code page*. For example, the command to list the code page of the DB850 database is as follows:

```
db2 get db cfg for DB850
```

The resulting output contains the following line:

```
Database code page = 850
```

The second way to show the code page of the created database is to connect to the database using DB2 Command WIndow (Windows operating system) or terminal (UNIX) with “-a” option. The option “-a” displays SQLCA information. For example, the SQL statement to connect to the DB850 database is as follows:

```
db2 -a connect to DB850 user <user> using <password>
```

In this statement, `<user>` is a proper userid to connect the database, and `<password>` is the password. If the connection is successfully established, the SQLCA message shows information about the connection (output of the connect statement) as follows:

```
Database Connection Information

Database server = DB2/6000 7.1.0
SQL authorization ID = DB2V7
Local database alias = DB850

SQLCA Information

sqlcaid : SQLCA  sqlcabc: 136  sqlcode: 0  sqlerrml: 47
sqlerrmc: 81 850 USERID DB943 QDB2/NT 1 1 0 850 0
sqlerrxp : SQL07010
sqlerrd : (1) 1 (2) 1 (3) 1 (4) 1 (5) 0 (6) 0
sqlwarn : (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)
sqlstate: 00000
```
Look at the `sqlerrmc` field. The second token of this field is the code page value of the server, and the ninth token is the value of client. In this example, the tokens are both 850, so the code page value of this database is 850 and the code page value of the client is also 850. For more information, see “Connect (Type1)” in the *IBM DB2 Universal Database SQL Reference*, SBOF-8933.

### 4.2.2 Conversion between different code pages

For the best performance, your applications should use the same code page as your databases. However, this is not always possible or practical. The DB2 product has the support for character conversion that allows your application and database to use different code pages.

The supported code pages by DB2 are listed in Table 21 on page 290. In this table the values under the heading “Group” can be used to determine where conversions are supported. Matching combinations of any two values show where connection and conversion is allowed by DB2. When you are to connect to the database from a client which is not in the same group as that of database, the connection will fail with an error, SQLCODE -332.

Any code page can be converted to any other code page that is listed in the same IBM-defined language group. For example, code page 437 can be converted to 37, 819, 850, 1051, 1252, or 1275, because they are all in the S-1 group.

Character conversion can occur when an application accessing a database is running in a different code page from that of the database. This conversion will always occur on the database server machines.

Let us look at the example in Figure 57. In this example, the client connecting to the server issues the `INSERT` SQL statement to the columns, `col1` and `col2`, of the table, `table1`, in a database, `database1`. The value to be inserted is a number, 1, and a string, ‘string’. Since the code page value of the client is 1252, the code page value of this ‘string’ is 1252. When this client connects to `database1`, the database manager recognizes that the character code page conversion is necessary for this client, because the server code page value (database code page value) is 850, and it is different from that of the client. Then the `INSERT` SQL statement is issued. The code page value of the ‘string’ is converted to the code page 850 by DB2 and stored in `table1`.
The reverse conversion occurs when the data comes from the server to the client. Let us consider the example in Figure 58. In this case, the client retrieves the data from the table using SELECT SQL statement in the condition of “col1 = 1”. The data to be retrieved is ‘string’, which we inserted above. When the SQL statement is issued, DB2 converts the encoding of this ‘string’ from 850 to 1252, then the client gets the data.

Figure 57. Data converted from client code page to server code page

Figure 58. Data converted from server code page to client code page
The next example is more complex. Let us consider Figure 59. In this example the client searches the string, 'string', in the column, col2, and retrieves the data. The code page value of 'string' for the SQL statement in the client is 1252. When the SQL statement is issued, the code page of 'string' is converted to 850 in the server so that it can be matched to the 'string' in the database. Then the resulting strings are converted from code page 850 to 1252 in the server, and the client gets the strings in code page 1252.

![Figure 59. Data converted in both directions](image)

When the character code conversion occurs, it is possible that one or more characters are not represented in the target code page. If this occurs, DB2 inserts a substitution character into the target string in place of the character that has no representation. The replacement character is then considered a valid part of the string. In situations where a substitution occurs, the SQLWARN10 indicator in the SQLCA is set to “W”.

In the following cases, the character code conversions will not occur.

**File names:**

You should either use the ASCII character set for file names or provide the file name in the hexadecimal values that are physically stored in the file system. Note that if you include a file name as part of an SQL statement, it gets converted as part of the statement conversion.
FOR BIT DATA or BLOB attributes:

These considerations apply to data that is targeted for, or comes from, a column assigned the FOR BIT DATA attribute, or data used in an SQL operation whose result is FOR BIT or BLOB data. In these cases, the data is treated as a byte stream and no conversion occurs. For rules regarding unequal code page, as well as assigning, comparing, and combining strings, see the IBM DB2 Universal Database SQL Reference, SBOF-8933.

For more information about character code page conversion, see the IBM DB2 Universal Database Application Development Guide, SC09-2949.

4.2.3 The implementation of Unicode in DB2 UDB

Within IBM, the UCS-2 code page has been registered as code page 1200. While UTF-8 code page has been registered as code page 1208. When new characters are added to the standard, the code page number (1200 or 1208) will not change. Code page 1200 always refers to the current version of Unicode/UCS-2 and has been used for UCS-2 support in DB2 UDB. Code page 1208 is always used as the multi-byte code page number for a UCS-2/UTF-8 support by DB2.

When a database is created in UCS-2/UTF-8, the character data is basically stored in UTF-8 (code page 1208). However the graphic string data is stored in UCS-2 (code page 1200). That is CHAR, VARCHAR, LONG VARCHAR, and CLOB data are stored in UTF-8, and GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB data are stored in UCS-2. We will simply refer to this Unicode database as a UCS-2 database.

Creating a UCS-2 database

By default, the database is created in the code page of the client creating the database. You can create UCS-2 database from a UTF-8 client (For example, the Universal locale of AIX), or if the DB2CODEPAGE registry variable is set to 1208, your database is created as code page 1208.

Note that the client code page is 1208, not 1200, when you use a Unicode client. You cannot create a UCS-2 client. If you set UCS-2 as DB2CODEPAGE registry variable, you will get an SQL1204N error as follows:

```
SQL1204N The code page "" and/or country code "" is not supported by the installed version of the database manager.
```
Alternatively, you can create a UCS-2 database explicitly specifying “UTF-8” as the CODESET and use any valid two-letter TERRITORY code supported by DB2 UDB. For example, the following command is creating a UCS-2 database “UNICODEDB” with TERRITORY code for US:

```
db2 create database UNICODEDB using codeset UTF-8 territory US
```

Note that you have to specify UTF-8 (not UCS-2) as CODESET when you create a UCS-2 database. If you specify UCS-2 as CODESET, you will get an SQL1205 error as follows:

```
>db2 create db UNICODEDB using codeset UCS-2 territory US
SQL1205N The code page "1200" and/or country code "1" that has been specified is not valid.
```

A UCS-2 database allows connection from every single-byte and multi-byte code page supported by DB2 UDB. Code page character conversions between the client’s code page and UTF-8 are automatically performed by the database manager. Data in graphic string types is always in UCS-2, and does not go through code page conversions. The command line processor (CLP) environment is an exception. If you select graphic string (UCS-2) data from the CLP, the returned graphic string data is converted (by the CLP) from UCS-2 to the code page of your client environment.

Connection of a UTF-8 client (code page 1208) to a non-Unicode database is possible, but is not supported. If you attempt such a connection, the character code page conversion can cause unpredictable results.

When characters are converted from a local code page to UTF-8, there may be expansion in the number of bytes. The \( n \) characters can be between \( n \) bytes and \( 3n \) bytes, depending on the ratio of ASCII and non-ASCII characters. There is no expansion for ASCII characters, but other UCS-2 characters expand by a factor of two or three. The number of bytes of each UCS-2 character in UTF-8 format can be determined from the data in Table 10 on page 137.

### 4.3 Code page considerations using DB2 XML Extender

This section contains code page considerations, supported code page settings, and some hints and tips for using XML Extender with DB2 UDB. Code pages supported by DB2 XML Extender are listed in Table 21 in Appendix D, “Country codes / code pages supported by DB2 UDB” on page 289.
4.3.1 Basic rules

When a client receives an XML document from or sends an document to a database, the database manager 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, the database manager converts the character encoding in the XML document to match that of the database, as we have mentioned previously. As a result, it may happen that the actual XML document code page is not consistent with the encoding declaration when you retrieve that document.

When executing DB2 XML Extender stored procedures, or either of the dboxadm enable_column or dboxadm enable_collection commands, the encoding of the XML document or DAD file must be in the code page of the client. When executing UDFs to import XML documents or portions thereof into the database, the document must encoded in the code page of the server on which DB2 is running (which may not necessarily be the code page of the database). UDFs which produce or update XML documents and place the document at a user-specified file name, will encode the document in the code page of the DB2 server. In this case, the document's encoding declaration may not reflect its actual code page.

Note that Windows NT does not support a UTF-8 locale; therefore it is impossible to produce UTF-8 documents with Extenders UTFs on a Windows NT server. Setting the registry variable DB2CODEPAGE only affects the code page setting of the client, not that of the DB2 server. Therefore, while it is possible to create a UTF-8 database on a Windows NT server by changing DB2CODEPAGE appropriately, this setting does not enable UDFs to produce UTF-8 documents.

Therefore, we strongly recommend that all clients and databases of a server have the same code pages to avoid the need for code page conversions when you use DB2 XML Extender.

Moreover, there is a limitation for the current version of the code of DB2 XML Extender. The current version of XML Extender works properly only when the client code page, database code page, and OS code page of the server machine are all equal, as shown in Figure 60. Since Windows NT does not have the same Unicode locale as the OS locale, you cannot use XML Extender with a Unicode database on Windows NT.
4.3.2 Encoding declaration of an XML document

The encoding declaration declares the encoding of the character data in the XML document. It is described in the XML declaration statement explicitly as follows:

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

The string after “encoding=” is the character encoding value. The character encoding values to be declared are defined in the XML recommendation. Some of these are UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-USC-4, ISO-8859-1, Shift_JIS, and so on. The default value of encoding declaration is UTF-8, therefore if you do not specify the encoding value, it means the encoding value is UTF-8. Although UTF-8 and UTF-16 are supported as default encoding value by XML processor, the support of other encoding values is dependent on the XML processor.

DB2 XML Extender does not check the encoding declaration of the document. The database manager checks only the code page of the client, and converts the character data if necessary (see 4.2.2, “Conversion between different code pages” on page 141). Therefore if the client code page and the server code page are different, it can happen that the actual code page of the document is converted to the other code page, which the encoding declaration of the document is not consistent with (shown in Figure 61). This may especially be a problem when you use XML Columns.
It is important to ensure the actual encoding value of an XML document to be consistent with the encoding declaration. When the client code page and the database code page is the same, the inconsistency will not occur.

![Diagram](image)

Figure 61. Encoding declaration and actual encoding inconsistency

### 4.3.3 Considerations for XML Columns and XML Collections

DB2 XML Extender handles the encoding of a document as described below.

**Inserting the document into an XML Column**

It is important to match a client code page and a document encoding value when you retrieve the document from an XML Column.

As we see in 4.3.2, “Encoding declaration of an XML document” on page 147, DB2 XML Extender does not check the encoding declaration of the document. When the code page of your client application is 943 (Shift_JIS) and you insert a document whose encoding declaration is Shift_JIS into a code page 1208 (UTF-8) database, the document code page is converted to 1208, but encoding declaration of the document is not changed. If another code page 1208 client retrieves the document, then the document code page is not converted, and the actual encoding and the encoding declaration of the document are not consistent.

**Decomposing the XML document to an XML Collection**

The data stored in the XML Collection is the content and values that are associated with element and attributes, respectively. The encoding declaration of the document is not stored in the database. The code page of the data stored in the XML Collection is the same as that of the database.
Composing the XML document from an XML Collection
The DB2 XML Extender creates XML documents in the database code page and copies the encoding declaration specified in the DAD file. This document is stored in the use-specified result table. When this document is retrieved from the result table, DB2 converts the document to the client’s code page, while the documents encoding declaration remains unchanged.

4.3.4 Setting the proper code pages
Suppose that using DB2 XML Extender with servlets as shown in Figure 62. The client of DB2 UDB is the servlet. It connects to the database to decompose an XML document to the table, and to compose an XML document from the data in the tables.

![Diagram](image.png)

Figure 62. Using XML Extender with servlets

In this case we need to consider the client code page, the OS locale of the database server, the database code page, and the document encoding declaration.

Now we consider setting the client code page, the OS locale, the database code page and the document code page, which are all Unicode (UTF-8).

**Client code page**
The client code page can be specified explicitly to set the DB2CODEPAGE registry variable to the proper code page value. This is described in Chapter 4.2.1, “Deriving code page values” on page 138. Now, to set the client code page to UTF-8, we have to set DB2CODEPAGE to 1208 as follows:

```sql
db2set DB2CODEPAGE=1208
```
OS locale

The way to set the OS locale of the database server is dependent on the operating system. For example, to set the OS locale of an AIX operating system to UTF-8, first we should confirm that the Unicode locale is installed in the system, using the `locale` command with the `-a` option, as follows:

```
locale -a
```

If the resulting output contains the Unicode locale, you can set the OS locale to Unicode. For example, the resulting output of Unicode locale on Japanese AIX is as follows:

```
$ locale -a
C
POSIX
Ja_JP
Ja_JP.IBM-943
Ja_JP.IBM-932
ja_JP
ja_JP.IBM-eucJP
JA_JP
JA_JP.UTF-8
```

In this output, JA_JP and JA_JP.UTF-8 is the Unicode locale on Japanese AIX. Note that JA_JP is a short name and JA_JP.UTF-8 is a long name.

To set the OS locale to the Unicode locale, set the LANG variable as follows:

```
$ export LANG=JA_JP.UTF-8
```

We can confirm the OS locale by using the `locale` command, as follows:

```
$ locale
LANG=JA_JP.UTF-8
LC_COLLATE="JA_JP.UTF-8"
LC_CTYPE="JA_JP.UTF-8"
LC_MONETARY="JA_JP.UTF-8"
LC_NUMERIC="JA_JP.UTF-8"
LC_TIME="JA_JP.UTF-8"
LC_MESSAGES="JA_JP.UTF-8"
LC_ALL=
```
**Database code page**
The database code page is specified when we create the database. To create a database in code page 1208 (UTF-8), we issue the following command:

```
DB2 create database DBname using codeset UTF-8 territory Tr
```

In this command, `DBname` is the name of the database to be created, and `Tr` is the proper territory name, for example, US, JP, and so on.

Alternatively, we can create a database of code page 1208 when we set the `DB2CODEPAGE` registry variable to 1208 and just create the database (without specifying code set and territory). These are also described in 4.2.1, “Deriving code page values" on page 138.

**Document encoding**
Finally, we create an XML document to be stored in the above database in Unicode (UTF-8) and put the encoding declaration in the XML declaration as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Then we can work with this document to store it in DB2 UDB.

**Summary**
It is recommended that you prevent DB2 character code page conversions when it is possible that clients will use code pages which conflict with the declared encoding of the XML document. The following tips will help ensure that the XML document encoding is consistent with the client code page.

Before passing the document to an XML tool, use one of the following recommendations:

- Convert the document to the declared encoding code page.
- Use the registry variable `DB2CODEPAGE` to force the client code page to a known value. When using this method, ensure that all variables and files are the specified code page, not just the XML document.
- Use the `USING CODESET` optional parameter of the `DB2 CREATE DATABASE` command to force the database code page to a known value.
Integrating XML with DB2 XML Extender and DB2 Text Extender
Part 2. The XMLapp application
Chapter 5. XMLapp application: scenario

In this chapter, we introduce our sample application, XMLapp, which will be more deeply analyzed in the following chapters. In this application we use XML and DB2 XML Extender and we try to demonstrate how to combine them with other e-business technologies such as servlets, Java Server Pages (JSPs), and WebSphere or VisualAge for Java.

Our application is based on the XMLapp application described in the IBM Redbook, The XML Files: Using XML and XSL with IBM WebSphere 3.0, SG24-5749-00. We have reused part of that sample application, changing and extending it to show how such an application can benefit from DB2 XML Extender. We were especially concerned with a Business-to-Business (B2B) approach involving Business-to-Consumer (B2C) applications.

As DB2 XML Extender offers two methods of working with XML documents, XML Columns, and XML Collections, we decided to implement the same application scenario, once with XML Columns and once with XML Collections, so that we could better experience what the advantages and limitations of both methods were.

The design approach of the XMLapp application outlined in the following section is not as thorough as it would be in real life. However, it suits our purpose, which is to provide examples of how to use DB2 XML Extender in a realistic e-business application.

5.1 Application overview

The sample application that was implemented for this book, XMLapp, is a two-sided application featuring a customer side and a supplier side. Both sides exchange information in a B2B approach with XML documents, and each side is accessed in a C2B approach by either the customers or the suppliers.

The customer side is a simplified variation on the shopping-cart approach most common on the Web nowadays. Navigating through the application with a Web Browser, users can logon or register, perform a search on a product catalog, place purchase orders to suppliers, and track their current and past orders.
From a lower level perspective, we can outline the list of components which provide the above-mentioned functionalities:

- A main application menu, which lists the options available to the user
- A logon screen where the user authenticates himself/herself in order to get access to the rest of the system’s functionality
- A registration process where new users enter their details before they can access the system using the above logon screen
- A search mechanism which provides shoppers with a fast means to find items of interest, based on a keyword description
- A shopping basket which lists the items currently selected for purchase by the user, and also provides editing functionality for the user to modify their choices
- An ordering mechanism which creates purchase orders based on the user’s selection and sends them (as XML data) to the appropriate supplier
- An order tracking component allowing the customers to follow the shipping evolution of all their orders or the details concerning the shipping, product by product, of a particular order

You can also refer to Figure 63 for a visual description of how these components fit inside the overall application workflow.
The supplier side represents the suppliers workspace, where the suppliers can check the orders they have received, and can update these orders once they have shipped an ordered product. The information regarding the shipped products is then sent back as XML data to the customer side so that the customers can follow the evolution of their orders.
The workflow for the supplier side is shown in Figure 64. The following components are used:

- A main menu where each supplier can find the orders he has received.
- A screen displaying the details of each order from a supplier perspective and where the supplier can indicate which products have been shipped.
- A screen where the pending information concerning the shipping of the products can be displayed and is sent back to customers.

![Figure 64. XMLapp — supplier side workflow](image-url)
5.2 Chapters describing the XMLapp application

Table 11 summarizes the various chapters describing the XMLapp application, and provides a brief description of their content.

Table 11. Chapters covering XMLapp

<table>
<thead>
<tr>
<th>Chapter</th>
<th>XML method</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6</td>
<td>XML Columns</td>
<td>Description of the XMLapp application and the technologies used</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>XML Collections</td>
<td>Description of the XMLapp application and the technologies used</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>XML Columns</td>
<td>How to install XMLapp in WebSphere Application Server</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>XML Collections</td>
<td>How to install XMLapp in WebSphere Application Server</td>
</tr>
<tr>
<td>Appendix A</td>
<td>XML Columns</td>
<td>How to install XMLapp in VisualAge for Java</td>
</tr>
</tbody>
</table>
In this chapter, we provide a technical description of each logical module of the application. We start with the customer side before considering the supplier side, as the customers must have first ordered products before the suppliers can ship them.

Where are XML Columns used?
For a quick look at implementations of the XML Columns method for this application, you can concentrate on the following sections in this chapter:

- Section 6.4.5.3, “The placeOrder() method” on page 186 and Section 6.6.3.4, “The servlet SupplierStorageServlet” on page 206
- Section 6.5.1, “The servlet BrowseOrderServlet” on page 192 and Section 6.6.3.1, “The servlet BrowseSupplierServlet” on page 205
- Section 6.5.2, “The servlet CustomerStorageServlet” on page 197 and Section 6.6.3.2, “The servlet UpdateServlet” on page 205

6.1 Application and database architecture
In a real-life implementation of our XMLapp application, the customer side application would be hosted on one machine, and each supplier would have his own supplier side application. To keep the situation simple, and to make it possible for you to reproduce our environment easily, we decided to host both sides of the application (including many sides of the application with the customer side and each supplier side) on the same machine. This decision does not remove any interest or complexity, nor does it change anything in the logic of the whole application. The only practical difference is that the URLs we are using within our application to communicate between both sides, and in the browsers to access both sides, will always point to the same machine.

As a consequence, and with the same purpose of simplicity and ease of implementation, we store all the persistent data from our sample XMLapp application in database tables from one single database, called L301. The tables were designed with a basic approach in mind, with columns clearly labeled and constraints kept to a minimum. One may notice that, in some places, the use of constraints and triggers might have been justified. However, in cases where implementing these features might have complicated the issue and distracted us from our initial purpose, we made a conscious decision to do away with them.
Since this redbook focuses on the use of XML and DB2 XML Extender and not on Internet security matters, we decided to simply store all customer information, including the logon information (userid and password) in one table, USERPROFILE. This is certainly not the recommended approach in a real-life implementation.

Information regarding the items available to order for the customers is stored in the PRODUCTS table.

Each order is stored as an XML Column in the table ORDERXML. The table ORDERXML1 is a side table created for the table ORDERXML created by extracting the user name of the customer that submitted the order so that both ordernumber and username are directly accessible in this table.

On the supplier side, each order received is stored as an XML Column in the table SUPPLIER_STORAGE. The pending information concerning the shipped products is placed in the SUPPLIER_SHIPPING table.

For more information regarding the table design, refer to the Data Definition Language (DDL) documents referred to in Chapter 8, “XMLapp: installing XML Columns version in WAS” on page 249, describing how you can install the XMLapp application on your system.

Table 12 provides a quick reference to each module in the application.

<table>
<thead>
<tr>
<th>Module</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logon and registration</td>
<td>6.2, “Logon and registration module” on page 162</td>
</tr>
<tr>
<td>Search engine</td>
<td>6.3, “The search engine” on page 173</td>
</tr>
<tr>
<td>Shopping basket and ordering subsystem</td>
<td>6.4, “The shopping basket and the ordering subsystem” on page 177</td>
</tr>
<tr>
<td>Order tracking</td>
<td>6.5, “The order tracking function” on page 190</td>
</tr>
<tr>
<td>Supplier side</td>
<td>6.6, “The supplier side” on page 201</td>
</tr>
</tbody>
</table>

6.2 Logon and registration module

The first time a user or customer wants to access the XMLapp application, he must register before being able to order any product. If a customer is already registered, he needs to logon using the userid and password given during the registration.
6.2.1 The user interfaces

The XMLapp application starts with the main menu shown in Figure 65.

Clicking on the **Register** hyperlink brings you to a form where you can enter your registration information and submit it to the registration process (a servlet called RegistrationServlet), as shown in Figure 66.
After having successfully submitted your registration information, you receive the screen shown in Figure 67 that confirms your registration.
Figure 67. Registration successful screen

Now that you are registered, the next time you want to access the application, you must logon using the Logon required screen shown in Figure 68.
These are the main interfaces making up the logon subsystem.

As you may have noticed in Figure 65 on page 163, the URL requested by the browser (http://banks/xmlapp/xml/MainMenu.xml) actually points to an XML file. This means that the browser is requesting an XML file, but receives an HTML file (otherwise, it would not have displayed so nicely in our browser!).

We explain how this is done in the following section.

6.2.2 Automatically applying an XSL stylesheet to an XML document

Receiving an HTML document when asking for an XML document is realized by configuring WebSphere to automatically apply XSL stylesheets to XML. To do that, we use a servlet called DefaultApplyXSL, which is available from the LotusXSL download site, either as part of LotusXSL or as a separate download file. DefaultApplyXSL has three modes of operation:

1. It can be called with request parameters which specify the XML document to be transformed. For example:

2. It can be called as an alias, which means that it will automatically present static XML files stored in the Web server’s path using an XSL stylesheet. For example, if http://server/book.xml is the URL of a static XML file, then you can configure http://server/xmlapp/book.xml to be the same file with the XSL stylesheet applied.

3. It can be called as a mime-filter, which means that it will intercept any dynamically generated output of a given mime-type, and apply the relevant XSL stylesheet. For example, you can configure all text/xml output to have an XSL stylesheet applied to it.

For the servlet DefaultApplyXSL to know how to transform the XML document, you need to define a processing instruction (PI) which associates a stylesheet with the XML. This is defined by the W3C Recommendation “Associating Style Sheets with XML documents” (see http://www.w3.org/TR/xml-stylesheet/). For example, the following PI would associate the sample.xsl XSL stylesheet with the XML document:

```xml
<?xml-stylesheet type="text/xsl" href="sample.xsl" ?>
```

Using aliases and mime-filters, you can develop a completely XML-based application, using static XML files, XML based JSPs, and servlets which generate XML. These can be used together with multiple XSL stylesheets to create a front-end. Other front-ends can be built for different client types, for example, Wireless Markup Language for mobile phone browsers.

In our XMLapp application, we use the mime-filter approach of the DefaultApplyXSL servlet. As a result, each time our browser requests an XML file with our Web application path in its request, this XML file is sent automatically to the DefaultApplyXSL servlet that transforms it into HTML, with the XSL stylesheet referred to in its processing instruction xml-stylesheet.

For more information on exactly how to configure the MIME-TYPE in WebSphere Application Server for the DefaultApplyXSL servlet, please refer to 8.3.2.4, “Creation of the DefaultApplyXSL servlet” on page 254.

In the following sections, we describe the XML document used for the MainMenu of the application and its associated XSL stylesheet. The principle used for the other screens of this logon and registration component is similar to the one described in the following section.

### 6.2.3 The XML and XSL files used for the user interface

The following XML document (MainMenu.xml) is used for the main menu of the application.
As you can see, the processing instruction `xml-stylesheet` refers to the LogonForm.xsl file. This file indicates to the DefaultApplyXSL servlet how to transform the XML document into an HTML one. The coding for LogonForm.xsl follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xml:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0" type="text/xsl" href="/xml/LogonForm.xsl"/>
<SCREEN>
  <HEADER>Main Menu</HEADER>
  <TITLE>Main Menu</TITLE>
  <PARAGRAPH>
    <HEADER>Menu for testing ITSO XMLapp</HEADER>
    <TEXT>************************************************
    <TEXT><HREF TARGET="/xmlapp/xml/LogonForm.xml">Logon</HREF></TEXT>
    <TEXT><HREF TARGET="/xmlapp/xml/RegisterForm.xml">Register</HREF></TEXT>
    <TEXT><HREF TARGET="/xmlapp/xml/SearchStart.xml">*** Search our
    product catalog ***</HREF></TEXT>
    <TEXT><HREF TARGET="../jsp/UserQ.jsp">Check the registered
    users</HREF></TEXT>
    <TEXT>************************************************</TEXT>
  </PARAGRAPH>
</SCREEN>
```

As you can see, the processing instruction `xml-stylesheet` refers to the LogonForm.xsl file. This file indicates to the DefaultApplyXSL servlet how to transform the XML document into an HTML one. The coding for LogonForm.xsl follows:
In this XSL document, you can see that many templates are not used to transform our MainMenu.xml document. All these templates will be used for transforming other XML documents as this same stylesheet is used for the transformation of many XML documents and not only for our MainMenu.xml.
Definitely, this XSL stylesheet is needed to convert the XML document to a form appropriate for the client type that does not natively support XML, but this does not need to be HTML. In our XMLapp application, we only use HTML browsers as clients, and therefore, we only use one type of XSL stylesheet to convert to HTML. But we could as well have used another stylesheet that would have converted the XML data into a different output adapted to another type of client.

This approach has the advantage of concentrating all conversions in one place. Moreover, if we wish to include a background image in every HTML user interface, that can be defined in the stylesheet. After all, that is exactly the reason for using stylesheets: they control the representation, not the actual data, which is the responsibility of the XML document.

Having presented the user interfaces, we now discuss the servlets used to logon, register, or change the registration information in the next sections.

6.2.4 The servlet RegisterServlet

In order to restrict the number of connections to the database to a minimum while maintaining a good level of performance, all the servlets that need to access the database in our application do it by using connection pooling. Connection pooling is implemented in WebSphere 3.x through the use of datasources. These datasources allow us to get an already open connection from a pool of connections maintained by WebSphere.

The init method of the servlets that need to access the database is used to look up the datasource of our database L301 with JNDI. The init method of the registration servlet RegisterServlet is presented here and is commented upon in the following notes.

```java
public void init() throws ServletException{
    super.init();
    try{
        id = getInitParameter("userID");
        pw = getInitParameter("password");
    } catch (Exception e) {} 
    if ((id == null) || (pw == null)) {id = Constant.ID; pw = Constant.PW;}
    String providerURL = null;
    try {
        providerURL = getInitParameter("providerURL");
    } catch (Exception e) { 
    }
    if ((providerURL == null) || (providerURL.equals("")))
        providerURL = Constant.PROVIDER_URL;
```
// Obtain an initial context
Hashtable parms = new Hashtable();
parms.put(Context.INITIAL_CONTEXT_FACTORY,
Constant.INITIAL_CONTEXT_FACTORY);
parms.put(Context.PROVIDER_URL, providerURL);
try {ctx = new InitialContext(parms);}
catch (Exception e)
{ System.out.print("ERROR: Could not get Context from the
providerURL.");
  e.printStackTrace();
  return;
}
String datasourceJNDI = null;
// If an init parameter for the JNDI name has not been
// specified, use a default
try {datasourceJNDI = getInitParameter("datasourceJNDI");}
catch (Exception e) {}
if ((datasourceJNDI == null) || (datasourceJNDI .equals(""))
  datasourceJNDI = Constant.DATASOURCE_JNDI;
// Look up the datasource
try {
  ds = (DataSource)ctx.lookup(datasourceJNDI);
} catch (NamingException e) {
  System.out.print("init(): ");
  e.printStackTrace();
  throw new ServletException(e.toString());
}

Notes:

1 In an effort to rationalize the use of constants in our Java code and to
facilitate the configuration on another system, the most frequently used
constants like userid and password to connect to the database, the name of
the datasource and some URLs have been grouped in the Java class
itso.xmlapp.db2const.Constant. The name of the datasource, the userid, the
password and the provider_url can be passed to the servlet as init
parameters. If they are not defined as init parameters, the values defined is
the Constant class are used.

2 JNDI uses the notion of PROVIDER_URL and
INITIAL_CONTEXT_FACTORY in order to get an initial context from which
we can search and get, in our case, the needed datasource.

3 The name of the datasource placed in the variable datasourceJNDI must
correspond to the datasource defined in WebSphere Application Server to
access the database. Please refer to 8.3.2.1, “Creating a datasource and a JDBC driver” on page 253 to see precisely how to define our datasource L301 used to access the database L301. For example, to access the datasource L301 in the jdbc namespace jdbc, the value of the datasourceJNDI variable would be ‘jdbc/L301’. A namespace is a JNDI notion in which all names are unique.

We finally cast the object returned by the lookup method to a DataSource type. Our servlet keeps this datasource during its whole lifetime until being destroyed and each time it needs a Connection to the database, it can easily and rapidly get one by calling the getConnection(userid, password) method of this datasource.

When handling a registration request, the servlet performs two main actions:

- Inserting the registration information entered by the customer and received as parameter into the USERPROFILE table. The insert only succeeds if the username is not yet used.
- Creating a new HTTPSession and adding the name of the user in the session data.

### 6.2.5 The servlet LogonServlet

When handling a logon request, two actions are executed by this servlet:

- It compares the logon information provided by the user with the information stored in the database.
- If the provided information is valid, it establishes a new HTTPSession and puts the username into it, if the user has entered a valid username and password.

### 6.2.6 The servlet RegisterChangeServlet

At the difference of the LogonServlet and RegisterServlet, the RegisterChangeServlet cannot be called from a static HTML page built from a static XML document. As the registration information must first be pulled out of the database and displayed so that the user can change it, a dynamic method is required such as a JSP or servlet. In our implementation, the RegisterChangeOutputServlet takes care of that.

The RegisterChangeServlet performs two main tasks:

- It checks whether a session already exists to ensure that the user has passed through the logon procedure.
• If the session exists with a valid username, the old registration information is replaced by the new.

6.3 The search engine

First, the user needs to register to our Web application. Then, the user can order products. To select the products from a wide variety, the user needs an efficient search engine, which should allow the user to browse through the range of products of their interest, check the price, check a brief description, and order the item.

Upon selecting to search for products, the user is asked for an input string (keyword) that will be used to concentrate on that user’s interests. This user input is requested through a form that is based on a static XML document displayed using XSL. The user interface contains a text field to receive the input string from the user. This input string is received by a JSP, search.jsp, which is the heart of the search engine.

The JSP checks for the session to avoid searches by users who are not registered. If the search is invoked by an unregistered user, then the JSP redirects the user to the logon form. Otherwise, the JSP establishes the database connection.

The search string is sought in the database, looking for a match within product ID, product name, and product description. To be exact, the search string is matched to the values of fields (PRODUCT_ID, PRODUCT_NAME, and DESCRIPTION) of the table "products". The result of the search is an XML file. The JSP file for the search is as follows:

```xml
<?xml version="1.0"?>
<?xml-stylesheet href="http://localhost/xsl/result.xsl" type="text/xsl"?>

<jsp:directive.page contentType="text/xml" />
<tsx:dbconnect id="l301" userid="xmlguru" passwd="xmlguru" url="jdbc:db2:L301" driver="COM.ibm.db2.jdbc.app.DB2Driver">
</tsx:dbconnect>

<% String search = request.getParameter("searchstr");%>
<% search = search.trim();%>
<% search = search.toUpperCase();%>
<% HttpSession hs = request.getSession(false);
    if(hs == null){
```
//send back to logon screen
response.setContentType("text/html");
response.sendRedirect("/xml/LogonForm.xml");
}
try{
//check if username exists in session, else back to logon screen
Object un = hs.getValue("username");
if( un == null ){
    response.setContentType("text/html");
    response.sendRedirect("/xml/LogonForm.xml");
}
}
catch(Exception e){
    response.sendRedirect("/xml/LogonForm.xml");
}

<tsx:dbquery id="qry" connection="L301">
    select * from products where UCASE(PRODUCT_ID) LIKE '
    SELECT * FROMproducts WHERE UCASE(PRODUCT_ID) LIKE '
    UCASE(PRODUCT_NAME) LIKE '%<%= search %>%' OR 
    UCASE(DESCRIPTION) LIKE '%<%= search %>%' 
</tsx:dbquery>

<Product_list>
<tsx:repeat>
<Product PRODUCT_ID="<tsx:getProperty name="qry" property="PRODUCT_ID"/>">
    <Product_name><tsx:getProperty name="qry" property="PRODUCT_NAME"/>
    <Description><tsx:getProperty name="qry" property="DESCRIPTION"/>
    <Supplier_ID><tsx:getProperty name="qry" property="SUPPLIER_ID"/>
    <Supplier_PART_NO><tsx:getProperty name="qry" property="SUPPLIER_PART_NO"/>
    <Unit_price><tsx:getProperty name="qry" property="UNIT_PRICE"/>
    <Units_in_stock><tsx:getProperty name="qry" property="UNITS_IN_STOCK"/>
    <Minimum_stock_level><tsx:getProperty name="qry" property="MINIMUM_STOCK_LEVEL"/>
</竣工品>
The output XML document contains all the information regarding the products matching that user’s interests. This can be used by any other connecting applications. The XSL which renders this XML document uses the product ID, product name, description, and price. The user has the option to specify a quantity of the item and order it. The user can also start searching again. The last user input string is also displayed to enhance the search from the user’s point of view. The XSL file which transforms the XML document (output of JSP) is as follows:

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
  <xsl:output method="html"/>
  <xsl:template match="*">
    <HTML>
      <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
        <META http-equiv="Expires" content="0"/>
        <TITLE></TITLE>
      </HEAD>
      <BODY background="/xsl/world.jpg">
        <FORM action="orderbasket.jsp">
          <TABLE BORDER="2">
            <THEAD>
              <th>Product ID</th>
              <th>Product Name</th>
              <th>Description</th>
              <th>Price</th>
              <th>Qty.</th>
            </THEAD>
            <TBODY>
              <xsl:for-each select="PRODUCT_LIST/PRODUCT">
                <tr>
                  <td><xsl:value-of select="@PRODUCT_ID"/></td>
                  <td><xsl:value-of select="PRODUCT_NAME"/></td>
                  <td><xsl:value-of select="DESCRIPTION"/></td>
                  <td><xsl:value-of select="UNIT_PRICE"/></td>
                  <td><input type="text" name="_PNAME_{@PRODUCT_ID}" size="40" value=""></td>
                </tr>
              </xsl:for-each>
            </TBODY>
          </TABLE>
        </FORM>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```
Let us suppose that the user provides an input string “XML”. The result of the combined effort of the JSP and the XSL, which we described before, looks as shown in Figure 69.
Figure 69. Result of the search for user argument “XML”

The **SEARCH AGAIN** command invokes the JSP again. You can enter an empty search string to get all the available products.

The **ADD TO ORDER** option sends a request to the ordering sub-system. This sub-system is presented in the next section.

### 6.4 The shopping basket and the ordering subsystem

As we have seen in Figure 69, the user has the possibility to add the products returned by the search to the order. When the customer clicks on the **Add to Order** button, the system adds the selected items and their quantities into a “shopping basket”. The shopping basket is an object that must live in the HttpSession for the user — so that it can remain between different pages.

#### 6.4.1 The user interfaces

For example, based on Figure 69 on page 177, if the user chooses to add one product MK-001 and two products MK-002 to his order, the shopping basket content displays as shown in Figure 70.
When the customer has added some products in the order basket, he still has the possibility to change his mind and for instance decide to increase or decrease the quantities he wants to order. He can also remove some products from the shopping basket (by changing the quantity to zero), remove all the products from the basket (by canceling the order) or search again for other products that could be added to the basket. Thus, the online application is very flexible and works just like when you go shopping in a supermarket. The only difference is that, in place of going through the cashier, you have to place the order. After having placed an order, you receive the display shown in Figure 71.
When the order is submitted, the content of your shopping basket is written to the database as an XML document and sent to the suppliers so that they can ship you the required products.

The status of your order has changed; from being NEW or UNSUBMITTED before placing the order, it has now become PENDING. This order will remain pending until all the products you ordered are shipped to you by the suppliers. At that moment the order will become SHIPPED.

Now, if you want, you can continue shopping by searching new products and adding them to a new shopping basket. Each time a new shopping basket is created, a new and unique order number is assigned to it so that you can easily track the evolution of your order and the shipping of the products.

In the following sections, we go further into the technical details of this implementation.

### 6.4.2 Implementation of the shopping basket as a bean

The shopping basket in our application is implemented as a Java bean called itso.xmlapp.db2beans.Basket. This bean has a very simple interface, which means that it is suitable to be used by a JSP page. The bean allows a product quantity to be set (when you add a new product or change the quantity), the
order to be placed or cleared, and finally its content to be retrieved as an XML string. Figure 72 shows the architecture of the ordering subsystem.

![Figure 72. Overview of the architecture of the ordering subsystem]

The core of this subsystem is the Basket bean. This Basket bean has the public interface shown here:

```java
public Basket();
public void clearOrder();
public void setCustomerOrderID(String newCustomerOrderID);
public void setUserName(String newUserName);
public void setOrderStatus(String newStatus);
public void setProductQuantity(String prodID, String quantity);
public void placeOrder();
public String getOrderXMLString();
public String getOrderXMLString(String supplier);
```

### 6.4.3 Program flow

The Basket bean is always owned and managed by the orderbasket JSP. This JSP is called by the search results page, and it also calls itself with updated data. The interaction is shown in Figure 73.
6.4.4 The orderbasket.jsp file

The code of the orderbasket JSP file is presented here and is commented upon in the following notes.

```jsp
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="http://localhost/xsl/basket.xsl"?>
<jsp:directive.page contentType="text/xml"/>
<jsp:useBean id="basket" class="itso.xmlapp.db2beans.Basket" scope="session">

<%= String redirectpage = "/xml/MainMenu.xml";
    String username = (String)request.getSession(true).getValue("username");
    boolean orderplaced=false;
    String orderno=null;
    if (username == null)
    {
        response.setContentType("text/html");
        response.sendRedirect(redirectpage);
    }
    try {
```
basket.setUserName(username);

java.util.Enumeration en = request.getParameterNames();
while ( en.hasMoreElements() ) {
    String paramName = (String)en.nextElement();
    if (paramName.startsWith("_PNAME_")) {
        String prodid = paramName.substring(7);
        String quantity = request.getParameterValues(paramName)[0];
        if (quantity != null && !quantity.equals("")) {
            int q = 1;
            try {
                q = Integer.parseInt(quantity);
            } catch (Exception e) {} 
            basket.setProductQuantity(prodid, (new Integer(q)).toString());
        }
    }
    en = request.getParameterNames();
    while ( en.hasMoreElements() ) {
        String paramName = (String)en.nextElement();
        if (paramName.equals("CUSTID")) {
            basket.setCustomerOrderID(request.getParameterValues(paramName)[0]);
            basket.setOrderStatus("UNSUBMITTED");
            if (paramName.equals("PLACEORDER")) {
                basket.placeOrder();
                basket.setOrderStatus("PENDING");
                orderplaced = true;
            }
            if (paramName.equals("CANCELORDER")) {
                basket.clearOrder();
            }
        }
    }
} catch (Exception e) {
    out.println("<ERROR>");e.printStackTrace();out.println("</ERROR>"); }
%
<%= basket.getOrderXMLString() %>
<% if (orderplaced) basket.clearOrder(); %>

Notes:

The output that is generated by this JSP file is an XML document as indicated by 1. The XSL stylesheet basket.xsl (2) is associated with this XML
output. The transformation occurs automatically with our DefaultApplyXSL servlet associated to the XML type.

3 The JSP defines the Basket bean that it requires. By using the class attribute and not the type attribute to define the class of this bean, the JSP instantiates a new instance of the Basket class by calling its Basket() constructor if it is not found in the HTTPSession to which this bean is scoped. This is the case the first time the customer adds a product to the order. The following times this JSP is called during this same session, the bean is found in the scope of the session and used by the JSP. As we will see later, by calling the constructor of the Basket bean, a new and unique order number is created and the order status defaults to NEW.

4 The JSP extracts the name of the user from the session, and call the setUserName() method of the Basket bean to place this user name in the basket. If the user name is not available in the session, the JSP redirects the user to the logon page.

7 Some parameter handling code is needed to extract the products and the quantities of these products desired by the customer. The JSP looks for parameters like _PNAME_PRODID=QUANT, where PRODID is a valid product code, and QUANT is a number.

8 For each of the products chosen by the user, the JSP calls the Basket setProductQuantity(PRODID, QUANT) method to pass it the product identifier and the quantity. The bean will either add the product if it is not already present or only change the quantity if the product is already in the basket.

9 The JSP looks for a parameter CUSTID= and if it is present, it calls setCustomerOrderID() on the Basket bean, and also updates the status to UNSUBMITTED. This CustomerOrderID notion is simply a possibility offered to the customer to enter his own code for the order in case he finds the generated order number to difficult to remember.

10 The JSP looks for a parameter PLACEORDER, which corresponds to the Place Order button. If this is present, it updates the status to PENDING and calls the basket placeOrder() method.

12 If there is a parameter CLEARORDER, which corresponds to the Clear Order button, the JSP calls the Basket clearOrder() method, which re-initializes the basket with a new empty order and a new order number, only keeping the current user name.
Finally, the JSP output is generated by getting the content of the basket as an XML document with the basket getOrderXMLString() method. The XSL stylesheet linked with this newly generated XML document is so conceived that it is smart enough to display PENDING orders differently than NEW or UNSUBMITTED orders. Details of the XSL stylesheet appear later in this chapter.

If the order has been placed it is finally cleared with the clearOrder() method. It was not possible to clear the order when placing it as we still needed the content in the basket to generate the XML output.

6.4.5 The Basket bean

Our itso.xmlapp.db2beans.Basket bean stores the information about the shopping basket in its instance variables. Its principal instance variables are shown here:

```java
private String orderStatus;
private String orderNumber;
private String orderDate;
private String userName;
private String customerOrderID;
private Vector products;
```

The products Vector contains instances of the helper class itso.xmlapp.db2beans.Product whose instance variables are displayed here:

```java
private int quantity;
private String description;
private String name;
private float price;
private String shipdate;
private String prodID;
private String supplierID;
private String supplierPartNo;
private int itemNo;
```

Now that the main actors have been introduced, we give an overview of the important actions that take place in the methods of our Basket.

6.4.5.1 The Basket() constructor and createEmptyBasket() method

The Basket() constructor initializes the bean by calling the createEmptyBasket() method to get a datasource, create an empty products Vector and set the order status to NEW and the order date to the date of the day.
6.4.5.2 The `setProductQuantity()` method

This method adds a new product (a new instance of the class `Product`) to the `products` vector when this product is not already listed in the vector. In case the product is already present in the vector, its quantity is updated or the product is removed if the quantity is zero.

The code of this method is presented here and is commented upon in the following notes.

```java
class XMLapp {
    public void setProductQuantity(String prodID, String quantity) {
        Product p;  
        String sup;  
        boolean prodAlreadyExisting = false;  
        boolean supAlreadyAdded = false;  
        int q;  
        try { q = Integer.parseInt(quantity); } catch (Exception e) { q = 1; }
        for (int i=0; i<products.size(); i++) {
            p = (Product) products.elementAt(i);  
            if (p.getProdID().trim().equals(prodID.trim())) {
                prodAlreadyExisting = true;  
                if (q==0) { products.removeElementAt(i); }
                else {
                    p.setQuantity(q);  
                    products.setElementAt(p, i); 
                }
            }
        }
        if (!prodAlreadyExisting) {
            String sql = "select product_name, description, supplier_id, supplier_part_no, unit_price from products where product_id = \"" + prodID.trim() + \"\"; 
            try {
                conn = ds.getConnection(id, pw);  
                Statement s = conn.createStatement();  
                ResultSet rs = s.executeQuery(sql);  
                if (rs.next()) { sup = rs.getString(3).trim();  
                    p = new Product(prodID, rs.getString(1).trim(), rs.getString(2).trim(), rs.getFloat(5), q, sup, rs.getString(4).trim());  
                    products.addElement(p);  
                    for (int j=0; j<suppliers.size(); j++) {
                        if (((String) suppliers.elementAt(j)).equals(sup)) {supAlreadyAdded = true;}}}
                if (!supAlreadyAdded) {suppliers.addElement(sup);}
                rs.close();  
            } catch (Exception e) {e.printStackTrace();}
        finally {
            try { if (conn != null) conn.close(); } catch (java.sql.SQLException s) {}  
        }
    }
}
```
Notes:

1. If the product information passed to the method refers to a product already existing in the products vector and if the new quantity desired by the customer is zero, this product is removed from the products vector and so from the shopping basket.

2. If the product information passed to the method refers to a product already existing in the products vector and if the new quantity desired by the customer is not zero, then the quantity of this product is updated in the products vector.

3. If the desired product is a new product not found in the products vector, then a new product is created with information retrieved from the PRODUCT table and added in the products vector.

4. To be more efficient in the method that will place the order, we store each supplier in a suppliers vector.

6.4.5.3 The placeOrder() method
This method performs two major tasks of the ordering subsystem:

- It writes the placed order as an XML document in an XML Column.
- It sends an XML document to each supplier to inform them about the ordering of their own products.

The code of this very important method is presented here and is commented upon in the following notes.

```java
public void placeOrder() {
    if (products.size()>0) {
        // 1. write the order xml document in the XML column in the table
        try {
            conn = ds.getConnection(id,pw);
            Statement s = conn.createStatement();
            String sql = "insert into orderxml values("
                          + '"' + getOrderNumber() + '"',"'
                          + getOrderXMLString() + "'")";
            s.executeUpdate(sql);
        } catch (Exception e) { e.printStackTrace(); }
        finally { try {
            // return the connection to the connection pool
            if (conn != null) conn.close();
        } catch (java.sql.SQLException s) {} }
        // 2. send to each supplier the part of the xml document regarding his products
        String order;
        URL supURL = null;
        for (int i=0; i<suppliers.size(); i++) {
            order = getOrderXMLString((String) suppliers.elementAt(i));
            if (order.length()>0) {
                try { supURL = new URL(Constant.POST_ORDER_TO_SUPPLIER_URL); }
                catch (Exception e) { e.printStackTrace(); }
                try { supURL.openConnection(); }
                catch (IOException e) { e.printStackTrace(); }
                try { supURL.getOutputStream().write(order.getBytes()); }
                catch (IOException e) { e.printStackTrace(); }
            }
        }
    }
}
```

186 Integrating XML with DB2 XML Extender and DB2 Text Extender
postToSupplier(order, supURL);

Notes:

1. The XML document containing the data about the placed order is generated by the getOrderXMLString() method that returns the XML document in a Java String. It is inserted in the ORDERXML table through a JDBC connection taken from the pool of connections with the datasource instantiated at the creation of the Basket bean.

2. The XML document to be sent to each supplier whose one of more product has been ordered is also generated by passing the name of the supplier (kept in the suppliers vector) to the overloaded getOrderXMLString() method.

3. This XML document must now be sent to the IT system of each supplier by a URL connection. As discussed when introducing this application, we restricted our application to one machine so that the URL (defined in the Constant class) points to our own server. However making a URL connection to another server or to the same server is just the same, it only requires that the resource you are asking for (in our case, a servlet) be there.

4. The sending operation is performed by the postToSupplier() method.

6.4.5.4 The getOrderXMLString() method

In our application, the XML document we store in the ORDERXML table and the XML document we send to the customer are defined by the same DTD. The only difference is that the XML document for the supplier contains only the products of this supplier whereas the XML document we store for the customer transaction contains all the products he ordered.

In a real-life application, it is unlikely that the internal XML document used will have the same format as the XML document used to exchanging the information with the suppliers but the only impact would be that the getOrderXMLString() method and the getOrderXMLString(String supplier) would be different while in our case, we are able to reuse the same code.

The DTD for our order XML document is presented here:

```xml
<?xml version="1.0"?>
<!ELEMENT order (order_no,status,customer_order_no,order_date,
ship_date,username,products)>
<!ELEMENT order_no (#PCDATA)>
<!ELEMENT status (#PCDATA)>
```
Integrating XML with DB2 XML Extender and DB2 Text Extender

```xml
<!ELEMENT customer_order_no (#PCDATA)>  
<!ELEMENT order_date (#PCDATA)>  
<!ELEMENT ship_date (#PCDATA)>  
<!ELEMENT username (#PCDATA)>  
<!ELEMENT products (product+)>
<!ELEMENT product (product_name, product_description, supplier_id, item_price, item_status, order_quantity, ship_date)>  
<!ATTLIST product item_no CDATA #REQUIRED  
 product_id CDATA #REQUIRED  
 supplier_part_no CDATA #REQUIRED>
<!ELEMENT product_name (#PCDATA)>  
<!ELEMENT product_description (#PCDATA)>  
<!ELEMENT supplier_id (#PCDATA)>  
<!ELEMENT item_price (#PCDATA)>  
<!ELEMENT item_status (#PCDATA)>  
<!ELEMENT order_quantity (#PCDATA)>  

If the customer had ordered the two products PRNT1234 and PAP02, the following document would be generated by the getOrderXMLString() method:

```xml
<order>
  <order_no>20000825221352279312000000</order_no>
  <status>NEW</status>
  <customer_order_no>ALPHA</customer_order_no>
  <order_date>2000-08-25</order_date>
  <username>TANGO</username>
  <products>
    <product item_no="1" product_id="IT-002" supplier_part_no="PRNT1234">
      <product_name>Color Printer (laser)</product_name>
      <product_description>Color Printer Laser 600 dpi</product_description>
      <supplier_id>SUPP01</supplier_id>
      <item_price>560.0</item_price>
      <item_status>NEW</item_status>
      <order_quantity>6</order_quantity>
    </product>
    <product item_no="2" product_id="ST-001" supplier_part_no="PAP02">
      <product_name>A4 White Paper</product_name>
      <product_description>White Paper in A4 format (200 sheets)</product_description>
      <supplier_id>SUPP03</supplier_id>
      <item_price>34.0</item_price>
      <item_status>NEW</item_status>
      <order_quantity>9</order_quantity>
    </product>
  </products>
</order>
```

The code of this method follows:

```java
private String getOrderXMLString(String supplierID) {  
  StringBuffer o = new StringBuffer();  
  boolean productAdded = false;
  int itemNo = 0;
  Product p;
  o.append("<order><order_no>" + getOrderNumber() + "</order_no>";
  o.append("<status>" + getOrderStatus() + "</status>";
  o.append("<customer_order_no>" + getCustomerOrderID().trim() + ">
"</customer_order_no>";
  o.append("<order_date>" + getOrderDate().trim() + "</order_date>";
  o.append("<ship_date>" + getShipDate().trim() + ""/ship_date>";
```
o.append("<username>"+getUserName() +"</username>");
o.append("<products>");
for (int i=0; i<products.size(); i++)
{  p = (Product) products.elementAt(i);
  if ((supplierID.length()==0) || (supplierID.equals(p.getSupplierID())))
  { o.append("<product item_no=" + ++itemNo + " product_id=" + p.getProdID().trim() + " supplier_part_no=" + p.getSupplierPartNo() + ">");
o.append("<product_name>"+p.getName() +"</product_name>");
o.append("<product_description>"+p.getDescription() +"</product_description>");
o.append("<supplier_id>"+p.getSupplierID() +"</supplier_id>");
o.append("<item_price>"+p.getPrice() +"</item_price>");
o.append("<item_status>NEW</item_status>");
o.append("<order_quantity>"+p.getQuantity() +"</order_quantity>");
o.append("<ship_date/>");
productAdded = true;
  }
}
o.append("</products></order>");
if (productAdded || (supplierID.length()==0)) {return o.toString();} else {return "";}
}

6.4.5.5 The postToSupplier() method

The code is of this method is presented here and is commented upon in the following notes:

private void postToSupplier(String XML, URL url) {
    StringBuffer sb = new StringBuffer("\n");
    try
    {
        String argString = URLEncoder.encode("xmldata") + "=" + URLEncoder.encode(XML);
        URLConnection con = url.openConnection();
        // Prepare for both input and output
        con.setDoInput(true);
        con.setDoOutput(true);
        // Turn off caching
        con.setUseCaches(false);
        // Work around a Netscape bug
        con.setRequestProperty("Content-Type","application/x-www-form-urlencoded");
        // Write the arguments as post data
        DataOutputStream out = new DataOutputStream(con.getOutputStream());
        out.writeBytes(argString);
        out.flush();
        out.close();
        java.io.BufferedReader in = new java.io.BufferedReader(new
        java.io.InputStreamReader(con.getInputStream()));
        while (in.ready())
        {  sb.append(in.readLine());
        }
in.close();
    } catch (IOException e) { e.printStackTrace(); }
}

Notes:

1 We know that the servlet of the supplier listening at the supplier URL expects the XML document to be passed as the value of the parameter xmldata.
We send the XML document to the supplier.

**6.4.5.6 The XSL stylesheet basket.xsl**

On our order XML document, the very same as the one sent to the supplier for B2B communication, we apply an XSL stylesheet and get our HTML page as shown in both Figure 70 on page 178 and Figure 71 on page 179.

The interesting aspects of the XSL stylesheet are:

- The stylesheet gives different HTML pages based on whether the order has been submitted or not. If the order status is *NEW* or *UNSUBMITTED*, then the form is enabled and users can modify the order. Once the order is *PENDING*, the user can only view the data, and a link is shown to view the status of this order in the order tracking subsystem. This is implemented using the `<xsl:if>` tags as shown here:

  ```xml
  <xsl:if test="(status = 'NEW' or status = 'UNSUBMITTED')">
    <H3>Enter 0 in quantity to delete item from order</H3>
  </xsl:if>
  ```

  In this case the HTML tag `<H3>` and its content is only produced if the element status in the order XML document is *NEW* or *UNSUBMITTED*.

- Another comparable `<xsl:if>` tag return another more appropriate HTML output if the status of the order is *PENDING*.

- The stylesheet calculates the total cost of each item by using simple arithmetic and XSL variables as illustrated here.

  ```xml
  <xsl:variable name="quant" select="order_quantity"/>
  <xsl:variable name="price" select="item_price"/>
  <xsl:variable name="total" select="$quant*$price"/>
  ```

  We first define the variables, quant and price, and their value. Then we create a new variable, total, based on the value of the previously defined variables. Finally, we can use the value of the new variable to create our HTML output.

**6.5 The order tracking function**

A customer can browse his orders either by clicking on the `Browse the orders in process` hyperlink on the main menu of the application after the logon screen or directly after having submitted a new order, by clicking on the
Track this order... hyperlink on the screen confirming that the order has been successfully submitted.

The customer first gets a list of all his orders as indicated in Figure 74.

![Past and Current Purchase Orders](image)

**Figure 74. Browsing the list of orders**

On this general screen, the customer can follow the evolution of all his orders and see what orders have already been shipped by the suppliers and when. An order is considered shipped when all the products have been shipped by the various suppliers.

But as one order groups products coming from different suppliers, it is unlikely that all products will be shipped the same day. Therefore, the customer can also follow the evolution in the shipment of each product making out his order by clicking on the hyperlink referencing the order number. Figure 75 displays the details of a particular order.
Integrating XML with DB2 XML Extender and DB2 Text Extender

6.5.1 The servlet BrowseOrderServlet

This servlet extracts the information it needs from the orders stored as XML Columns in the database to display the information shown in Figure 74 and Figure 75.

To browse the details of one order, we could have simply read the whole XML document from the table and applied it an XSL stylesheet especially designed for our order XML document. However, in order to better illustrate the ease and flexibility offered by DB2 XML Extender in the manipulation of XML documents stored as XML Columns, we chose to use a more difficult method.

In this application we are reusing an XSL stylesheet that was originally created for the redbook *The XML Files: Using XML and XSL with IBM WebSphere 3.0*, SG24-5479-00. This stylesheet expects an XML document with a structure different from our order XML document. Therefore, we must first extract the information contained in our order XML documents to create another XML document with the right format.
During the development of our BrowseOrderServlet, we have been happily surprised by the combined power and facility offered by DB2 XML Extender to extract any data from XML documents only by using SQL statements.

We now discuss three important methods of our BrowseOrderServlet:

- The performTask() method is used for both the doGet() and doPost() methods. It uses both of the following methods.
- The generateOrderList() methods generate the XML document with data about all the orders of one supplier.
- The generateOrderDetail() methods generate the XML document with data about one order.

The first code extract shown here comes from the performTask() method, and is commented upon in the following notes.

```java
String ssListURL = Constant.SSL_PAGE_PREFIX + "/xsl/orderlist.xsl";  // 1
String ssDetailURL = Constant.SSL_PAGE_PREFIX + "/xsl/orderdetail.xsl";
...
response.setContentType("text/html");
PrintWriter out = response.getWriter();
XSLTResultTarget rthtml = new XSLTResultTarget(out);
String orderNo = request.getParameter("orderno");
if (orderNo != null && orderNo.length() > 0) {
    URL ssURL = new URL(ssDetailURL);
    InputStreamReader reader = new InputStreamReader(ssURL.openStream());
    XSLTInputSource isxsl = new XSLTInputSource(reader);
    StringReader orderxml = new StringReader(generateOrderDetail(orderNo).toString());
    XSLTInputSource isxml = new XSLTInputSource(orderxml);
    XSLProcessor xslProcess = new XSLProcessor();
    xslProcess.process(isxml, isxsl, rthtml);
} else {
    URL ssURL = new URL(ssListURL);
    InputStreamReader reader = new InputStreamReader(ssURL.openStream());
    XSLTInputSource isxsl = new XSLTInputSource(reader);
    StringReader orderxml = new StringReader(generateOrderList(username).toString());
    XSLTInputSource isxml = new XSLTInputSource(orderxml);
    XSLProcessor xslProcess = new XSLProcessor();
    xslProcess.process(isxml, isxsl, rthtml);
}
out.close();
```
Notes:

This servlet does not use the DefaultApplyXSL servlet to transform the XML into HTML but it rather does the transformation itself. Based on the request received, this method:

- Gets the XML document by calling the right generateOrder... method
- Gets the appropriate XSL stylesheet for the XML document
- Transforms the XML with the XSL into HTML and outputs it

To execute the transformation in the servlet, we need two inputs: an XSLTInputSource for the XML document, another XSLTInputSource for the XSL stylesheet, and one output: an XSLTResultTarget for the HTML page.

We create the XSLTInputSource for the XML document based on the XML document String returned by the generateOrderList() or generateOrderDetail() method.

We create the XSLTInputSource for the XSL stylesheet based on the URL where this stylesheet can be found. When doing the transformation, the XSLProcessor will read from the input stream opened at this URL.

The XSLProcessor executes the transformation and directly returns the HTML to the result target, our browser.

The second code excerpt is the generateOrderList() method:

```java
public StringBuffer generateOrderList(String username) {
    Connection conn = null;
    StringBuffer xml = new StringBuffer("<ORDERSET>");
    String order_no = null;
    String customer_order_no = null;
    String ship_date = null;
    try{
        conn = ds.getConnection(id,pw);
        StringBuffer xml = new StringBuffer("<ORDERSET>");
        String order_no = null;
        String customer_order_no = null;
        String ship_date = null;
        try{
            conn = ds.getConnection(id,pw);
            java.sql.PreparedStatement ps1 = conn.prepareStatement("select order_no from orderxml1 where username=?");
            String ps2sql = "select db2xml.extractVarchar(orderdoc, '/order/customer_order_no'),db2xml.extractVarchar(orderdoc, '/order/order_date'),db2xml.extractVarchar(orderdoc, '/order/ship_date') from orderxml where order_no = ?";
            java.sql.PreparedStatement ps2 = conn.prepareStatement(ps2sql);
            ps1.setString(1, username);
            ResultSet rs1 = ps1.executeQuery();
            ResultSet rs2 = null;
            } catch (Exception e) {
                e.printStackTrace();
            }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return xml;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```
Vector order_no_V = new Vector();
while(rs1.next())
  {order_no_V.addElement(rs1.getString(1));}
for (int i=0;i<order_no_V.size();i++)
  {order_no = (String) order_no_V.elementAt(i);
   ps2.setString(1, order_no);
   rs2 = ps2.executeQuery();
   rs2.next();
   customer_order_no = rs2.getString(1);
   if (customer_order_no == null) {customer_order_no = "No number";}
   ship_date = rs2.getString(3);
   if (ship_date == null) {ship_date = "Not shipped";}
   xml.append("<ORDER ORDER_NO=" + order_no + " CUSTOMER_ORDER_NO=" +
   customer_order_no + " USER_NAME=" + username + " ORDER_DATE=" +
   rs2.getString(2) + " SHIP_DATE=" + ship_date + "/">" +
   ship_date + "/">" +
   xml.append("<ERROR>" + e.toString() + "</ERROR>");
   try {if (conn != null) conn.close();
   } catch (java.sql.SQLException s) {
   } finally {
   xml.append("</ORDERSET>" ); return xml; }

Notes:

1 This method returns an XML document with the tag <ORDERSET> as root element.

2 Our first access to read the XML documents is through the side table ORDERXML1. This table associates the order number and the user name for each order and it makes it easy for us to find all the existing orders related to the customer passed as arguments to this method. We store each order number in a vector.

3 For each order associated to the supplier and stored in the vector, we must extract from the order XML document all the info we need, namely the order number given by the customer, the order date and the ship date. All this extraction occurs in the SQL statement thanks to the extracting UDFs provided by DB2 XML Extender. Without these extracting UDFs, we would have to read the entire XML document, parse it in our Java program and write the needed code the extract each element or attribute value desired.
We build the new XML document to be returned with the information extracted from our orders XML documents.

The last part of code presented in this section comes from the `generateOrderDetail()` method. This code listing is commented upon in the following notes.

```java
public StringBuffer generateOrderDetail(String order_no) {
    String orderdoc = new StringBuffer("<ORDERDETAIL>");
    // one of the problem we face is retrieving the number of product in the order xml doc
    // we can't use the XPath function last() as it is not supported by DB2 XML Extender
    String ps1sql = "select db2xml.extractVarchar(orderdoc, '/order/customer_order_no')," + " db2xml.extractVarchar(orderdoc, '/order/username')," + " db2xml.extractVarchar(orderdoc, '/order/order_date')," + " db2xml.extractVarchar(orderdoc, '/order/ship_date')" + " from orderxml where order_no = ?;";
    try{
        conn = ds.getConnection(id,pw);
        java.sql.PreparedStatement ps1 = conn.prepareStatement(ps1sql);
        java.sql.Statement s2 = conn.createStatement();
        ps1.setString(1, order_no);
        ResultSet rs1 = ps1.executeQuery();
        String stmt2, ship_date, customer_order_no;
        ResultSet rs2;
        if (rs1.next())
            {customer_order_no = rs1.getString(1);
             if (customer_order_no == null) {customer_order_no = "No number";}
             ship_date = rs1.getString(4); if (ship_date == null) {ship_date = "Not shipped";}
             orderdoc.append("<ORDER ORDER_NO=" + order_no + " CUSTOMER_ORDER_NO=" + customer_order_no + " USER_NAME=" + rs1.getString(2) + " ORDER_DATE=" + rs1.getString(3) + " SHIP_DATE=" + ship_date + "/">;"
        }
        orderdoc.append("<ORDER_ITEM_LIST>");
        int item_no = 1;
        String item_no_s = Integer.toString(item_no);
        boolean keepgoing = true;
        String product_id;
        // now we must loop for each item_no
        while(keepgoing)
        {
            String ps2sql = "select product_id," + " db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_s + "]/item_price')," + " db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_s + "]/ship_date')," + " db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_s + "]/item_status')," + " db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_s + "]/order_quantity')," + " supplier_part_no,product_name,description"
                + " from orderxml,products" + " where ((order_no = '" + order_no + ")
                + " and (db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_s + "]/product_id')=product_id'))";
            resultSet = s2.executeQuery(ps2sql);
            if (rs2.next())
            {ship_date = rs2.getString(3); if (ship_date == null) {ship_date = "Not shipped";}}
            orderdoc.append("<ORDER_ITEM ITEM_NO=" + item_no_s + " STATUS=" + rs2.getString(4) + " SHIP_DATE=" + ship_date + "/">;"
        }
    }
```

Integrating XML with DB2 XML Extender and DB2 Text Extender
This method returns an XML document with the tag `<ORDERDETAIL>` as root element.

We first extract the single-occurring elements or attributes we need from the order XML document. This is executed without any difficulty as we did in the `generateOrderList()` method. Based on this information, we generate the first part of the new XML document to be returned.

The second part of the XML document to generate contains the information about the various products ordered. We progress item by item in extracting the information from the order XML document stored in the database. We also join the information extracted from the ORDERXML to the PRODUCTS table so that we can, for example, include the description, in the newly generated XML document although it was not stored in the original order XML document.

This concludes the browsing part of the order tracking component. In the next section, we describe the second major part of this component: the updating part performed with the `CustomerStorageServlet` servlet.

### 6.5.2 The servlet CustomerStorageServlet

This servlet, though being a servlet like any servlet, has a major difference compared to the servlet we presented so far: this servlet is not supposed to be called in a B2C approach (that is from a browser) but in a B2B approach (that is by a software component from another server, or in our case from the same server).

This `CustomerStorageServlet` has three main roles:
1. It receives an XML document sent by a servlet from the supplier side of the application with information about the shipped products
2. It parses this XML document to extract the information about the order number and the shipped products
3. It updates the order XML documents stored in the database (in the ORDERXML table) with the information parsed from the received XML document

The code of this servlet is presented here and is commented upon in the following notes.

```java
public void performTask(HttpServletRequest request, HttpServletResponse response) throws 
javax.servlet.ServletException, java.io.IOException {
    try{
        java.sql.Date sqlDate, lastSqlDate;
        Connection conn = null;
        ResultSet rs;
        Vector order_nos = new Vector(); // to keep all the different order_no
        String[] att;
        int i,j;
        boolean alreadyExist, keepGoing, allShipped;
        String status, sql, order_no, product_id, ship_date, last_ship_date, item_no_S;
        int item_no;
        String orderxml = request.getParameter("xmldata").trim(); // use of SAX to extract the order number
        SAXParser p = new SAXParser();
        SAXHandler h = new SAXHandler();
        p.setDocumentHandler(h);
        StringReader srxml = new StringReader(orderxml);
        InputSource isxml = new InputSource(srxml);
        p.parse(isxml);
        Vector shippedProducts = h.getShippedProducts(); // update the XML documents for each shipped product
        try
        {
            conn = ds.getConnection(id,pw);
            Statement s = conn.createStatement();
            for (i=0;i<shippedProducts.size();i++)
            {
                att = (String[]) shippedProducts.elementAt(i);
                order_no = att[0];
                alreadyExist = false;
                for (j=0;j<order_nos.size();j++)
                {
                    if (order_no.equals(order_nos.elementAt(j))) {alreadyExist=true;}
                }
                if (!alreadyExist) order_nos.addElement(order_no);
                product_id = att[1];
                ship_date = att[2];
                sql = "update orderxml set orderdoc = db2xml.Update(orderdoc, "/order/products/product[@product_id=" + product_id + "]/item_status', 'SHIPPED') 
where order_no =" + order_no + ";";
                s.executeUpdate(sql);
                sql = "update orderxml set orderdoc = db2xml.Update(orderdoc, "/order/products/product[@product_id=" + product_id + "]/ship_date', " + ship_date + ");";
                s.executeUpdate(sql);
            }
            for (i=0;i<order_nos.size();i++)
            {
                // update the status of the order if all products are shipped
                order_no = (String) order_nos.elementAt(i); 
                keepGoing = true;
                allShipped = true;
            }
        }
```

Integrating XML with DB2 XML Extender and DB2 Text Extender
last_ship_date = "";
item_no = 1;
item_no_S = Integer.toString(item_no);
while(keepGoing)
{
  sql = "select db2xml.extractvarchar(orderdoc,  
  '/order/products/product[@item_no="" + item_no_S + ""]/item_status'),
  db2xml.extractvarchar(orderdoc, '/order/products/product[@item_no="" + item_no_S + 
  "]/ship_date') from orderxml where (order_no = "" + order_no + "]"};
  rs = s.executeQuery(sql);
  if (rs.next())
    {status = rs.getString(1);
      if (status != null)
        {if (!(rs.getString(1).trim().equals("SHIPPED")))
           {allShipped=false; keepGoing = false;}
        else {ship_date = rs.getString(2);
               if (item_no==1) {last_ship_date = new String(ship_date);}
               else {if (ship_date.compareTo(last_ship_date)>0)
                  {last_ship_date = new String(ship_date);}}
        }
        }
    }
    else
    {keepGoing = false;}
    else
    {keepGoing = false;}
  item_no_S=Integer.toString(++item_no);
}
if (allShipped)
{
  sql = "update orderxml set orderdoc = db2xml.Update(orderdoc,  
  '/order/status', 'SHIPPED') where (order_no ="" + order_no + "]"};
  s.executeUpdate(sql);
  sql = "update orderxml set orderdoc = db2xml.Update(orderdoc,  
  '/order/ship_date', " + last_ship_date + "]" where (order_no ="" + order_no + "]"};
  s.executeUpdate(sql);
}
}
catch (Exception e)
{
  e.printStackTrace();
}
finally
{
  try {// return the connection to the connection pool
    if (conn != null) conn.close();
    } catch (java.sql.SQLException e) {} }
}
catch(Exception e2){
  e2.printStackTrace();
}

Notes:
1. The first step is to receive the XML document.
   This is realized by requesting the parameter xmlData as we would do with any parameter.
2. The second step is to parse the information from this XML document
2. We use a SAX parser rather than a DOM parser as it is clearly adapted to our purpose of extracting some information out of the XML data.

3. The SAXHandler class is a helper class that we created for the purpose of parsing all our XML documents and not only the update info sent back the supplier side. This class extends the org.xml.class.HandlerBase class by implementing only the methods we are interested in such as characters() or processingInstruction().

4. We register our helper class to the parser as a DocumentHandler so that the parser will notify it of the events it generates when parsing XML data.

5. We parse the received XML document.

6. Thanks to the events generated by the parser, our helper class has collected the information included in the XML document about the shipping of the products. This info is accessible through its getShippedProducts() method.

3. The last step is to update our orders XML documents with the information received about the shipped products. The information sent by the suppliers contains all products shipped since the last update with their associated order number and ship date. The number of updates to execute can be potentially high and certainly concerns more than just one order XML document.

7. We keep track of the order number of each order that we updated. This is necessary because once all the updates are executed, each updated ordered must be scanned to check if all the products of this order are shipped and if yes, the general status of the order must be updated to SHIPPED too.

8. These are the SQL statements used to update the status of each shipped product and the date of its shipping. This SQL statement is very powerful because DB2 XML Extender with its XPath support allows us in the same SQL statement, first to locate the product in the XML document and then to update its status.

10. For each order updated with a shipped product, we must check the status of all products and update the status of the document when they are all shipped.
6.6 The supplier side

On the other side of the XMLapp application (which conceptually is located on another machine, but practically is hosted by the same server), the suppliers can browse for their orders in a B2C interaction, indicate which products have just shipped, and send the information back to the customer through a B2B interaction.

6.6.1 The user interfaces

The first screen the suppliers receive when working with their XMLapp applications is an overview of all their existing orders, the status of each order, and to which supplier it applies. This first screen is shown in Figure 76.

![XMLapp Suppliers side](Image)

**ORDERS UNDER PROCESS**

Please click on the link to view the details of a specific order and modify it.

<table>
<thead>
<tr>
<th>Orders No.</th>
<th>Order status</th>
<th>Supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000082215922506859000000</td>
<td>SHIPPED</td>
<td>SUPP01</td>
</tr>
<tr>
<td>2000082218394889270600000</td>
<td>SHIPPED</td>
<td>SUPP01</td>
</tr>
<tr>
<td>2000082218822988036000000</td>
<td>UNSUBMITTED</td>
<td>SUPP01</td>
</tr>
<tr>
<td>2000082221592593180600000</td>
<td>SHIPPED</td>
<td>SUPP01</td>
</tr>
<tr>
<td>2000082221592593180600000</td>
<td>SHIPPED</td>
<td>SUPP03</td>
</tr>
<tr>
<td>2000082221592593180600000</td>
<td>SHIPPED</td>
<td>SUPP02</td>
</tr>
<tr>
<td>2000082223335280101000000</td>
<td>UNSUBMITTED</td>
<td>SUPP01</td>
</tr>
<tr>
<td>200008266000704534477000000</td>
<td>UNSUBMITTED</td>
<td>SUPP03</td>
</tr>
</tbody>
</table>

Click [here](#) to see the information to be sent back to the customers regarding the shipping of their orders.

*Figure 76. Supplier side: orders overview*
Suppliers can then, for example, choose any order they have not shipped yet by clicking on the hyperlink of their choice, and can browse the details to see which product they have in their warehouse and are able to ship. Once the shipment is realized, they update the current status of the product on the same screen, as shown in Figure 77.

![Figure 77. Supplier side: browsing order details and shipping the products](image)

Finally, after the shipment of some products, the suppliers can send this information back to the customer side so that the original orders are updated, and the customers are able to follow the evolution in the shipment of their ordered products. This step could have been automated and made transparent to the suppliers, but since this application has also a demonstration purpose, we must manually initiate the update interaction in order to better realize the undergoing process with exchange of XML data.
Figure 78 displays this interaction. This screen is accessed by clicking on the Here hyperlink that was shown in Figure 76 on page 201.

![XMLapp Suppliers side](image)

### Suppliers side

**Order information to be sent back to customers**

Please click on the PROCESS button to create an XML document and send it.

<table>
<thead>
<tr>
<th>Shipping date</th>
<th>Order number</th>
<th>Product id</th>
<th>Supplier part number</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000-08-29</td>
<td>20000829182833148019000000</td>
<td>IT-003</td>
<td>SCN455</td>
</tr>
<tr>
<td>2000-08-29</td>
<td>20000829182833148019000000</td>
<td>IT-004</td>
<td>SCN776</td>
</tr>
<tr>
<td>2000-08-29</td>
<td>20000829182833148019000000</td>
<td>IT-002</td>
<td>PRNT1234</td>
</tr>
</tbody>
</table>

Click [here](#) to see the other orders.

*Figure 78. Supplier side: updating the customer side with shipped products*

#### 6.6.2 The JSP files: SupMainMenu and SupB2B

The screens shown in Figure 76 on page 201 and Figure 77 on page 202 are both the output of JSPs, respectively — Sup MainMenu and SupB2B.

These JSPs access the database and read the table with the following tags: 
The following code excerpt taken from SupMainMen.jsp illustrates how to use these tags to output an HTML table with data extracted from an XML document stored in our database:

```xml
<tsx:dbconnect id="SUP001" userid="xmlguru" passwd="xmlguru" url="jdbc:db2:L301" driver="COM.ibm.db2.jdbc.app.DB2Driver">
</tsx:dbconnect>
<tsx:dbquery id="qry" connection="SUP001">
  select order_no, db2xml.extractVarchar(orderxml, '/order/status') as status, db2xml.extractVarchar(orderxml, '/order/products/product[@item_no="1"]/supplier_id') as supplier from supplier_storage
</tsx:dbquery>

....

<TABLE BORDER="2" ALIGN="CENTER">
<THEAD><th>Orders No.</th><th>Order status</th><th>Supplier</th></THEAD>
<TBODY>
<tsx:repeat>
<tr>
<td><a href="../servlet/BrowseSupplierServlet?ORDER_NO=<tsx:getProperty name="qry" property="ORDER_NO" />&SUPPLIER_ID=<tsx:getProperty name="qry" property="SUPPLIER" />"><tsx:getProperty name="qry" property="ORDER_NO" /></a></td>
<td><tsx:getProperty name="qry" property="STATUS" /></td>
<td><tsx:getProperty name="qry" property="SUPPLIER" /></td>
</tr>
</tsx:repeat>
</TBODY>
</TABLE>

It is important to notice that using the <tsx:dbconnect> tag with the attributes url and driver causes the servlet generated from the JSP to open a new connection to the database without making use of the connection pools maintained by WebSphere Application Server. With Version 3.5 of WebSphere Application Server (and VisualAge for Java), a new attribute jndiname is supported, where you can specify the JNDI name of the connection pool to use.
6.6.3 The servlets used on the supplier side

This side of the application uses 4 servlets:

- **BrowseSupplierServlet**: This is the servlet called when a supplier wants to view the details of one of his orders. The supplier invokes it by clicking on one of the hyperlinks to the orders shown in Figure 77 on page 202.

- **UpdateServlet**: This is the servlet called when the supplier wants to update the current status of the products in his orders by clicking on the **MODIFY** button Figure 77 on page 202.

- **UpdateB2BServlet**: This is the servlet used to create the XML document containing the data about the shipped products and send it to the customer side. It is invoked by the supplier when clicking on the **PROCESS** button shown in Figure 78 on page 203.

- **SupplierStorageServlet**: This is the servlet equivalent to **CustomerStorageServlet**. It makes up the receptive part of the B2B interchange so that the XML document sent to each supplier from the customer side are received when an order has been submitted. It is never called directly by a user.

6.6.3.1 The servlet **BrowseSupplierServlet**

This servlet is similar to the **BrowseOrderServlet** introduced in 6.5.1, “The servlet **BrowseOrderServlet**” on page 192. Still, there is a big difference: in the **BrowseOrderServlet**, we extracted the needed information from the order XML document, used it to build a new XML document adapted to a given stylesheet and finally transformed this XML document into HTML while in the **BrowseSupplierServlet**, we simply read the order XML document stored in the database and transform it with an appropriate XSL stylesheet.

If you compare the code of both servlets, you can see a striking contrast in the amount of processing needed. Nevertheless, the **BrowseOrderServlet** was very useful to demonstrate how to extract all the required data from an XML document stored as an XML Column.

6.6.3.2 The servlet **UpdateServlet**

This servlet has a double role:

- It inserts in the **SUPPLIER_SHIPPING** table the information about the products shipped by the supplier. This table is used as a temporary container to know what information has still to be sent to the customer side. This information is deleted once the XML document with the products shipped by the suppliers has been successfully transmitted to the customer side.
It updates the orders XML documents of each suppliers that has shipped a new product. After these updates it must also review each XML document whose at least one product has been updated, so that the general status of the XML document is updated to SHIPPED if all products are now shipped. This last processing is similar to the one described for the CustomerStorageServlet on Page 197.

### 6.6.3.3 The servlet UpdateB2BServlet
This servlet reads the relational data put aside in the SUPPLIER_SHIPPING table by the UpdateServlet and based on that information it builds the XML document to send to the customer side. It then sends the XML data in a way similar to the how the Basket bean submits the order information to the supplier side and already described in 6.4.5.5, “The postToSupplier() method” on page 189.

### 6.6.3.4 The servlet SupplierStorageServlet
Our last servlet is the counterpart of the CustomerStorageServlet. This servlet receives the XML document sent to each supplier by the customer side and containing the information about the products ordered by one customer. Once the XML data is received, it is parsed by a SAXParser to extract the order number with the help of our SAXHandler class previously introduced in 6.5.2, “The servlet CustomerStorageServlet” on page 197.

This concludes our presentation of the XMLapp application implemented with XML Columns. You can now try it out for yourself and skip to Chapter 8, “XMLapp: installing XML Columns version in WAS” on page 249 to run this application on WebSphere Application Server, or if you prefer, you can also try to run this same application within the IBM WebSphere Test Environment of VisualAge for Java. A few changes are needed, due to some limitations of VisualAge compared to WebSphere. Look in Appendix A, “XMLapp: installing XML Columns version in VAJ” on page 269 for more information about the procedure to follow.
Chapter 7. XMLapp: implementation with XML Collections

In this chapter, we describe how to implement the sample application scenario using the XML Collections feature of DB2 XML Extender. The application runs on WebSphere Application Server and is implemented using servlets.

7.1 Application overview

Our application is based on the scenario discussed in Chapter 5, “XMLapp application: scenario” on page 155. We implement the same sample application, but here we use XML Collections instead of XML Columns. So the scenario remains the same, but the database architecture and the code used to implement it are totally different.

7.2 Database architecture

We store the details of the order and the supplier information in a DB2 database. To keep it simple, the order and supplier details are kept in the same database, which is not the case in a real-world application.

The following tables are stored in the database:

- USERS
- ORDER_HEADERS
- ORDER_ITEMS, PRODUCTS
- SUPPLIER_ORDERS
- SUPP_ORDER_DETAILS
- RESULT_TAB

Note that RESULT_TAB is not directly related to the application. We use it to temporarily store the XML documents generated.

The database architecture shown in Figure 79 for the sample application has these characteristics:

- The USERS table contains the information related to the users, such as username, password, and so on.
- The ORDER_HEADERS table contains data about each order.
- The ORDER_ITEMS table stores the item level details of each order.
The PRODUCTS table contains the list of products that can be ordered along with a description, price details and the corresponding supplier for each product.

The SUPPLIER_ORDERS table contains a record for each order that has been sent to a supplier.

SUPP_ORDER_DETAILS stores the item level details of each order. In a real-world application the order and supplier information would be stored in a separate database but since it is a sample application we store this in a single database.

---

**USER**

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>PASSWORD</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>ADDRESS</th>
<th>CITY</th>
<th>STATEORPROVINCE</th>
<th>ZIPCODE</th>
<th>PHONE</th>
<th>EMAIL</th>
</tr>
</thead>
</table>

**ORDER_HEADERS**

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>CUSTOMER_ORDER_NO</th>
<th>ORDER_DATE</th>
<th>SHIP_DATE</th>
<th>USERNAME</th>
</tr>
</thead>
</table>

**ORDER_ITEMS**

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>PRODUCT_ID</th>
<th>ITEM_PRICE</th>
<th>ITEM_STATUS</th>
<th>ITEM_NO</th>
<th>ORDER_QUANTITY</th>
<th>SHIP_DATE</th>
</tr>
</thead>
</table>

**PRODUCTS**

<table>
<thead>
<tr>
<th>PRODUCT_ID</th>
<th>PRODUCT_NAME</th>
<th>DESCRIPTION</th>
<th>SUPPLIER_ID</th>
<th>SUPPLIER_PART_NO</th>
<th>UNITS_PRICE</th>
<th>UNITS_IN_STOCK</th>
<th>MINIMUM_STOCK_LEVEL</th>
</tr>
</thead>
</table>

**SUPPLIER_ORDERS**

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>CUSTOMER_ORDER_NO</th>
<th>ORDER_DATE</th>
</tr>
</thead>
</table>

**SUPPLIER_ORDER_DETAILS**

<table>
<thead>
<tr>
<th>ORDER_NO</th>
<th>PRODUCT_ID</th>
<th>STATUS</th>
<th>QUANTITY</th>
<th>SHIP_DATE</th>
</tr>
</thead>
</table>

*Figure 79. Database architecture*
7.3 XML application implementation

We will discuss the implementation of the application in the sections to follow. The instructions for installing the application are given in Chapter 9, “XMLapp: installing XML Collections version in WAS” on page 263. In order to explain the application, we will be listing important sections of the code. For complete details, you can go through the source code.

7.3.1 Logon and user profile management

In this section we discuss logon and user profile management.

7.3.1.1 Main menu servlet

The MainMenuServlet displays the main menu by applying a stylesheet LogonForm.xsl to a static XML file called MainMenu.xml and transforming it to HTML. The following code listing shows how this is done. An instance of class ProcessXML is created. The URL for the XML document to be transformed, the URL for the stylesheet to be applied, and the PrintWriter object to which the output has to be sent is passed to the process method:

```java
PrintWriter out = res.getWriter();
res.setContentType("text/html");
String ssMainMenuURL = "http://localhost/xml/MainMenu.xml";
String ssXSLURL = "http://localhost/xsl/LogonForm.xsl";
try {
    ProcessXML proc = new ProcessXML();
    proc.process(ssMainMenuURL, ssXSLURL, out);
    System.out.println("Sucessfull");
} catch (Exception e) {
    e.printStackTrace(out);
}
```

The code listing for the process method of the ProcessXML class follows. In this method an object of class XSLProcessor is instantiated, and the arguments to the method process of ProcessXML are passed to the process method of XSLProcessor class which does the actual transformation and sends back the result. XSLProcessor is a wrapper class for the Xalan XSLT processor available in the com.lotus.xsl package.

```java
XSLProcessor xlsProc = new XSLProcessor();
xlsProc.process(ssXMLURL, ssXSLURL, out);
catch (Exception e) {
    e.printStackTrace(out);
}
The main menu can be invoked by typing in the following URL:

http://localhost/XMLExtApp/servlet/MainMenuServlet

The screen you would get on invoking this URL is shown in Figure 80.

![Main menu](image)

Figure 80. Main menu
7.3.1.2 Register form servlet
The RegisterFormServlet is used to display the registration form for new users to register themselves. This servlet transforms the static page RegisterForm.xml using the stylesheet LogonForm.xsl to HTML. It uses the ProcessXML class explained earlier to perform the transformation. The registration screen is shown in Figure 81.

![Registration form](image)

Figure 81. Registration form
7.3.1.3 Register servlet

The RegisterServlet is used to register a new user. The code given below shows how this is done. If the registration is successful, it creates a new session and binds the username to the session and returns a screen stating that the registration was successful. Otherwise, it returns an error screen:

... 
username = (String) req.getParameter("id");
passwd = (String) req.getParameter("pw");
if (username == null || passwd == null || username.length() < 1 ||
passwd.length() < 1)
page = "http://localhost/xml/RegisterInvalid.xml";
else {
String sql = "Insert into xmlguru.users values (?,?,?,?,?,?,?,?,?,?)";
String firstName = (String) req.getParameter("firstname");
String lastName = (String) req.getParameter("lastname");
String address = (String) req.getParameter("address");
String city = (String) req.getParameter("city");
String state = (String) req.getParameter("state");
String zipcode = (String) req.getParameter("zipcode");
String phone = (String) req.getParameter("phone");
String email = (String) req.getParameter("email");
try {
con.setAutoCommit(true);
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1,username);
stmt.setString(2,passwd);
stmt.setString(3,firstName);
stmt.setString(4,lastName);
stmt.setString(5,address);
stmt.setString(6,city);
stmt.setString(7,state);
stmt.setString(8,zipcode);
stmt.setString(9,phone);
stmt.setString(10,email);
stmt.executeUpdate();
HttpSession session = req.getSession(true);
session.putValue("username",username);
page = "http://localhost/xml/RegisterOk.xml";
}catch(SQLException se) {
page = "http://localhost/xml/RegisterError.xml";
//se.printStackTrace(out);
}
7.3.1.4 Logon page servlet

This LogonPageServlet transforms the static XML document LogonForm.xml using the stylesheet LogonForm.xsl to present the user with a logon screen. It also uses the ProcessXML class explained earlier to do the transformation. The screen you would get on invoking this servlet is shown in Figure 82.

![Figure 82. Logon form](image)

7.3.1.5 Logon servlet

The LogonServlet checks whether the userid and password entered by the user are valid. This servlet is invoked when the user enters his username and password in the logon screen shown in Figure 82 and clicks the Logon button. If the servlet is able to validate the user, the Logon Successful screen is displayed as shown in Figure 83; otherwise, an error screen is displayed. If the user is validated, then a new session is created, and the username is bound to the session. The code listing for these activities follows:
String sql = "Select passwd from xmlguru.users where username=?";
con.setAutoCommit(true);
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, username);
ResultSet rs = stmt.executeQuery();
rs.next();
passwd = rs.getString(1);
if (pw.equals(passwd)) {
    page = "http://localhost/xml/LogonOk.xml";
    HttpSession session = req.getSession(true);
    session.putValue("username", username);
} else
    page = "http://localhost/xml/LogonError.xml";

Figure 83. Logon Successful

We are pleased to do business with You!

******************************************************************************

Please follow the links below to continue:

*** Search our product catalog ***
*** Browse the orders in process ***

******************************************************************************

Figure 83. Logon Successful
7.3.1.6 User query servlet

The UserQueryServlet will dynamically generate an XML document containing the list of users and their details, apply a stylesheet user.xsl to transform it to HTML and send it back. The XML document is composed by DB2 XML extender using the dxxGenXML stored procedure of DB2 XML extender. The code listing follows:

```java
String procName = "db2xml.dxxGenXML"; // Name of stored procedure used to generate XML document.
String sql = "Call " + procName + "(" + procName + ")";
CallableStatement cs = con.prepareCall(sql);
String buffer = new String();
StringBuffer dad = new StringBuffer();
//Read the DAD file into a StringBuffer.
BufferedReader in = new BufferedReader(new FileReader(DADFile));
while ((buffer = in.readLine()) != null)
dad.append(buffer);
...
//Register the input & output parameters with the stored procedure.
   cs.registerOutParameter(6, Types.INTEGER);
   cs.setInt(6, actualNumOfDocs);
   cs.registerOutParameter(7, Types.INTEGER);
   cs.setInt(7, errCode);
   cs.registerOutParameter(8, Types.VARCHAR);
   cs.setString(8, msgText);
   cs.setString(1, dad.toString());
   cs.setString(2, resultTable);
   int overrideType = 0; // overrideType: NO_OVERRIDE
   cs.setInt(3, overrideType);
   cs.setString(4, null);
   ...
   cs.setInt(5, maxNumOfDocs);
   ...
//Set the transaction isolation to repeatable read.
   con.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
   Statement stmt = con.createStatement();

   //Delete any rows in the result table so that the document we generate is the only one in the table.
   stmt.executeUpdate("Delete From " + resultTable);
   cs.execute();
   errCode = cs.getInt(7);
   msgText = cs.getString(8);
   ...
//The generated XML document is exported to a file.
```

Chapter 7. XMLapp: implementation with XML Collections  215
sql = "select db2xml.content(db2xml.XMLClob(doc)," + tempFileName + ") from result_tab ";
stmt = con.createStatement();
stmt.executeQuery(sql);

The DAD file used by the stored procedure in the code listing follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>
Select 'const' as const,username,passwd,firstname,lastname from xmlguru.users order by const,username</SQL_stmt>
</Xcollection>
</DAD>
```
The servlet displays the screen as shown in Figure 84.

Figure 84. List of all the users
Here is a listing of the dynamically generated XML file that was used to produce the screen in Figure 84 on page 217:

```xml
<?xml version="1.0"?>
<!DOCTYPE person SYSTEM "C:\DTDFiles\user.dtd">
<USERLIST>
  <USER>
    <USERNAME>a</USERNAME>
    <PASSWORD>a</PASSWORD>
    <FIRSTNAME>a</FIRSTNAME>
    <LASTNAME>a</LASTNAME>
  </USER>
  <USER>
    <USERNAME>b</USERNAME>
    <PASSWORD>b</PASSWORD>
    <FIRSTNAME>vxc</FIRSTNAME>
    <LASTNAME>bvcc</LASTNAME>
  </USER>
  <USER>
    <USERNAME>cd</USERNAME>
    <PASSWORD>cd</PASSWORD>
    <FIRSTNAME>cd</FIRSTNAME>
    <LASTNAME>cd</LASTNAME>
  </USER>
  <USER>
    <USERNAME>s</USERNAME>
    <PASSWORD>s</PASSWORD>
    <FIRSTNAME>sun</FIRSTNAME>
    <LASTNAME>km</LASTNAME>
  </USER>
  <USER>
    <USERNAME>t</USERNAME>
    <PASSWORD>t</PASSWORD>
    <FIRSTNAME>test</FIRSTNAME>
    <LASTNAME>t</LASTNAME>
  </USER>
  <USER>
    <USERNAME>tango</USERNAME>
    <PASSWORD>tango</PASSWORD>
    <FIRSTNAME>tango</FIRSTNAME>
    <LASTNAME>tango</LASTNAME>
  </USER>
  <USER>
    <USERNAME>u</USERNAME>
    <PASSWORD>u</PASSWORD>
    <FIRSTNAME>u</FIRSTNAME>
    <LASTNAME>u</LASTNAME>
  </USER>
</USERLIST>
```
Chapter 7. XMLapp: implementation with XML Collections

The stylesheet used to transform the XML document follows:

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
        <META http-equiv="Expires" content="0"/>
        <STYLE TYPE="text/css">
          BODY{font-family : Verdana,sans-serif;}
          .formText{font-family : Verdana,sans-serif;}
          TD{font-family : Verdana,sans-serif; font-weight: bold;}
          .button{ font-family:Verdana,sans-serif;font-size:100%;}
          .tableText{
            font-weight : bold;
            font-family : monospace;
            text-align : right;}
        </STYLE>
      </HEAD>
      <TITLE>Current order</TITLE>
      <BODY background="/xsl/world.jpg">
        <CENTER>
          <DIV>
            <IMG SRC="/xsl/xmlapp.gif" ALT="XMLapp" ALIGN="Middle" BORDER="0" HSPACE="0" VSPACE="0"/>
            <H3>List of all the users</H3>
          </DIV>
          <HR/>
          <TABLE BORDER="1" class="tableText" WIDTH="80%">
            <THEAD>
              <TD>USER NAME</TD>
              <TD>PASSWORD</TD>
              <TD>FIRST NAME</TD>
              <TD>LAST NAME</TD>
            </THEAD>
            <TBODY>
```

```xml```
The HTML that was generated on applying the XSL transformation to the XML document follows:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
<html>
<head>
<meta content="text/html; charset=iso-8859-1" http-equiv="Content-Type">
<meta content="0" http-equiv="Expires">
<style type="text/css">
body{font-family : Verdana,sans-serif;}
.formText{font-family : Verdana,sans-serif;}
td{font-family : Verdana,sans-serif; font-weight: bold;}
.button{ font-family:Verdana,sans-serif;font-size:100%;}
tableText{
  font-weight : bold;
  font-family : monospace;
  text-align : right;}
</style>
<title>Current order</title>
</head>
<body background="/xsl/world.jpg">
<center>
<table>
<thead>
<tr>
<td><xsl:value-of select="USERNAME"/></td>
<td><xsl:value-of select="PASSWORD"/></td>
<td><xsl:value-of select="FIRSTNAME"/></td>
<td><xsl:value-of select="LASTNAME"/></td>
</tr>
</thead>
<tbody>
</tbody>
</table>
</center>
</body>
</html>
```

The HTML that was generated on applying the XSL transformation to the XML document follows:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
<html>
<head>
<meta content="text/html; charset=iso-8859-1" http-equiv="Content-Type">
<meta content="0" http-equiv="Expires">
<style type="text/css">
body{font-family : Verdana,sans-serif;}
.formText{font-family : Verdana,sans-serif;}
td{font-family : Verdana,sans-serif; font-weight: bold;}
.button{ font-family:Verdana,sans-serif;font-size:100%;}
tableText{
  font-weight : bold;
  font-family : monospace;
  text-align : right;}
</style>
<title>Current order</title>
</head>
<body background="/xsl/world.jpg">
<center>
<table>
<thead>
<tr>
<td><xsl:value-of select="USERNAME"/></td>
<td><xsl:value-of select="PASSWORD"/></td>
<td><xsl:value-of select="FIRSTNAME"/></td>
<td><xsl:value-of select="LASTNAME"/></td>
</tr>
</thead>
<tbody>
</tbody>
</table>
</center>
</body>
</html>
```
7.3.2 Search engine

In this section we discuss the search engine for our application.

7.3.2.1 Search servlet

The SearchServlet is used to search for products. When the user gives an input string to search the servlet will produce a list of products that matches the input string. The user can then add these products to order. The input string is matched against the PRODUCT_ID, PRODUCT_NAME and DESCRIPTION fields of the table PRODUCTS. The servlet uses the dxxGenXML stored procedure to compose the XML document containing the result of the search, an XSL stylesheet product.xsl is applied to the XML document to transform it to HTML which is then displayed on the screen. The code listing for this servlet follows:

...String procName = "db2xml.dxxGenXML"; // Name of stored procedure used to generate XML document. String sql = "Call " + procName + "(? ? ? ? ? ? ? ?) ";
CallableStatement cs = con.prepareCall(sql);
writeLog("The name of DAD file is :" + DADFile, 0);
String buffer = new String();
StringBuffer dad = new StringBuffer();
//Read the DAD file into a StringBuffer.
in = new BufferedReader(new FileReader(DADFile));
while ((buffer = in.readLine()) != null)
dad.append(buffer);
//Set the override SQL
if (search == null || search.length() == 0)
override = "Select 'const' as const,p.product_id,product_name,description,unit_price from xmlguru.products p order by const,product_id\0";
else {
search = search.trim();
search = search.toUpperCase();
override = " Select 'const' as const,p.product_id, product_name, description, unit_price from xmlguru.products p where UCASE(product_id) LIKE '%" + search +"%' OR UCASE(PRODUCT_NAME) LIKE '%" + search +"%' OR UCASE(DESCRIPTION) LIKE '%" + search +"%' order by const,product_id\0";
// override value
}
...
//Register the input & output parameters to the stored procedure.
        cs.registerOutParameter (6, Types.INTEGER);
        cs.setInt(6, actualNumOfDocs);
        cs.registerOutParameter (7, Types.INTEGER);
        cs.setInt(7, errCode);
        cs.registerOutParameter (8, Types.VARCHAR);
        cs.setString(8, msgText);
        cs.setString(1, dad.toString());
        cs.setString(2, resultTable);
        int overrideType = 1; // overrideType: SQL_OVERRIDE
        cs.setInt(3, overrideType);
        cs.setString(4,override);
        int maxNumOfDocs = 100; // just set to 100
        cs.setInt(5, maxNumOfDocs);
        ...

        //Set the transaction isolation to repeatable read.
        con.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
        Statement stmt = con.createStatement();
        //Delete any rows in the result table so that the document we generate is
        //the only one in the table.
        stmt.executeUpdate(" Delete From "+resultTable);
        cs.execute();
        ...

222  Integrating XML with DB2 XML Extender and DB2 Text Extender
//Export the generated XML document to a file.
sql = "select db2xml.content(db2xml.XMLClob(doc),'"+tempFileName+'') from
result_tab ";
stmt = con.createStatement();
stmt.executeQuery(sql);
...

//Transform the XML document to HTML.
ProcessXML proc = new ProcessXML();
proc.process(tempFileName,ssProductURL,out);
...

A listing of the DAD file used by the dxxGenXML stored procedure follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>
Select 'const' as const,p.product_id,product_name,description,unit_price
from xmlguru.products p order by const,product_id</SQL_stmt>
<prolog><?xml version="1.0"?></prolog>
<doctype>!DOCTYPE PRODUCT_LIST SYSTEM "C:\DTFiles\product.dtd"</doctype>
<root_node>
<element_node name="PRODUCT_LIST">
  <element_node name="PRODUCT" multi_occurrence="YES">
    <attribute_node name="PRODUCT_ID">
      <column name="product_id"/>
    </attribute_node>
    <element_node name="PRODUCT_NAME">
      <text_node>
        <column name="product_name"/>
      </text_node>
    </element_node>
    <element_node name="DESCRIPTION">
      <text_node>
        <column name="description"/>
      </text_node>
    </element_node>
    <element_node name="PRICE">
      <text_node>
        <column name="unit_price"/>
      </text_node>
    </element_node>
  </element_node>
</element_node>
</root_node>
```
The servlet displays the screen shown in Figure 85 as result of the search.

![Figure 85. Search results](image)

7.3.3 Implementation of shopping basket and ordering system

When the user clicks the **Add to order** button on the search screen, as shown in Figure 85 on page 224, the system needs to add the selected items along with the quantity into a shopping basket. The shopping basket and ordering system is implemented by the OrderBasketServlet. This servlet uses a DOM tree to implement the shopping basket. This DOM object lives within the HttpSession for the user so that it can remain between different pages. The items ordered and their quantities are passed as parameters which look like 

\[
\_\text{PNAME}\_\text{PROID}=\text{QUANT}, \text{ where PROID is the product ID of the item ordered, and QUANT is the quantity ordered.}
\]
7.3.3.1 Program flow
The program flow for shopping and ordering is given below.

1. The user invokes the OrderBasketServlet for the first time by doing a search and adding a set of items to the order. Within the doGet method of OrderBasketServlet, the buildTree method is called. Since this is a new order the buildTree will create a new DOM tree consisting of a unique order number, order status set to UNSUBMITTED, order date set to current date and username set to the name of current user. A listing of this part of the code follows:

```java
if (Doc == null) {
    sql = "Select max(order_no) from xmlguru.order_headers";
    stmt = con.createStatement();
    rs = stmt.executeQuery(sql);
    rs.next();
    num = rs.getString(1);
    if (num == null)
        num = "20000807202456408695000000";
    rs.close();
    BigInteger ordernum = new BigInteger(num);
    BigInteger constant = new BigInteger("1");
    ordernum = ordernum.add(constant);
    orderno = ordernum.toString();//Generate unique order number
    Doc = (Document)
        Class.forName("com.ibm.xml.dom.DocumentImpl").newInstance();
    root = Doc.createElement("ORDER");
    root.setAttribute("ORDER_NO", orderno);
    root.setAttribute("ORDER_STATUS", "UNSUBMITTED");
    Doc.appendChild(root);
    Element orderdate = Doc.createElement("ORDER_DATE");
    orderdate.appendChild(Doc.createTextNode(new java.sql.Date(System.currentTimeMillis()).toString()));
    root.appendChild(orderdate);
    Element name = Doc.createElement("USERNAME");
    name.appendChild(Doc.createTextNode(username));
    root.appendChild(name);
}
```

Once this is done, the ordered items are added to the DOM tree. The code for this follows: It should be kept in mind that there is an option to search and add more items to the order. This again is taken care of by the following section of code.

```java
int i = 0;
en = req.getParameterNames();
while (en.hasMoreElements()) {
    
```
paramName = (String) en.nextElement();
if (paramName.startsWith("_PNAME_")) {
    String prodid = paramName.substring(7);
    String quantity = req.getParameterValues(paramName)[0];
    if (quantity != null && !quantity.equals("")) {
        try {
            int q = Integer.parseInt(quantity);
            catch (NumberFormatException nfe) {
                writeLog("Number format exception while parsing quantity", 1);
                nfe.printStackTrace(out);
                System.exit(-1);
            }
            i++;
            sql = "Select product_name,description,unit_price from xmlguru.products
                where product_id='" + prodid + "';"
            stmt = con.createStatement();
            rs = stmt.executeQuery(sql);
            if (rs.next()) {
                int price = rs.getInt(3);
                product = Doc.createElement("PRODUCT");
                if (root == null)
                    root = Doc.getDocumentElement();
                root.appendChild(product);
                Element productId = Doc.createElement("PRODUCT_ID"殿下);
                productId.appendChild(Doc.createTextNode(prodid.toString()));
                product.appendChild(productId);
                Element productName = Doc.createElement("PRODUCT_NAME"殿下);
                productName.appendChild(Doc.createTextNode(rs.getString(1)));
                product.appendChild(productName);
                Element desc = Doc.createElement("DESCRIPTION"殿下);
                desc.appendChild(Doc.createTextNode(rs.getString(2)));
                product.appendChild(desc);
                Element itemPrice = Doc.createElement("ITEM_PRICE"殿下);
                itemPrice.appendChild(Doc.createTextNode(Integer.toString(price)));
                product.appendChild(itemPrice);
                Element itemStatus = Doc.createElement("ITEM_STATUS"殿下);
                itemStatus.appendChild(Doc.createTextNode("UNSUBMITTED"殿下));
                product.appendChild(itemStatus);
                Element itemNo = Doc.createElement("ITEM_NO"殿下);
                itemNo.appendChild(Doc.createTextNode(Integer.toString(i)));
                product.appendChild(itemNo);
                Element itemQuantity = Doc.createElement("ITEM_QUANTITY"殿下);
                itemQuantity.appendChild(Doc.createTextNode(quantity));
                product.appendChild(itemQuantity);
                rs.close();
            }
        }
    }
}
The output screen, after adding an order, is shown in Figure 86.

![Order basket](image)

**Figure 86. Order basket**

2. The user can change the quantity for any items in his order by editing the quantity field and clicking on “Change quantities” button. This will cause the changeOrder method to be invoked. The user can also delete an item from the order by setting the quantity to 0. The code listing used to implement changeOrder follows:

```java
en = req.getParameterNames();
while (en.hasMoreElements()) {
    paramName = (String) en.nextElement();
    if (paramName.startsWith("_PNAME_")) {
        String prodid = paramName.substring(7); // End of while.
```
String quantity = req.getParameterValues(paramName)[0];
if (quantity != null && !quantity.equals("")) {
    try {
        q = Integer.parseInt(quantity);
    } catch (NumberFormatException nfe) {
        writeLog("Number format exception while parsing quantity", 1);
        nfe.printStackTrace(out);
        System.exit(-1);
    }

    NodeList elements = Doc.getElementsByTagName("PRODUCT");
    for(int i=0;i < elements.getLength(); i++) {
        Element orderItem = (Element) elements.item(i);
        if(prodid.equals(orderItem.getElementsByTagName("PRODUCT_ID").item(0).getFirstChild().getNodeValue()) && q == 0)
            orderItem.getParentNode().removeChild(orderItem);
        else {
            int val =
                Integer.parseInt(orderItem.getElementsByTagName("ITEM_QUANTITY").item(0).getFirstChild().getNodeValue());
            if(prodid.equals(orderItem.getElementsByTagName("PRODUCT_ID").item(0).getFirstChild().getNodeValue()) & & q == val)
                orderItem.getElementsByTagName("ITEM_QUANTITY").item(0).getFirstChild().setNodeValue(quantity);
        }
    }
}

3. The user can place his order by clicking on the “Place order” button as shown in Figure 86. Once a user places an order the shopping basket represented as a DOM tree is converted to an XML document which is then decomposed using dxxShredXML stored procedure of DB2 XML extender into ORDER_HEADERS and ORDER_ITEMS table. This is implemented using the placeOrder() method.

Once this is done, an XML document is generated using dxxGenXML stored procedure to send the order details to the supplier. This is implemented using the sendToSupplier method. In a real-world application once the XML document is generated and sent to the supplier, there will be processes running on the supplier side to receive the document and store it in a database on the supplier side.

But since ours is a sample scenario, the supplier details are kept in the same database as the order details. We now decompose the XML document to be sent in the sendToSupplier method, to store the data in
the SUPPLIER_ORDERS and the SUPP_ORDERDETAILS tables respectively.

placeOrder()

The placeOrder() method decomposes the shopping basket represented as an XML document into the ORDER_HEADERS and ORDER_ITEMS tables using the dxxShredXML stored procedure. The code listing for implementing this follows:

```java
String storedProcName = "db2xml.dxxShredXML";
String temp;
// Read the DAD file into a StringBuffer
BufferedReader in = new BufferedReader(new FileReader(DADFile));
while ((temp = in.readLine()) != null)
dad.append(temp);
in.close();
in = new BufferedReader(new FileReader(tempFileName));
while ((temp = in.readLine()) != null)
xml.append(temp);
in.close();
writeLog("The XML file is:", 0);
writeLog(xml.toString(), 0);
// prepare the CALL statement
CallableStatement cs;
String sql = "Call " + storedProcName + "(?,?,?,?)";
cs = con.prepareCall(sql);
// register the output parameters
int errCode = 0;
String msgText = "";
    cs.registerOutParameter(3, Types.INTEGER);
    cs.setInt(3, errCode);
    cs.registerOutParameter(4, Types.VARCHAR);
    cs.setString(4, msgText);
// set all parameters
    cs.setString(1, dad.toString());
    cs.setString(2, xml.toString());
// call the stored procedure
    con.setAutoCommit(false); // Enable transactions
    cs.execute();
    if (errCode != 0) {
        writeLog("Error executing stored procedure\n" + "Error code=" + errCode
                + "\nMessage =" + msgText, 0);
            throw new SQLException("Error executing stored procedure\n" + "Error code=" + errCode + "\nMessage =" + msgText);
    }
// Commit the transaction
    con.commit();
```

Chapter 7. XMLapp: implementation with XML Collections  229
con.setAutoCommit(true);

A listing of the DAD file used to decompose the shopping cart XML document follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\doc\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<prolog>xml version="1.0"?</prolog>
<doctype>!DOCTYPE person SYSTEM "C:\SG246130\XMLExtApp\DTDFiles\order.dtd" </doctype>
<root_node>
<element_node name ="ORDER">
    <RDB_node>
        <table name="xmlguru.order_headers" key="order_no"/>
        <table name="xmlguru.order_items" key="order_no item_no"/>
        <condition>
            xmlguru.order_headers.order_no=xmlguru.order_items.order_no
        </condition>
    </RDB_node>
<attribute_node name="ORDER_NO">
    <RDB_node>
        <table name="xmlguru.order_headers"/>
        <column name="ORDER_NO" type="varchar(30)"/>
    </RDB_node>
</attribute_node>
<attribute_node name="ORDER_STATUS">
    <RDB_node>
        <table name="xmlguru.order_headers"/>
        <column name="STATUS" type="varchar(30)"/>
    </RDB_node>
</attribute_node>
<element_node name="CUSTOMER_ORDER_NO">
    <text_node>
        <RDB_node>
            <table name="xmlguru.order_headers"/>
            <column name="CUSTOMER_ORDER_NO" type="char(30)"/>
        </RDB_node>
    </text_node>
</element_node>
<element_node name="ORDER_DATE">
    <text_node>
        <RDB_node>
            <table name="xmlguru.order_headers"/>
            <column name="ORDER_DATE" type="date"/>
        </RDB_node>
    </text_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
```
Chapter 7. XMLapp: implementation with XML Collections

```xml
<element_node name="USERNAME">
  <text_node>
    <RDB_node>
      <table name="xmlguru.order_headers"/>
      <column name="USERNAME" type="varchar(64)"/>
    </RDB_node>
  </text_node>
</element_node>

<element_node name="PRODUCT" multi_occurrence="YES">
  <element_node name="PRODUCT_ID">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="PRODUCT_ID" type="char(10)"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="ITEM_PRICE">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="ITEM_PRICE" type="decimal(10,2)"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="ITEM_STATUS">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="ITEM_STATUS" type="varchar(30)"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="ITEM_NO">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="ITEM_NO" type="smallint"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="ITEM_QUANTITY">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
      </RDB_node>
    </text_node>
  </element_node>
</element_node>
```
sendToSupplier()

This method will compose an XML document using the dxxGenXML stored procedure which is then decomposed into the supplier tables - SUPPLIER_ORDERS and SUPP_ORDERDETAILS.

A listing of the DAD file for composing the XML document follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
  <validation>NO</validation>
  <Xcollection>
    <prolog><?xml version="1.0"?></prolog>
    <doctype>!DOCTYPE ORDER SYSTEM "C:\DTDFiles\Supplier.dtd"</doctype>
    <root_node>
      <element_node name="ORDER">
        <RDB_node>
          <table name="xmlguru.order_headers"/>
          <table name="xmlguru.order_items"/>
          <condition>
            xmlguru.order_headers.order_no=xmlguru.order_items.order_no
          </condition>
        </RDB_node>
      </element_node>
      <attribute_node name="ORDER_NO">
        <RDB_node>
          <table name="xmlguru.order_headers"/>
          <column name="ORDER_NO"/>
          <condition>order_no='20000807202456408695000003'</condition>
        </RDB_node>
      </attribute_node>
      <attribute_node name="CUSTOMER_ORDER_NO">
        <RDB_node>
          <table name="xmlguru.order_headers"/>
          <column name="CUSTOMER_ORDER_NO"/>
        </RDB_node>
      </attribute_node>
      <attribute_node name="ORDER_DATE">
        <RDB_node>
        </RDB_node>
      </attribute_node>
    </root_node>
  </Xcollection>
</DAD>
```
<table name="xmlguru.order_headers"/>
  <column name="ORDER_DATE"/>
</RDB_node>
</attribute_node>
<attribute_node name="ORDER_STATUS">
  <RDB_node>
    <table name="xmlguru.order_headers"/>
    <column name="STATUS"/>
  </RDB_node>
</attribute_node>
<element_node name="PRODUCT" multi_occurrence="YES">
  <element_node name="PRODUCT_ID">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="product_id"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="STATUS">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="item_status"/>
      </RDB_node>
    </text_node>
  </element_node>
  <element_node name="QUANTITY">
    <text_node>
      <RDB_node>
        <table name="xmlguru.order_items"/>
        <column name="order_quantity"/>
      </RDB_node>
    </text_node>
  </element_node>
</element_node>
</element_node>
</Xcollection>
</DAD>

The DAD file for decomposing the XML document to supplier tables is given below.

<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
  <validation>NO</validation>
</DAD>
<Xcollection>
<prolog><?xml version="1.0"?></prolog>
<doctype>!DOCTYPE person SYSTEM "C:\DTFiles\supplier.dtd" </doctype>

<root_node>
<element_node name="ORDER">
  <RDB_node>
    <table name="xmlguru.supplier_orders" key="order_no"/>
    <table name="xmlguru.supp_order_details" key="order_no product_id"/>
    <condition>
      xmlguru.supplier_orders.order_no=xmlguru.supp_order_details.order_no
    </condition>
  </RDB_node>
  <attribute_node name="ORDER_NO">
    <RDB_node>
      <table name="xmlguru.supplier_orders"/>
      <column name="ORDER_NO" type="varchar(30)"/>
    </RDB_node>
  </attribute_node>
  <attribute_node name="CUSTOMER_ORDER_NO">
    <RDB_node>
      <table name="xmlguru.supplier_orders"/>
      <column name="CUSTOMER_ORDER_NO" type="varchar(40)"/>
    </RDB_node>
  </attribute_node>
  <attribute_node name="ORDER_DATE">
    <RDB_node>
      <table name="xmlguru.supplier_orders"/>
      <column name="ORDER_DATE" type="date"/>
    </RDB_node>
  </attribute_node>
  <attribute_node name="ORDER_STATUS">
    <RDB_node>
      <table name="xmlguru.supplier_orders"/>
      <column name="ORDER_STATUS" type="varchar(30)"/>
    </RDB_node>
  </attribute_node>
  <element_node name="PRODUCT" multi_occurrence="YES">
    <element_node name="PRODUCT_ID">
      <text_node>
        <RDB_node>
          <table name="xmlguru.supp_order_details"/>
          <column name="PRODUCT_ID" type="varchar(30)"/>
        </RDB_node>
      </text_node>
    </element_node>
  </element_node>
</element_node>
</root_node>
7.3.4 Order tracking

In this section we discuss the order tracking function.

7.3.4.1 Browse order servlet

The BrowseOrderServlet provides two different functionalities:

- If no order number is provided, the servlet will retrieve all the orders for the current user. The list of orders is composed as an XML document using the dxxGenXML stored procedure of DB2 XML extender. This XML document is transformed to HTML using a stylesheet orderList.xsl and sent back to the user.

- If an order number is provided as part of the HTTP parameter, the servlet will retrieve all the details about that order. This information is composed using the dxxGenXML stored procedure and transformed using a stylesheet orderDetail.xsl and sent back to the user.

The following code listing shows the doGet() method of the servlet. Based on whether the order number is null or not the orderList or orderDetail method is called respectively.

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {


try {
    HttpSession session = req.getSession(false);
    ...
    if (session == null)
        proc.process(loginURL, ssXSLURL, out);
    else {
        username = session.getValue("username").toString();
        if (username == null || username.length() == 0)
            proc.process(loginURL, ssXSLURL, out);
        else {
            String orderno = req.getParameter("orderno");
            if (orderno != null && orderno.length() > 0) {
                writeLog("Username" + username, 0);
                writeLog("Orderno" + orderno, 0);
                orderDetail(username, orderno);
                else {
                    orderList(username);
                }
            } out.close();
    }

7.3.4.2 Displaying the list of orders for a user
When no order number is specified the servlet will retrieve all the orders for
the current user. This is implemented by the orderList() method. This method
uses the dxxGenXML stored procedure to generate the XML document and a
stylesheet orderList.xsl is applied to it to transform it to HTML before being
sent back to user. The code listing follows:

```java
String procName = "db2xml.dxxGenXML"; // Name of stored procedure used
to generate XML document.
CallableStatement cs = con.prepareCall(sql);
...
```

```java
String sql = "Select
username,order_no,customer_order_no,order_date,ship_date from
xmlguru.order_headers where username='" +username + "' order by
username,order_no"; // override value
...
```

```java
cs.registerOutParameter (6, Types.INTEGER);
cs.setInt(6, actualNumOfDocs);
```
A listing of the DAD file used by the dxxGenXML stored procedure follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dad.dtd">
<DAD>
  <validation>NO</validation>
  <Xcollection>
    <SQL_stmt>
      Select username,order_no,customer_order_no,order_date,ship_date from xmlguru.order_headers
      order by username, order_no
    </SQL_stmt>
    <prolog><?xml version="1.0"?></prolog>
    <doctype>!DOCTYPE USERNAME SYSTEM "C:\DTDFiles\OrderList.dtd"</doctype>
    <root_node>
      <element_node name="USERNAME">
        <attribute_node name="CUSTOMER_NAME">
```

```xml
  cs.registerOutParameter (7, Types.INTEGER);
  cs.setInt(7, errCode);
  cs.registerOutParameter (8, Types.VARCHAR);
  cs.setString(8, msgText);
  cs.setString(1, dad.toString());
  cs.setString(2, resultTable);
  int overrideType = 1;  // overrideType: SQL_OVERRIDE
  cs.setInt(3, overrideType);
  cs.setString(4,override);
  int maxNumOfDocs = 100;  // just set to 100
  cs.setInt(5, maxNumOfDocs);
  ...
  //Set transaction isolation to repeatable read.
  con.setTransactionIsolation(java.sql.Connection.TRANSACTION_REPEATABLE_READ);
  Statement stmt = con.createStatement();
  //Delete any rows in the result table so that the document we generate
  //is the only one in the table.
  stmt.executeUpdate("Delete From "+resultTable);
  cs.execute();
  ...
  sql = "select db2xml.content(db2xml.XMLClob(doc), "'+tempFileName1+'")
  from result_tab";
  stmt = con.createStatement();
  stmt.executeQuery(sql);
  con.close();
  writeLog("Query for order list executed",0);
  stmt = null;
  ProcessXML proc = new ProcessXML();
  proc.process(tempFileName1,ssListURL,out);
  ...
```
7.3.4.3 Displaying the details of a particular order

When an order number is specified, the servlet will retrieve the full details for that order number. This is implemented by the orderDetails() method. This method again uses the dxxGenXML stored procedure to generate the XML document and a stylesheet orderDetail.xsl is applied to it to transform it to HTML before being sent back to user. The code listing is almost similar to the one for orderList except for the SQL statement used for overriding:

```sql
String override = "Select o.order_no, username, customer_order_no, order_date,p.product_id,product_name,description,i.order_id, item_price, item_status,item_no,shipping_date,order_quantity,total_price from xmlguru.order_headers o,xmlguru.products p,table(select substr (char(timestamp(generate_unique())),16) as order_id, order_no, product_id, item_price,item_status,item_no,ship_date as shipping_date, order_quantity, item_price*order_quantity as total_price from xmlguru.order_items) as i where o.order_no=i.order_no and p.product_id=i.product_id and o.order_no='"+orderno +"' order by order_no,product_id,order_id\0";
```
A listing of the DAD file used to generate the XML document follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>
Select o.order_no, username, customer_order_no,order_date, p.product_id, product_name,description,i.order_id,item_price,item_status,item_no,shipping_date,order_quantity,total_price from xmlguru.order_headers o, xmlguru.products p, table(select substr(char(timestamp(generate_unique())),16) as order_id, order_no,product_id,item_price,item_status,item_no,ship_date as shipping_date,order_quantity,item_price*order_quantity as total_price from xmlguru.order_items) as i where o.order_no=i.order_no and p.product_id=i.product_id order by order_no, product_id, order_id
</SQL_stmt>
<prolog><?xml version="1.0"?></prolog>
<doctype>!DOCTYPE CUSTOMER SYSTEM "C:\DTDFiles\OrderDetail.dtd"</doctype>
[root_node]
<element_node name="CUSTOMER">
  <attribute_node name="CUSTOMER_NAME">
    <column name="USERNAME"/>
  </attribute_node>
</element_node>
<element_node name="ORDER">
  <attribute_node name="ORDER_NO">
    <column name="ORDER_NO"/>
  </attribute_node>
  <attribute_node name="CUSTOMER_ORDER_NO">
    <column name="CUSTOMER_ORDER_NO"/>
  </attribute_node>
  <attribute_node name="ORDER_DATE">
    <column name="ORDER_DATE"/>
  </attribute_node>
</element_node>
<element_node name="PRODUCTS" multi_occurrence="YES">
  <element_node name="PRODUCT_ID">
    <text_node>
      <column name="PRODUCT_ID"/>
    </text_node>
  </element_node>
  <element_node name="PRODUCT_NAME">
    <text_node>
      <column name="PRODUCT_NAME"/>
    </text_node>
  </element_node>
</element_node>
</Xcollection>
</DAD>
```
<element_node name="DESCRIPTION">
    <text_node>
        <column name="DESCRIPTION"/>
    </text_node>
</element_node>
<element_node name="ITEM_PRICE">
    <text_node>
        <column name="ITEM_PRICE"/>
    </text_node>
</element_node>
<element_node name="TOTAL_PRICE">
    <text_node>
        <column name="TOTAL_PRICE"/>
    </text_node>
</element_node>
<element_node name="ITEM_STATUS">
    <text_node>
        <column name="ITEM_STATUS"/>
    </text_node>
</element_node>
<element_node name="ITEM_NO">
    <text_node>
        <column name="ITEM_NO"/>
    </text_node>
</element_node>
<element_node name="SHIP_DATE">
    <text_node>
        <column name="shipping_date"/>
    </text_node>
</element_node>
<element_node name="ORDER_QUANTITY">
    <text_node>
        <column name="ORDER_QUANTITY"/>
    </text_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
The output screen with details of a particular order is shown in Figure 87.

![Order details](image)

Figure 87. Order details

### 7.3.5 Supplier

When an order is made by an user the supplier tables are updated with the details of the new order by decomposing an XML document describing the order. The supplier can then view the orders and specify a shipping date for individual items within an order. There is no XML involved in implementing this part.
7.3.5.1 Supplier order servlet

The SupplierOrderServlet will list all the orders available in SUPPLIER_ORDERS table and the supplier id of the supplier who is going to fulfill that order. This servlet will directly output HTML. The code listing for this servlet follows:

```java
res.setContentType("text/html");
out.print("<HTML><HEAD><META http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
"http-equiv="Expires" content="0"/>
");
out.print("<STYLE TYPE="text/css"> BODY{font-family : Verdana,sans-serif;}
formText{font-family : Verdana,sans-serif;}TD{font-family : Verdana,sans-serif; font-weight:
bold;});
out.print("<STYLE TYPE="text/css"> BODY{font-family : Verdana,sans-serif;font-size:100%;}
formText{font-family : Verdana,sans-serif;}TD{font-family : Verdana,sans-serif; font-weight:
bold;}
";
out.print("<h1><center>Suppliers side<br>ORDERS UNDER PROCESS</center></h1>

String sql = "Select distinct o.order_no,o.order_status,supplier_id from
xmlguru.supplier_orders o,xmlguru.supp_order_details d,xmlguru.products
where o.order_no = d.order_no and d.product_id =
xmlguru.products.product_id order by o.order_no";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
    orderno = rs.getString(1);
    status = rs.getString(2);
    supid = rs.getString(3);
    out.println("<tr><td><a href="BrowseSupplierServlet?ORDER_NO=" +
    orderno + "&SUPPLIER_ID=" + supid + ">
    "+ orderno + "+</a></td>
    out.println("</td><td>" + status + "</td>
    out.println("</tr>
}
out.println("</TBODY></TABLE><br></BODY></HTML>";
```

242 Integrating XML with DB2 XML Extender and DB2 Text Extender
Chapter 7. XMLapp: implementation with XML Collections

The output screen for this servlet is shown in Figure 88.

![Output Screen](image)

**Figure 88. Supplier orders**

### 7.3.5.2 Browse supplier servlet

The BrowseSupplierServlet is allows a supplier to view the item details within an order. This servlet is invoked by clicking on the order number in Figure 88. The code listing for this servlet follows:

```java
orderno = req.getParameter("ORDER_NO");
supid = req.getParameter("SUPPLIER_ID");
res.setContentType("text/html");
...
out.print("</CENTER></b><br></br><TABLE BORDER="2"
ALIGN="CENTER"><THEAD><th>Order No.</th><th>Customer Order No</th><th>Order Date</th></THEAD><TBODY>"
```
String sql = "Select customer_order_no,order_date from
xmlguru.supplier_orders where order_no ='"+orderno+"'";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(sql);
rs.next();
out.print("<tr><td>"+orderno+"</td><td>"+rs.getString(1)+"</td><td>"+rs
.getString(2)+"</td></tr></TBODY></TABLE>");
rs.close();
out.print("<BR><HR width=\"100%\"><BR>");
sql = "Select supplier_part_no, xmlguru.supp_order_details.product_id,
product_name,status,ship_date,quantity from xmlguru.supp_order_details,
xmlguru.products where xmlguru.supp_order_details.product_id=
xmlguru.products.product_id and xmlguru.supp_order_details.order_no=
'"+orderno+"' and xmlguru.products.supplier_id='"+supid+"'";
out.println("<FORM action=\"com.itso.servlets.UpdateServlet\"
method=\"GET\"><input type=\"HIDDEN\" name=\"ORDER_NO\"
value=\""+orderno+"\">");
out.println("<input type=\"HIDDEN\" name=\"SUPPLIER_ID\"
value=\""+supid+"\">");
out.println("<TABLE BORDER=\"1\"><THEAD><th>Shipped today</th><th>Part
No</th><th>Product ID</th><th>Product Name</th><th>Status</th><th>Ship
Date</th><th>Quantity</th>");
stmt = con.createStatement();
rs = stmt.executeQuery(sql);
while (rs.next()) {
if(rs.getString(5) == null)
out.println("<tr align=\"center\"><td><input type=\"checkbox\"
name=\"PART_"+rs.getString(1)+"\"
value=\""+rs.getString(2)+"\"></td><td>"+rs.getString(1)+"</td><td>"+rs
.getString(2)+"</td><td>"+rs.getString(3)+"</td><td>"+rs.getString(4)+"
</td><td>NOT SHIPPED</td><td>"+rs.getInt(6)+"</td></tr>");
else
out.println("<tr
align=\"center\"><td></td><td>"+rs.getString(1)+"</td><td>"+rs.getStrin
g(2)+"</td><td>"+rs.getString(3)+"</td><td>"+rs.getString(4)+"</td><td>
"+rs.getString(5)+"</td><td>"+rs.getInt(6)+"</td></tr>");
}
out.println("</TBODY></TABLE>");
out.println("<br><CENTER><input value=\"MODIFY\" name=\"modify\"
type=\"submit\"></CENTER></FORM>");
out.println("<BR><HR width=\"100%\"><BR>");
out.println("<center>Click <A href=\"SupplierOrderServlet\"> Here </A>
to go Orders List</center>");
out.println("<br></BODY></HTML>");
...

244

Integrating XML with DB2 XML Extender and DB2 Text Extender


The output screen or this servlet is shown in Figure 89.

![Supplier Order Details](image)

Figure 89. Supplier order details

### 7.3.5.3 Update servlet

The update servlet is invoked when you check Shipped today check box and click on the Modify button as shown in Figure 89. This servlet will update the SHIP_DATE column in the ORDER_ITEMS and SUPP_ORDER_DETAILS to CURRENT DATE and the STATUS column to SHIPPED. In addition to this it checks if all the orders for that order number have been shipped, if so it updates the STATUS column of ORDER_HEADERS and SUPPLIER_ORDERS to SHIPPED, otherwise it sets it to PARTIALLY SHIPPED. The code listing for this servlet follows:
String sql = "Update xmlguru.supp_order_details set ship_date = CURRENT
DATE,status='SHIPPED' where order_no = ? and product_id = ?";
String sql1 = "Update xmlguru.order_items set ship_date = CURRENT
DATE,item_status='SHIPPED' where order_no = ? and product_id = ?";
String sql2 = null;
String sql3 = null;
ResultSet rs = null;
int count1 = -1;
int count2 = -1;
PrintWriter out = res.getWriter();
res.setContentType("text/html");
en = req.getParameterNames();
orderno = req.getParameter("ORDER_NO");
try {
    while (en.hasMoreElements()) {
        paramName = (String) en.nextElement();
        if (paramName.startsWith("PART")) {
            prodid = req.getParameter(paramName);
            con.setAutoCommit(false);
            stmt = con.prepareStatement(sql);
            stmt.setString(1, orderno);
            stmt.setString(2, prodid);
            stmt.executeUpdate();
            stmt = con.prepareStatement(sql1);
            stmt.setString(1, orderno);
            stmt.setString(2, prodid);
            stmt.executeUpdate();
            sql2 = "Select count(*) from xmlguru.supp_order_details where order_no
            ="+orderno+""
            sql3 = "Select count(*) from xmlguru.supp_order_details where order_no=
            "+orderno+" and ship_date is null"
            statement = con.createStatement();
            rs = statement.executeQuery(sql2);
            rs.next();
            count1 = rs.getInt(1);
            rs.close();
            statement.close();
            rs = null;
            statement = con.createStatement();
            rs = statement.executeQuery(sql3);
            rs.next();
            count2 = rs.getInt(1);
            statement.close();
            if (count2 ==0) {
                stmt = con.prepareStatement("Update xmlguru.supplier_orders set
order_status = 'SHIPPED' where order_no=?" );
stmt.setString(1, orderno);
stmt.executeUpdate();
stmt = con.prepareStatement("Update xmlguru.order_headers set status = 'SHIPPED' where order_no=? ");
stmt.setString(1, orderno);
stmt.executeUpdate();
}
else if (count2 < count1) {
stmt = con.prepareStatement("Update xmlguru.supplier_orders set order_status = 'PARTIALLY SHIPPED' where order_no=? ");
stmt.setString(1, orderno);
stmt.executeUpdate();
stmt = con.prepareStatement("Update xmlguru.order_headers set status = 'PARTIALLY SHIPPED' where order_no=? ");
stmt.setString(1, orderno);
stmt.executeUpdate();
}
con.commit();
con.setAutoCommit(false);
Integrating XML with DB2 XML Extender and DB2 Text Extender
Chapter 8. XMLapp: installing XML Columns version in WAS

In this section we describe the installation and configuration steps that are necessary in order to run the ITSO XML Application build using XML Columns. It is assumed that the basic infrastructure — WebSphere, DB2, and IBM HTTP Server — are installed and functional (Table 13).

Table 13. Products and versions used

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
<th>Recommended directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM HTTP Server</td>
<td>1.3.6.2</td>
<td>C:\IBM HTTP Server</td>
</tr>
<tr>
<td>IBM JDK</td>
<td>1.1.7p (08/23)</td>
<td>C:\jdk1.1.7</td>
</tr>
<tr>
<td>DB2 Universal Database for Windows using JDBC</td>
<td>7.1 1.0</td>
<td>C:\SQLLIB</td>
</tr>
<tr>
<td>DB2 UDB XML Extender</td>
<td>7.1 1.0</td>
<td>C:\dxx</td>
</tr>
<tr>
<td>IBM WebSphere Application Server Standard Edition</td>
<td>3.02</td>
<td>C:\WebSphere</td>
</tr>
</tbody>
</table>

Sample Code on the Internet

The sample code for this redbook is available as the SG246130.zip file on:

ftp://www.redbooks.ibm.com/redbooks/SG246130/

Download SG246130.zip and read the README.TXT file included in the zip file. Any updates to the book will also be found there. See Appendix E, "Using the additional material" on page 297.

8.1 Files contained in SG246130 Columns directory

Table 14 describes the content of the zip file SG246130 columns directory.

Table 14. SG246130 Columns directory

<table>
<thead>
<tr>
<th>File or directory</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMe.txt</td>
<td>Describes the content of the zip file and where the files or directories it contains should be restored</td>
</tr>
<tr>
<td>DAXSL.jar</td>
<td>The Java classes necessary for the DefaultApplyXSL servlet transforming XML into HTML</td>
</tr>
<tr>
<td>XMLAppConfig.xml</td>
<td>Our WebSphere Application Server configuration exported with the XMLConfig tool shipped with WebSphere (for information)</td>
</tr>
</tbody>
</table>
The following sections provide instructions regarding how to use these files and directories to run the XMLapp application.

### 8.2 Installing the databases

The ITSO XML Application requires 1 database (in addition to the WAS database that is required by WebSphere run-time). The application database is called **L301**.

<table>
<thead>
<tr>
<th>File or directory</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ddl\</code></td>
<td>The database scripts and command files used for creating the database elements required by the application</td>
</tr>
<tr>
<td><code>XMLApp\web\dtd\</code></td>
<td>The DTD files validating some of the XML files used in the application (for information)</td>
</tr>
<tr>
<td><code>XMLApp\web\jsp\</code></td>
<td>The jsp files developed for this application</td>
</tr>
<tr>
<td><code>XMLApp\servlets\itso\</code></td>
<td>The Java code and Java classes developed for this application (in three packages itso.xmlapp.db2beans, itso.xmlapp.db2const, itso.xmlapp.db2servlets)</td>
</tr>
<tr>
<td><code>xml\</code></td>
<td>13 XML files used for most of the output screens used in the application</td>
</tr>
<tr>
<td><code>xsl\</code></td>
<td>8 XSL files and 2 images used to transform the XML information into HTML</td>
</tr>
</tbody>
</table>

The JSP files of the ITSO XML Application for WebSphere Application Server use a hard-coded userid/password combination for database access. To avoid running into trouble, we suggest that the first thing you should do is to create a new user for your system. The userid is **xmlguru**, and the password is **xmlguru**, both in lowercase. If you do not want to set the database user privileges for this user using the DB2 utilities, we suggest that you make this user an administrative one.

You could run also this application with any userid and password but then you need to update the 5 JSP files and the class Constant containing all the constant values used by the beans and servlets in this application.
In order to create the database, follow the steps given here.

1. Create a directory called SG246130 in C:\. The application needs this directory to be present on C:\

2. Unzip the file SG246130.zip to the directory C:\SG246130 created in step 1. Ensure that the directory structure within the zip file is intact.

3. The application database L301 is created manually by using the DB2 Command Window and entering the command:

   db2 create db L301

   This command is also contained in the xmlapp1.ddl script file.

   cd \SG246130\Columns\ddl
   db2 -f xmlapp1.ddl

4. You must now bind DB2 XML Extender and CLI to this database by issuing the following commands in the Command Window:

   db2 connect to L301
   cd /d %DB2DXXPATH%\bnd
   db2 bind @dxxbind.lst
   cd /d %DB2PATH%\bnd
   db2 bind @db2cli.lst
   db2 terminate

5. The L301 database must now be enabled for XML Extender with the dxxadm administration command. The command to enable the database is:

   dxxadm enable_db L301

   This command is also contained in the EnableDatabase.bat batch file.

6. The SQL definition statements to create the tables in the database L301 are included in the xmlapp2.ddl script file located in the ddl directory.

   The statements for populating the tables are in the data.ddl script file. You should run these two scripts by executing the following commands in the DB2 Command Window in this order:

   cd \SG246130\Columns\ddl
   db2 -f xmlapp2.ddl -t
   db2 -f data.ddl -t

7. The last step is to enable the XML Column, orderdoc, in the table, orderxml, using the orderColdad DAD file. The command for enabling this column is:

   dxxadm enable_column L301 xmlguru.orderxml orderdoc orderCol.dad -r
   order_no
In this command, \texttt{orderCol.dad} must be fully qualified with the path name if it is not in the current directory.

### 8.3 Configuring WebSphere Application Server 3.02

Several things must be done in order to be able to run the ITSO XML Application successfully in WebSphere. We start by defining the datasource and JDBC driver to access the database L301.

The configuration of WebSphere Application Server can be done using the supplied configuration file or manually through the administration console.

#### 8.3.1 Using XMLConfig tool

We need to configure WebSphere Application Server by importing the configuration file XMLAppConfig.xml. Perform the following steps to import the configuration.

1. Open the file XMLAppConfig.xml for editing using your favorite text editor.
2. In the file replace \texttt{YOURNODENAME} with node name on your machine. This is case sensitive.
3. In the XMLAppConfig.xml there are many references to the drive and directory where WebSphere is installed. If these references are different from where you installed WebSphere, then make appropriate changes to those references.
4. Save and close the file.
5. Be sure that WebSphere service is started, otherwise start IBM WS AdminServer under services.
6. Open a Command prompt
7. From the prompt give the following command:

   ```bash
   XMLConfig -import C:\SG246130\columns\XMLAppConfig.xml -adminNodeName YOURNODENAME
   ```

   In this command, \texttt{YOURNODENAME} is the node name on your machine.
8. Next, you must install the driver to your host. If the WebSphere Administrative Console is running, restart it in order to refresh the configuration. In the Topology page of WebSphere Administrative Console, select the DB2Driver object, and in the pop-up menu, select Install. In the window that is displayed, select your host name and the \texttt{c:\sqllib\java\db2java.zip} file containing DB2 JDBC classes.
9. Open the Administrator's console of WebSphere and click the **Topology** tab; you can see the XMLExtApp application. Start your application and ensure that the application is running.

Continue in 8.3.3, “Install the Java classes and JSP files” on page 259.

### 8.3.2 Manual configuration

These steps are here to show how you would manually create the WebSphere configuration.

#### 8.3.2.1 Creating a datasource and a JDBC driver

You need to install the JDBC driver for DB2 before defining the datasources for WebSphere. In the Types view, select JDBCDrivers; and in the pop-up menu, select Create. We will call it DB2Driver, and will use the implementation class `com.ibm.db2.jdbc.app.DB2Driver`.

After the driver for DB2 has been defined, you can define the datasource L301. The datasource is added in the **Types** view by selecting **Datasources->Create**.

Next, you must install the driver to your host. In the **Topology** page of WebSphere Administration Console, select the DB2Driver object, and in the pop-up menu, select **Install**. In the window that is displayed, select your host name and the jar file containing DB2 JDBC classes (db2java.zip).

Now you can use your datasource in your Web application.

#### 8.3.2.2 Changing the classpath of the Default Server

The jar file DAXSL.jar is required for our application. We can either place this jar file in the classpath of the Web Application or in the classpath of the Default Server. We decide to place it in the classpath of the Default Server. Therefore, we can put the jar file in any directory and then refer to it in the **Command line arguments** parameter of the Default Server.

Suppose that we create a new directory C:\jars where we add the file DAXSL.jar. Therefore, we must add the following argument in the command line arguments:

```
-classpath c:\jars\DAXSL.jar
```

As a result, the command line with all arguments for the Default Server is:

```
-mx128 -classpath c:\jars\DAXSL.jar
```
To set up this startup string, go to the **Topology** page, then select **Default Server** and the **General** page.

### 8.3.2.3 Creation of the XMLapp Web application

WebSphere Application Server uses the concept of a *Web application* to separate applications with different requirements from each other. It is possible to use the default application of WebSphere for our application, but we decided to create a new application called *xmlapp*, because some changes are needed in the application configuration.

Using the WebSphere Administration Console, in the **Topology** page, expand the **WebSphereAdminDomain** tree until you can see the **servletEngine**. Select that, and in the pop-up menu select **Create->Web Application**. Fill in the values as shown in Figure 90. Be sure to remove the default /webapp portion of the Web application path. Click **Create**.

![Create Web Application](image)

*Figure 90. Web application definition*

### 8.3.2.4 Creation of the DefaultApplyXSL servlet

The next thing to do is to define the DefaultApplyXSL servlet called DAXSL. In the **Topology** view, select your newly created application, and in the pop-up menu select **Create->Servlet**. Fill in the values as shown in Figure 91.
After you have created the DAXSL servlet, you need to set up its properties. In the **Topology** view, select the DAXSL servlet, and enter the values as shown in Figure 92.

**Figure 91. DAXSL servlet definition**

**Figure 92. DAXSL properties**
Now select the xmlapp Web application, and define the MIME-filter for text/xml content type as shown in Figure 93.

![Figure 93. MIME filter definition](image)

### 8.3.2.5 Creation of the JSP1.0 enabler

Because parts of the ITSO XML Application are implemented as Java Server Pages, you need to install a JSP enabler for the xmlapp application. In the Tasks page, select **Add a JSP enabler**, and click the **Start Task** button (the green button in the toolbar). Set the values as shown in Figure 94.
8.3.2.6 Creation of the other named servlets

Note

It is always a good idea to use named servlets. When the servlets are named, your servlet package name or class name can be changed without changes in the applications that are using the servlet. Another reason is security: Only servlets with names can be invoked, and you can use the security features of WebSphere Application Server to protect named servlets.
You have already created a named servlet called DAXSL (see Figure 91 on page 255). Using the same procedure, create another ten servlets in the xmlapp application with the information provided in Table 15.

Table 15. Defining servlets used in xmlapp Web application

<table>
<thead>
<tr>
<th>Name</th>
<th>Fully qualified name</th>
<th>Servlet Web path</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrowseOrderServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/BrowseOrderServlet</td>
</tr>
<tr>
<td></td>
<td>BrowseOrderServlet</td>
<td></td>
</tr>
<tr>
<td>BrowseSupplierServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/BrowseSupplierServlet</td>
</tr>
<tr>
<td></td>
<td>BrowseSupplierServlet</td>
<td></td>
</tr>
<tr>
<td>CustomerStorageServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/CustomerStorageServlet</td>
</tr>
<tr>
<td></td>
<td>CustomerStorageServlet</td>
<td></td>
</tr>
<tr>
<td>SupplierStorageServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/SupplierStorageServlet</td>
</tr>
<tr>
<td></td>
<td>SupplierStorageServlet</td>
<td></td>
</tr>
<tr>
<td>LogonServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/LogonServlet</td>
</tr>
<tr>
<td></td>
<td>LogonServlet</td>
<td></td>
</tr>
<tr>
<td>RegisterServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/RegisterServlet</td>
</tr>
<tr>
<td></td>
<td>RegisterServlet</td>
<td></td>
</tr>
<tr>
<td>RegisterChangeServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/RegisterChangeServlet</td>
</tr>
<tr>
<td></td>
<td>RegisterChangeServlet</td>
<td></td>
</tr>
<tr>
<td>RegisterChangeOutputServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/RegisterChangeOutputServlet</td>
</tr>
<tr>
<td></td>
<td>RegisterChangeOutputServlet</td>
<td></td>
</tr>
<tr>
<td>UpdateB2BServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/UpdateB2BServlet</td>
</tr>
<tr>
<td></td>
<td>UpdateB2BServlet</td>
<td></td>
</tr>
<tr>
<td>UpdateServlet</td>
<td>itso.xmlapp.db2servlets.</td>
<td>/xmlapp/servlet/UpdateServlet</td>
</tr>
<tr>
<td></td>
<td>UpdateServlet</td>
<td></td>
</tr>
</tbody>
</table>
8.3.3 Install the Java classes and JSP files

You must also copy the directory XMLapp with its content into the directory C:\WebSphere\AppServer\hosts\default_host\, where C:\WebSphere\AppServer is the installation directory of WebSphere Application Server.

Edit the Constant.java located in the directory c:\WebSphere\AppServer\hosts\default_host\XMLApp\servlets\itso\xmlapp\db2const. Then update YourNodeName with the node name of your machine. In a command prompt, compile the Java file:

    javac Constant.java

You must also copy the DAXSL.jar file to the c:\jars directory.

8.3.4 Install the XML and XSL files

We now must copy the XML and XSL files in the appropriate directories so that either the WebSphere Application Server or the IBM HTTP Server can find the files they require for the application.

1. The XML files: Copy the xsl directory and the 13 XML files it contains in the directory C:\IBM HTTP Server\htdocs, where C:\IBM HTTP Server is the installation directory of the IBM HTTP Server.

2. The XSL files: Copy the xsl directory and the 10 XSL files it contains in the directory C:\IBM HTTP Server\htdocs, where C:\IBM HTTP Server is the installation directory of the IBM HTTP Server.

Figure 95 shows the directory structure where the files should be placed.
Figure 95. Directory view for XMLapp
8.4 Running the application

Once you have completed these steps, you can access the application by typing the following URL in a Web browser.

Customer: http://<server>/xmlapp/xml/MainMenu.xml

Supplier: http://<server>/supplier/SupMainMenu.jsp

You should first start on the customer side, as the supplier side works with the orders submitted by the customers.

You can also use the SG246130.html page. This page has links to the XML Columns and XML Collections applications.
Chapter 9. XMLapp: installing XML Collections version in WAS

In this section we describe the necessary installation and configuration steps that must be done in order to run the ITSO XML Application build using XML Collections. It is assumed that the basic infrastructure — WebSphere, DB2, and IBM HTTP Server — are installed and functional (see Figure 16).

Table 16. Products and Versions used

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
<th>Recommended Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM HTTP Server</td>
<td>1.3.6.2</td>
<td>C:\IBM HTTP Server</td>
</tr>
<tr>
<td>IBM JDK</td>
<td>1.1.7p (08/23)</td>
<td>C:\jdk1.1.7</td>
</tr>
<tr>
<td>DB2 Universal Database for Windows using JDB</td>
<td>7.1</td>
<td>C:\SQLLIB</td>
</tr>
<tr>
<td>DB2 UDB XML Extender</td>
<td>7.1</td>
<td>C:\dxx</td>
</tr>
<tr>
<td>IBM WebSphere Application Server Standard</td>
<td>3.02</td>
<td>C:\WebSphere</td>
</tr>
</tbody>
</table>

Sample Code on the Internet
The sample code for this redbook is available as the SG246130.zip file on:
ftp://www.redbooks.ibm.com/redbooks/SG246130/
Download SG246130.zip and read the README.TXT file included in the zip file. Any updates to the book will also be found there. See Appendix E, "Using the additional material" on page 297

9.1 The files contained in SG246130 collections directory

Table 17 describes the content of the zip file SG246130.ZIP collections directory

Table 17. SG246130 collections directory

<table>
<thead>
<tr>
<th>File or directory</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMe.txt</td>
<td>Describes any updates to the book.</td>
</tr>
<tr>
<td>db_setup.cmd</td>
<td>The file used for creating the database and configuring it for the application.</td>
</tr>
<tr>
<td>create_tables.ddl</td>
<td>The file used for creating the tables used by the application.</td>
</tr>
<tr>
<td>product.ddl</td>
<td>The file used for importing data to products table.</td>
</tr>
</tbody>
</table>
In the sections which follow, you will find instructions on how to get the application up and running.

### 9.2 Installing the databases

The application requires a database called **ORDER** to be created and populated with a set of tables.

---

**Important**

The application uses a hard-coded userid/password combination for database access. To avoid running into trouble, we suggest that you should create a new administrative user for your system. The userid is **xmlguru**, and the password is **xmlguru**, both in **lowercase**.

---

In order to create the database, follow the steps given here.

1. Create a directory called SG246130 in C:\. The application needs this directory to be present on C:\
2. Unzip the file SG246130.zip to the directory C:\SG246130 created in step 1. Ensure that the directory structure within the zip file are intact.
3. Open a DB2 command window by clicking on **START->PROGRAMS->IBM DB2->COMMAND WINDOW**.
4. In the DB2 command window change directory to C:\SG246130\collections.

5. Run the script db_setup.cmd. The contents of this script are given here. Make sure the script runs without any errors.

   ```
   rem
   rem db_setup --- creating and populating database
   rem
   db2 "create database ORDER"
   db2 "connect to order user xmlguru using xmlguru"
   @echo -------- Setting up the database ORDER for XML extender --------
   cd /d %DB2DXXPATH%nd
   db2 "bind @dxxbind.lst"
   cd /d %DB2PATH%nd
   db2 "bind @db2cli.lst"
   db2 "terminate"
   dxxadm enable_db order -l xmlguru -p xmlguru
   db2 "terminate"
   ```

6. Run the script create_tables.ddl from C:\SG246130\collections directory by giving the command given here. Make sure the script runs without any errors.

   ```
   db2 -f create_tables.ddl -t
   ```

7. Run the script create_tables.ddl from C:\SG246130\collections directory by giving the command given here. Make sure the script runs without any errors.

   ```
   db2 -f product.ddl -t
   ```

Now you need to copy the files to the appropriate directories by following the steps given here.

1. Copy the directory XML_Coll under C:\SG246130\collections to the document root of your Web server.

2. Copy the directory XSL_Coll under C:\SG246130\collections to the document root of your Web server.

3. Copy the directory DTDFiles under C:\SG246130\collections to the document root of your Web server.

4. Copy the directory XMLExtApp under C:\SG246130\collections to the ```install_dir\WebSphere\AppServer\hosts\default_host``` where ```install_dir``` is the directory where you installed WebSphere Application Server.

---

Chapter 9. XMLapp: installing XML Collections version in WAS 265
Figure 96 shows the directory structure where the files should be placed.

![Directory view of XMLExtApp]

Figure 96. Directory view of XMLExtApp

We need to configure WebSphere Application Server by importing the configuration file XMLExtAppConfig.xml. Perform the following steps to import the configuration.
1. Open the file XMLExtAppConfig.xml for editing using notepad.

2. In the file replace **YOURNODENAME** with node name on your machine. This is case sensitive.

3. In the XMLExtAppConfig.xml there are many references to the drive and directory where WebSphere is installed. If these references are different from where you installed WebSphere, then make appropriate changes to those references.

4. Save and close the file.

5. Open a MS-DOS prompt

6. From the prompt give the following command:

   ```
   XMLConfig -import C:\SG246130\collections\XMLExtAppConfig.xml
   -adminNodeName YOURNODENAME
   ```

   In this command, **YOURNODENAME** is the node name on your machine.

7. Open Administrator's console of WebSphere and click the **Topology** tab. You can see the XMLExtApp application as shown in Figure 97. Ensure that the application is running.

![Figure 97. Administrative console](image)
9.3 Running the application

Once you have completed these steps, you can access the application by typing in the following URL in a Web browser.

Customer: http://server/XMLExtApp/servlet/MainMenuServlet
Supplier: http://server/XMLExtApp/servlet/SupplierOrderServlet

You should first start by the customer side, as the supplier side works with the orders submitted by the customers.

You can also use the SG246130.html page. This page has links to the XML Columns and XML Collections applications.

9.4 Application logging

The application will log messages to log files in the C:\SG246130\collections\log directory. The log file name is the same as the servlet name with a “log” extension. All the dynamically generated XML documents are also written to this directory. The file name of the XML files is given by the tempFileName parameter in the properties file for each servlet.
Appendix A. XMLapp: installing XML Columns version in VAJ

In this section we describe the necessary installation and configuration steps that you must execute in order to run the XMLapp application in the WebSphere Test Environment of Visual Age for Java. It is assumed that the basic infrastructure — DB2, DB2 XML Extender and VisualAge 3.02 Enterprise Edition — are installed and functional (see Table 18). The installation of DB2 XML Extender has been described in this book; please refer to Chapter 2, “DB2 XML Extender” on page 33 for more information regarding the DB2 XML Extender installation. The installation of DB2 and VisualAge for Java has already been largely described in other redbooks and will not be repeated here. Refer, for example, to Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java, SG24-5755-00.

Table 18. Products and versions used

<table>
<thead>
<tr>
<th>Product</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisualAge for Java Enterprise Edition</td>
<td>3.02</td>
</tr>
<tr>
<td>DB2 Universal Database for Windows</td>
<td>7.1</td>
</tr>
<tr>
<td>DB2 XML Extender</td>
<td>7.1</td>
</tr>
</tbody>
</table>

Sample Code on the Internet

The sample code for this redbook is available as the SG246130.zip file on:
ftp://www.redbooks.ibm.com/redbooks/SG246130/

Download SG546130.zip and read the README.TXT file included in the zip file. Any updates to the book will also be found there. See Appendix E, “Using the additional material” on page 297.

A.1 The content of the zip file

Table 19 describes the content of the zip file SG246130.ZIP VAJ directory.

Table 19. SG246130 VAJ directory

<table>
<thead>
<tr>
<th>File or directory</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMe.txt</td>
<td>Describes the content of the zip file and where the files or directories it contains should be restored</td>
</tr>
</tbody>
</table>
The following sections provide instructions regarding how to use these files and directories to run the XMLapp application in VisualAge for Java.

### A.2 Installing the databases

The XMLapp application requires one database that we called L301.

---

**Important**

The JSP files of the XMLapp application use a hard-coded userid/password combination for database access. To avoid experiencing any problems, we suggest that the first thing you should do is to create a new user for your system. The userid is `xmlguru`, and the password is `xmlguru`, both in lowercase. If you do not want to set the database user privileges for this user using the DB2 utilities, we suggest that you make this user an administrative one.

You could also run this application with any userid and password, but then you would need to update the 5 JSP files and the class Constant containing all the major constants used by the beans and servlets in this application.

---

In order to create the database, follow the steps given below.

1. Create a directory called SG246130 in C:\. The application needs this directory to be present on C:\
2. Unzip the file SG246130.zip to the directory C:\SG246130 created in step 1. Ensure that the directory structure within the zip file are intact.
3. The application database L301 is created manually by using the DB2 Command Window and entering the command:
   ```
   db2 create db L301
   ```

---

<table>
<thead>
<tr>
<th>File or directory</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG246130_VAJ.jar</td>
<td>All the code of the application exported from the VisualAge for Java project</td>
</tr>
<tr>
<td>DataSourceFactory.jar</td>
<td>The required classes for configuring datasources in VisualAge for Java</td>
</tr>
<tr>
<td>dll\</td>
<td>The database scripts and MS-DOS batch files used for creating the database elements</td>
</tr>
<tr>
<td>IBM WebSphere Test</td>
<td>All the resources and configuration files used by this application</td>
</tr>
<tr>
<td>Environment\</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
This command is also contained in the xmlapp1.ddl script file.

4. You must now bind DB2 XML Extender and CLI to this database by issuing the following commands in the Command Window:

   db2 connect to L301
   cd /d %DB2DXXPATH%nd
   db2 bind @dxxbind.lst
   cd /d %DB2PATH%nd
   db2 bind @db2cli.lst
   db2 terminate

5. The L301 database must now be enabled for XML Extender with the dbxadm administration command. The command to enable the database is:

   dbxadm enable_db L301

   This command is also contained in the EnableDatabase.bat batch file.

6. The SQL definition statements to create the tables in the database L301 are included in the xmlapp2.ddl script file located in the ddl directory. The statements for populating the tables are in the data.ddl script file. You should run these two scripts by executing the following commands in the DB2 Command Window in this order:

   db2 -f xmlapp2.ddl -t
   db2 -f data.ddl -t

7. The last step is to enable the XML column orderdoc in the table ORDERXML using the orderCol.dad DAD file. The command for enabling this column is:

   dbxadm enable_column L301 orderxml orderdoc orderCol.dad -r order_no

   where orderCol.dad must be fully qualified with the path name if it is not in the current directory.

---

A.3 Configuring VisualAge for Java 3.02 Enterprise Edition

The application we are running in VisualAge for Java does exactly the same as our XMLapp application in WebSphere Application Server. But as the WebSphere Test Environment of VisualAge does not support exactly all the possibilities of WebSphere, we had to make a few changes to the XMLapp application of WebSphere. Or actually, it was rather the opposite: when deploying the application developed in VisualAge, we could benefit from all the features of WebSphere Application Server, and we made a few changes to the Visual Age for Java code.
The differences between the VAJ and the WAS versions are these:

- In VAJ, we are not able to define a MIME filter type to automatically call a servlet that transforms it. Therefore we must manually call the servlet transforming the XML into HTML rather than letting it be called automatically. This means than the URL used to start the application and used within the application are different as we must refer directly to the transforming servlet rather than simply outputting the XML document. To improve the performance, the servlet DefaultApplyXSL (DAXSL) used in WebSphere Application Server is replaced by the servlet XSLServlet doing the same job but accepting both parameters and attributes to get the information about the location of the XML document to be parsed (xmlURL).

This means that any servlet can call the XSLServlet servlet immediately with the `getServletContext().getRequestDispatcher().forward()` method passing the xmlURL argument as an attribute rather than redirecting this request to the XSL servlet and passing the xmlURL argument as a parameter (when the redirect() method is called, the servlet returns to the browser and indicates him to call the new URL).

- As a result of the restriction explained above, it is also impossible to use JSP files with an XML output since this XML would not get transformed into HTML. We must replace these JSP files (2 files in the application: search.jsp and orderbasket.jsp) by their equivalent servlets (SearchServlet and OrderBasketServlet) which create the same XML as the JSP files, but transforms it immediately into HTML with XSLT.

Apart from these two major differences regarding the application itself, it is also important to emphasize that for VisualAge for Java 3.02 Enterprise Edition to support JDBC datasources (connection pooling) and multiple Web applications, some configurations steps are required. These steps are explained in the next sections after the creation of a new project.

### A.3.1 Creation of a new project

Your first step is to create a new project and to import the jar file SG246130_VAJ into it. From now on, we call this project SG246130_VAJ, but you can choose any name.

Make sure your workspace classpath contains the db2java.zip file containing the DB2 Driver. To check that, go to the Workbench and click **Window > Options... > Resources.** Add the fully qualified db2java.zip file in the Workspace class path field if not yet included.
A.3.2 Defining a datasource

This configuration step is explained in detail in the IBM Web site at the following URL:


You can read the information there and download the required and optional files. We summarize the needed steps below, and the required files are provided in the downloadable zip file SG246130_VAJ.

We suppose that you are using the initial configuration of the IBM WebSphere Test Environment in VisualAge for Java. Follow these steps to define your datasource to access the database L301:

1. Import the DataSourceFactory.jar file into the IBM WebSphere Test Environment project.
2. Copy the CreateDataSource.html file provided in the zip file directory, IBM WebSphere Test Environment\hosts\default_host\default_app\web to that same directory on your computer.
   You can find this IBM WebSphere Test Environment directory on your computer in C:\IBM\Java\Ide\project_resources\ directory, where C:\IBM\Java is the installation directory of VisualAge for Java.
3. Start the Persistent Name Server:
   a. In the Workbench, select the EJB tab.
   b. From the EJB menu, select Open To => Server Configuration.
   c. In the Server Configuration browser, highlight the Persistent Name Server.
   d. Right-click the Persistent Name Server, and select Properties.
   e. In the Properties box, ensure that the Data Source field contains a valid DB2 URL (to an existing database, L301 or another one). Make sure the Connection Type field contains the value COM.ibm.db2.jdbc.app.DB2Driver. Press OK. At this point, the PersistentNameServer creates one schema EJSADMIN and three tables in the database indicated by the DB2 URL where it stores the information regarding JNDI.
   f. From the Servers menu, select Start Server.
   g. Look in the Console window for the output. Wait until you see the message "Server open for business."
4. Start the WebSphere Test Environment:
   a. From the Workspace menu, select Tools => Launch WebSphere Test Environment.
   b. Look in the Console window for the output. Wait until the output stops (usually 10 to 15 seconds).

5. In your Web browser, go to:
   http://localhost:8080/CreateDataSource.html

6. Fill in the name, class, URL, and type fields.
   a. For the name, enter L301.
   b. For the class name, enter COM.ibm.db2.jdbc.app.DB2Driver.
   c. For the URL field, enter jdbc:db2:.
   d. Leave JDBC for the type field.

7. Press Submit. Wait until you get the confirmation that the datasource has been created successfully.

To get more information concerning datasources in VisualAge for Java, please refer to the URL indicated at the start of this section.

A.3.3 Configuring multiple Web applications

This configuration step is explained in detail on the IBM Web site on the following URL:

It can also be found in the redbook Servlet and JSP Programming, SG24-5755-00.

You can read the information in these sources, but we summarize the needed steps below.

You have two ways to start the WebSphere Test Environment:

- The SERunner class (also started by clicking Workspace > Tools > Launch WebSphere Test Environment)
- The ServletEngine class
Figure 98 shows both classes in the VisualAge for Java Workbench. The default class is SERunner, but it does not give us the possibility of configuring more than one Web application. By using ServletEngine, we can define more than one.

The following steps describe how to configure a new Web application called xmlapp. We assume that you are using the original configuration of the IBM WebSphere Test Environment.

1. Create a backup copy of the default.servlet_engine (for example by renaming it default.servlet_engine.bak) configuration file located in the directory C:\IBMVJava\Ide\project_resources\IBM WebSphere Test Environment\ directory, where C:\IBMVJava is the installation directory of VisualAge for Java. You must restore this file if you want to use again
SERunner and the original configuration of the IBM WebSphere Test Environment.

2. Copy the IBM WebSphere Test Environment directory provided in the zip file on the existing IBM WebSphere Test Environment directory. Copying this directory replaces your default.servlet_engine file and adds the new subdirectories necessary for the xmlapp Web application. At the same time, it also adds all the XML, XSL and JSP files necessary for the application.

The main difference in the new default.servlet_engine file is that a new websphere-webgroup element has been added for the xmlapp Web application.

By copying this directory, you also added the file xmlapp.webapp in the subdirectory hosts/default_host/webapp/servlets. This file contains the configuration of all the named servlets used in the xmlapp web application and also set the JSP1.0 enabler.

3. In the VisualAge for Java Workbench, right-click the ServletEngine class and select Properties.

- On the Program folder, add the following string in the command line arguments field:

```
-serverRoot "c:\IBM\Java\Ide\project_resources\IBM WebSphere Test Environment"
```

In this command, `c:\IBM\Java\Ide\project_resources\IBM WebSphere Test Environment` is your IBM WebSphere Test Environment directory.

In the properties field, add:

```
ivj.version=3.02
```

- On the Class Path folder, add the name of your SG246130_VAJ project in the project path field. In the extra directories path, add all the directories mentioned in the SERunner class (the easiest way is to cut and paste it).

4. The configuration is now complete. The last step is to start the ServletEngine by selecting the ServletEngine class and clicking Run > Run Main.

### A.4 Completing the configuration of the xmlapp application

All the servlets have been loaded when importing the application in the new project. They have been defined when copying the xmlapp.webapp file (together with the IBM WebSphere Test Environment directory).
All the JSP, XML and XSL files have been copied with the IBM WebSphere Test Environment directory during the configuration of the xmlapp Web application.

The last step that could be required is to change the userid and password to access the database L301, in case you have not chosen xmlguru and xmlguru. If you need to change them, you must change them in the JSP files and in the itso.xmlapp.db2const.Constant class in your SG246130_VAJ project.

A.5 Running the application

To start using the application, make sure you first started the Persistent Name Server, and after this server has been started, the Servlet Engine.

You can access the customer side with your browser on the following URL:
http://localhost:8080/xmlapp/servlet/XSLServlet?xmlURL=/xml/MainMenu.xml

For the supplier side, go to the following URL:
http://localhost:8080/xmlapp/jsp/SupMainMenu.jsp

You should first start by the customer side, as the supplier side works with the orders submitted by the customers.

A.6 Accessing your IBM WebSphere Test Environment from another PC

By using localhost, we can only access the IBM WebSphere Test Environment from the same machine where VisualAge is running. In order to more closely match the real-life situation, it is possible to configure the IBM WebSphere Test Environment so that you can access it with a browser from another machine.

You must first add one element in the default.servlet_engine configuration file. To edit, the file make sure the ServletEngine has been stopped. Open the file and add the following:
<hostname-binding hostname="HOSTNAME:8080" servlethost="default_host"/>

In this element, HOSTNAME is the name of your machine. To find this name, you can issue the command HOSTNAME on the command prompt. This line must be added at the end of the file just before the end tag of the websphere-servlet-engine element, as shown below:
<hostname-binding hostname="localhost:8080" servlethost="default_host"/>
You must also change the Constant class in your project. Modify the PAGE_PREFIX String to replace localhost by your new HOSTNAME.

You are now able to go to the customer side with your browser at the following URL:

http://HOSTNAME:8080/xmlapp/servlet/XSLServlet?xmlURL=/xml/MainMenu.xml

For the supplier side, go to the following URL:

http://HOSTNAME:8080/xmlapp/jsp/SupMainMenu.jsp

In the two previous URLs, HOSTNAME is the name your machine.
Appendix B. XML extender diagnostic trace

The XML extender provides a trace facility by logging a variety of information to a file on the server. It should be used only when it is absolutely necessary, because it has a significant overhead of writing to a file.

The trace facility is invoked using the `dxxtrc` command. It should be invoked with the `on` option and a directory that will contain the trace file. The files are of the form `dxx<traceID>.log` where `traceID` is of the name of the instance. The name of the instance is the value of the environment variable `DB2INSTANCE`.

B.1 Start tracing:

Syntax:

```
dxxtrc on trace_directory
```

*trace_directory* is the name of the directory where the file `dxx<traceID>.log` is to be placed.

Example:

```
dxxtrc on d:\log
```

B.2 Stop tracing

Applying the `off` option on `dxxtrc` will turn the trace facility off. It does not remove the trace file. Before stopping the trace, run `db2stop` to clean up some locks on the log file.

Syntax:

```
dxxtrc off
```

Examples:

```
db2stop

dxxtrc off

db2start
```
A sample of the trace log file is shown below:

*** Trace turned on Wed Aug 02 10:06:44 2000

dxxICUSetDefaultName:Entered.
dxxICUSetDefaultName:Exit, errCode=0.dxxInitializeParser:Exit, return
rc=0
dxxInit:dxxInitializeParser rc = 0
DXX_checkLocale:Entered.
DXX_checkLocale:dxxPtr->icu->osLocale is 'en_US'
DXX_checkLocale:db2DbcsCodePg == 0 implies that db2CodePage must
be in codepage_tab.
Thus there is no need for looking up Db2CodepageTab.
DXX_checkLocale:Entered.
DXX_checkLocale:dxxPtr->icu->osLocale is 'en_US'
DXX_checkLocale:db2DbcsCodePg == 0 implies that db2CodePage must
be in codepage_tab.
Thus there is no need for looking up Db2CodepageTab.
DXX_checkLocale:Entered.
DXX_checkLocale:dxxPtr->icu->osLocale is 'en_US'
DXX_checkLocale:db2DbcsCodePg == 0 implies that db2CodePage must
be in codepage_tab.
Thus there is no need for looking up Db2CodepageTab.
DXX_checkLocale:Entered.
DXX_checkLocale:dxxPtr->icu->osLocale is 'en_US'
DXX_checkLocale:db2DbcsCodePg == 0 implies that db2CodePage must
be in codepage_tab.
Thus there is no need for looking up Db2CodepageTab.
DXX_checkLocale:Exit, errCode = 0

dxxEnableDB_STP:codepage is 1252 and isDbUTF8 is 0
dxxEnableDB_STP:entered. dbname:test
dxxEnableDB_STP:exit. error code:0; message:DXXA006I The database test was enabled successfully.

DXXA006I The database test was enabled successfully.
Appendix C. Generic DB2 XML Extender servlet

The goal of this servlet is to provide the XML Extender's functionality to extract XML data from DB2 over HTTP. The application reads all the parameters provided and tries to establish a database connection. In case of a successful connection the servlet calls the dxxGenXML stored procedure and returns its XML output. The generic servlet allows you to compose an XML document based on a DAD file (Xcollection) and displays it using the XSL stylesheet you want. If no stylesheet is specified and the default one is not found, or if the specified stylesheet is not found, you receive the XML document as a file to download. If you call the servlet without parameters, an HTML welcome page will be displayed (Figure 99) where you can enter the servlet parameters.

![DB2 XML Extender Generic Servlet](image)

*Figure 99. Screen shot of welcome page*
C.1 Overview

You can use this sample as a template which can easily be modified or expanded in its functionality. One might think of using decomposition instead of composition. Generally, servlets are used to generate HTML, which is then sent to a client browser over HTTP. However, servlets can generate any kind of output stream, including XML. For example, the XML output from this servlet could be consumed by another business application, which might be another servlet, an applet, Java programs, or other servers. Any request parameter entered overrides the corresponding init parameter (if it exists; see Table 20).

Table 20. Parameters for dxxServlet

<table>
<thead>
<tr>
<th>Init parameters</th>
<th>Request parameters</th>
<th>Default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>userID</td>
<td>UserID</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>password</td>
<td></td>
</tr>
<tr>
<td>providerURL</td>
<td>providerURL</td>
<td>iiop:///</td>
</tr>
<tr>
<td>initialContextFactory</td>
<td>initialContextFactory</td>
<td>com.ibm.ejs.ns.jndi.CNInitialContextFactory</td>
</tr>
<tr>
<td>datasourceJNDI</td>
<td>datasourceJNDI</td>
<td></td>
</tr>
<tr>
<td>dbURL</td>
<td>dbURL</td>
<td>jdbc:db2:</td>
</tr>
<tr>
<td>dbDriver</td>
<td>dbDriver</td>
<td>COM.ibm.db2.jdbc.app.DB2Driver</td>
</tr>
<tr>
<td>dbName</td>
<td>dbName</td>
<td></td>
</tr>
<tr>
<td>dadFileURL</td>
<td></td>
<td><a href="http://localhost/default.xsl">http://localhost/default.xsl</a></td>
</tr>
</tbody>
</table>

You can connect to the database in three ways (in order of priority):

- A datasource name (for which you can specify another providerURL and initialContextFactory).
- A dbURL and a dbDriver.
- A dbName only. This last option causes the default dbDriver to be used and the dbURL to be built based on the default value and the entered dbName.

To run the sample, you have to enter your DB2 userID and password, one of the three database connections mentioned above, and the dadFileURL.
C.2 Installing the dxxServlet in WAS

In this section we describe the necessary configuration and installation steps that have to be done in order to run the dxxServlet. It is assumed that the basic infrastructure — WebSphere, DB2, and IBM HTTP Server — are installed and functional.

C.2.1 Installing the database

The dxxServlet can use any XML Extender enabled database, but in this tutorial, we will refer to the SALES_DB database from the scenario in Chapter 2 of the “XML Extender Administration and Programming” manual. See the Web page:


There is a set of scripts provided 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.

Execute the following commands to prepare your database:

getstart_db.cmd

This creates the database SALES_DB and populates four tables.

getstart_prep.cmd

This binds the database with the XML Extender stored procedures and the DB2 CLI.

Now you have to configure WebSphere to deploy the dxxServlet, as described in the following sections.

Note: After running the dxxServlet, you can use the following script for cleanup:

getstart_clean.cmd

This cleans up the tutorial environment.

C.2.2 Configuring WebSphere Application Server 3.02

WebSphere Application Server uses the notion of a Web Application to separate applications with different requirements from each other. For this servlet we will use the default application of WebSphere, because we do not need the MIME-filter for text/xml, as we do for the other servlets of XMLapp.
The jar file jsp10.jar is required for our application. Therefore, we refer to it in the command line arguments parameter of the Default Server. Add the following argument:

-classpath c:\WebSphere\AppServer\lib\jsp10.jar

We then have to define our servlet in WebSphere. In the Topology view, select the application default_app, and in the pop-up menu select Create->Servlet. Fill in the values as shown in Figure 100.

![Create Servlet](image)

**Figure 100. Creation of dxxServlet**

### C.2.3 Installing the Java classes

You must copy the directory itso with its content into the directory C:\WebSphere\AppServer\hosts\default_host\default_app\servlets, where C:\WebSphere\AppServer is the installation directory of WebSphere Application Server.
C.2.4 Installing the DAD and XSL files

We now have to copy the DAD and XSL files into the appropriate directories so that the IBM HTTP Server can find the files it requires for the servlet:

1. default.xsl: Copy the file from C:\WebSphere\AppServer\web\xml\xsl\default\ in the directory C:\Program Files\IBM HTTP Server\htdocs, where C:\Program Files\IBM HTTP Server is the installation directory of the IBM HTTP Server.

2. getstart_xcollection.dad: Copy the file C:\dxx\samples\dad\getstart_xcollection.dad in the directory C:\Program Files\IBM HTTP Server\htdocs.

C.2.5 Running the application

After starting the default server from WebSphere’s Administrative Console, you can access the application by typing in the following URL in a Web browser:

http://<your_host>/servlet/dxxServlet

To generate your XML document, enter your DB2 userID and password, jdbc:db2:sales_db in the dbURL entry field and enter http://localhost/getstart_xcollection.dad in the dadFileURL entry field. Click the PROCESS button to start the XML composition. The result will look as shown in Figure 101.
Figure 101. Screen shot of resulting page
Table 21 shows information on the countries and code pages supported by DB2 UDB. Since DB2 XML Extender is now available on AIX, Sun Solaris, Linux, and Windows NT, this table shows limited information on these operating systems. You can see the complete tables in *DB2 Universal Database Administration Guide, SBOF-8934*.

The following is an explanation of each column in the table:

<table>
<thead>
<tr>
<th>Code Page</th>
<th>Shows the IBM-defined code page as mapped from the operating system code set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Shows whether a code page is single-byte (“S”) or multi-byte (“D”). The “-n” is a number used to create a letter-number combination. Matching combinations show where connection and conversion is allowed by DB2. For example, all “S-1” groups can work together.</td>
</tr>
<tr>
<td>Code Set</td>
<td>Shows the code set associated with the supported language. The code set is mapped to the DB2 code page.</td>
</tr>
<tr>
<td>Tr.</td>
<td>Shows the two-letter territory identifier.</td>
</tr>
<tr>
<td>Country Code</td>
<td>Shows the country code that is used by the database manager internally to provide country-specific support.</td>
</tr>
<tr>
<td>Locale</td>
<td>Shows the locale values supported by the database manager.</td>
</tr>
<tr>
<td>OS</td>
<td>Shows the operating system that supports the languages and code sets.</td>
</tr>
<tr>
<td>Country Name</td>
<td>Shows the name of the country or countries.</td>
</tr>
<tr>
<td>XML</td>
<td>Shows the code page supported by DB2 XML Extender with “Y” .</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>1046</td>
<td>S-6</td>
</tr>
<tr>
<td>1089</td>
<td>S-6</td>
</tr>
<tr>
<td>1256</td>
<td>S-6</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>915</td>
<td>S-5</td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>915</td>
<td>S-5</td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>1383</td>
<td>D-4</td>
</tr>
<tr>
<td>1386</td>
<td>D-4</td>
</tr>
<tr>
<td>1383</td>
<td>D-4</td>
</tr>
<tr>
<td>1381</td>
<td>D-4</td>
</tr>
<tr>
<td>1386</td>
<td>D-4</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>922</td>
<td>S-10</td>
</tr>
<tr>
<td>922</td>
<td>S-10</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>915</td>
<td>S-5</td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
</tr>
<tr>
<td>813</td>
<td>S-7</td>
</tr>
<tr>
<td>813</td>
<td>S-7</td>
</tr>
<tr>
<td>737</td>
<td>S-7</td>
</tr>
<tr>
<td>1253</td>
<td>S-7</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
</tr>
</tbody>
</table>

Integrating XML with DB2 XML Extender and DB2 Text Extender
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>IS</td>
<td>354</td>
<td>-</td>
<td>WIN</td>
<td>Iceland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>IE</td>
<td>353</td>
<td>en_IE</td>
<td>AIX</td>
<td>Ireland</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>IE</td>
<td>353</td>
<td>En_IE</td>
<td>AIX</td>
<td>Ireland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>IE</td>
<td>353</td>
<td>en_IE</td>
<td>Sun</td>
<td>Ireland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>IE</td>
<td>353</td>
<td>en_IE</td>
<td>Linux</td>
<td>Ireland</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>IE</td>
<td>353</td>
<td>-</td>
<td>WIN</td>
<td>Ireland</td>
<td>Y</td>
</tr>
<tr>
<td>916</td>
<td>S-8</td>
<td>ISO8859-8</td>
<td>IL</td>
<td>972</td>
<td>iw_IL</td>
<td>AIX</td>
<td>Israel</td>
<td>Y</td>
</tr>
<tr>
<td>916</td>
<td>S-8</td>
<td>ISO-8859-8</td>
<td>IL</td>
<td>972</td>
<td>iw_IL</td>
<td>Linux</td>
<td>Israel</td>
<td>Y</td>
</tr>
<tr>
<td>1255</td>
<td>S-8</td>
<td>1255</td>
<td>IL</td>
<td>972</td>
<td>-</td>
<td>WIN</td>
<td>Israel</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>IT</td>
<td>39</td>
<td>it_IT</td>
<td>AIX</td>
<td>Italy</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>IT</td>
<td>39</td>
<td>It_IT</td>
<td>AIX</td>
<td>Italy</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>IT</td>
<td>39</td>
<td>it</td>
<td>Sun</td>
<td>Italy</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>IT</td>
<td>39</td>
<td>it_IT</td>
<td>Linux</td>
<td>Italy</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>IT</td>
<td>39</td>
<td>-</td>
<td>WIN</td>
<td>Italy</td>
<td>Y</td>
</tr>
<tr>
<td>954</td>
<td>D-1</td>
<td>IBM-eucJP</td>
<td>JP</td>
<td>81</td>
<td>ja_JP</td>
<td>AIX</td>
<td>Japan</td>
<td></td>
</tr>
<tr>
<td>932</td>
<td>D-1</td>
<td>IBM-932</td>
<td>JP</td>
<td>81</td>
<td>Ja_JP</td>
<td>AIX</td>
<td>Japan</td>
<td>Y</td>
</tr>
<tr>
<td>954</td>
<td>D-1</td>
<td>eucJP</td>
<td>JP</td>
<td>81</td>
<td>ja</td>
<td>Sun</td>
<td>Japan</td>
<td></td>
</tr>
<tr>
<td>954</td>
<td>D-1</td>
<td>EUC-JP</td>
<td>JP</td>
<td>81</td>
<td>ja_JP</td>
<td>Linux</td>
<td>Japan</td>
<td></td>
</tr>
<tr>
<td>943</td>
<td>D-1</td>
<td>IBM-943</td>
<td>JP</td>
<td>81</td>
<td>-</td>
<td>WIN</td>
<td>Japan</td>
<td>Y</td>
</tr>
<tr>
<td>970</td>
<td>D-3</td>
<td>IBM-eucKR</td>
<td>KR</td>
<td>82</td>
<td>ko_KR</td>
<td>AIX</td>
<td>Korea,South</td>
<td>Y</td>
</tr>
<tr>
<td>970</td>
<td>D-3</td>
<td>5601</td>
<td>KR</td>
<td>82</td>
<td>ko</td>
<td>Sun</td>
<td>Korea,South</td>
<td>Y</td>
</tr>
<tr>
<td>1363</td>
<td>D-3</td>
<td>1363</td>
<td>KR</td>
<td>82</td>
<td>-</td>
<td>WIN</td>
<td>Korea,South</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>Lat</td>
<td>3</td>
<td>-</td>
<td>AIX</td>
<td>Latin America</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>Lat</td>
<td>3</td>
<td>-</td>
<td>AIX</td>
<td>Latin America</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>Lat</td>
<td>3</td>
<td>-</td>
<td>Sun</td>
<td>Latin America</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>Lat</td>
<td>3</td>
<td>-</td>
<td>Linux</td>
<td>Latin America</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>Lat</td>
<td>3</td>
<td>-</td>
<td>WIN</td>
<td>Latin America</td>
<td>Y</td>
</tr>
<tr>
<td>921</td>
<td>S-10</td>
<td>IBM-921</td>
<td>LV</td>
<td>371</td>
<td>Lv_LV</td>
<td>AIX</td>
<td>Latvia</td>
<td>Y</td>
</tr>
</tbody>
</table>

Appendix D. Country codes / code pages supported by DB2 UDB

293
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>921</td>
<td>S-10</td>
<td>IBM-921</td>
<td>LV</td>
<td>371</td>
<td>-</td>
<td>WIN</td>
<td>Latvia</td>
<td>Y</td>
</tr>
<tr>
<td>921</td>
<td>S-10</td>
<td>IBM-921</td>
<td>LT</td>
<td>370</td>
<td>LT_LT</td>
<td>AIX</td>
<td>Lithuania</td>
<td>Y</td>
</tr>
<tr>
<td>921</td>
<td>S-10</td>
<td>IBM-921</td>
<td>LV</td>
<td>370</td>
<td>-</td>
<td>WIN</td>
<td>Lithuania</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>NL</td>
<td>31</td>
<td>nl_NL</td>
<td>AIX</td>
<td>Netherlands</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>NL</td>
<td>31</td>
<td>NI_NL</td>
<td>AIX</td>
<td>Netherlands</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>NL</td>
<td>31</td>
<td>nl_NL</td>
<td>AIX</td>
<td>Netherlands</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>NL</td>
<td>31</td>
<td>-</td>
<td>WIN</td>
<td>Netherlands</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>NZ</td>
<td>64</td>
<td>En_NZ</td>
<td>AIX</td>
<td>New Zealand</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>NZ</td>
<td>64</td>
<td>en_NZ</td>
<td>AIX</td>
<td>New Zealand</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>NZ</td>
<td>64</td>
<td>en_NZ</td>
<td>AIX</td>
<td>New Zealand</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>NZ</td>
<td>64</td>
<td>-</td>
<td>WIN</td>
<td>New Zealand</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>NO</td>
<td>47</td>
<td>no_NO</td>
<td>AIX</td>
<td>Norway</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>NO</td>
<td>47</td>
<td>No_NO</td>
<td>AIX</td>
<td>Norway</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>NO</td>
<td>47</td>
<td>no</td>
<td>Sun</td>
<td>Norway</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>NO</td>
<td>47</td>
<td>no_NO</td>
<td>Linux</td>
<td>Norway</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>NO</td>
<td>47</td>
<td>-</td>
<td>WIN</td>
<td>Norway</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO8859-2</td>
<td>PL</td>
<td>48</td>
<td>pl_PL</td>
<td>AIX</td>
<td>Poland</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO-8859-2</td>
<td>PL</td>
<td>48</td>
<td>pl_PL</td>
<td>Linux</td>
<td>Poland</td>
<td>Y</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
<td>1250</td>
<td>PL</td>
<td>48</td>
<td>-</td>
<td>WIN</td>
<td>Poland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>PT</td>
<td>351</td>
<td>pt_PT</td>
<td>AIX</td>
<td>Portugal</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>PT</td>
<td>351</td>
<td>Pt_PT</td>
<td>AIX</td>
<td>Portugal</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>PT</td>
<td>351</td>
<td>pt</td>
<td>Sun</td>
<td>Portugal</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>PT</td>
<td>351</td>
<td>pt_PT</td>
<td>Linux</td>
<td>Portugal</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>PT</td>
<td>351</td>
<td>-</td>
<td>WIN</td>
<td>Portugal</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO8859-2</td>
<td>RO</td>
<td>40</td>
<td>ro_RO</td>
<td>AIX</td>
<td>Romania</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO-8859-2</td>
<td>RO</td>
<td>40</td>
<td>ro_RO</td>
<td>Linux</td>
<td>Romania</td>
<td>Y</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
<td>1250</td>
<td>RO</td>
<td>40</td>
<td>-</td>
<td>WIN</td>
<td>Romania</td>
<td>Y</td>
</tr>
<tr>
<td>915</td>
<td>S-5</td>
<td>ISO8859-5</td>
<td>RU</td>
<td>7</td>
<td>ru_RU</td>
<td>AIX</td>
<td>Russia</td>
<td>Y</td>
</tr>
</tbody>
</table>

294 Integrating XML with DB2 XML Extender and DB2 Text Extender
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>915</td>
<td>S-5</td>
<td>ISO-8859-5</td>
<td>RU</td>
<td>7</td>
<td>ru_RU</td>
<td>Linux</td>
<td>Russia</td>
<td>Y</td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
<td>1251</td>
<td>RU</td>
<td>7</td>
<td>-</td>
<td>WIN</td>
<td>Russia</td>
<td>Y</td>
</tr>
<tr>
<td>915</td>
<td>S-5</td>
<td>ISO8859-5</td>
<td>SP</td>
<td>381</td>
<td>sr_SP</td>
<td>AIX</td>
<td>Serbia/Montenegro</td>
<td>Y</td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
<td>1251</td>
<td>SP</td>
<td>381</td>
<td>-</td>
<td>WIN</td>
<td>Serbia/Montenegro</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO8859-2</td>
<td>SK</td>
<td>422</td>
<td>sk_SK</td>
<td>AIX</td>
<td>Croatia</td>
<td>Y</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
<td>1250</td>
<td>SK</td>
<td>422</td>
<td>-</td>
<td>WIN</td>
<td>Croatia</td>
<td>Y</td>
</tr>
<tr>
<td>912</td>
<td>S-2</td>
<td>ISO8859-2</td>
<td>SI</td>
<td>386</td>
<td>sl_SI</td>
<td>AIX</td>
<td>Slovenia</td>
<td>Y</td>
</tr>
<tr>
<td>1250</td>
<td>S-2</td>
<td>1250</td>
<td>SI</td>
<td>386</td>
<td>-</td>
<td>WIN</td>
<td>Slovenia</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>ZA</td>
<td>27</td>
<td>en_ZA</td>
<td>AIX</td>
<td>South Africa</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>ZA</td>
<td>27</td>
<td>En_ZA</td>
<td>AIX</td>
<td>South Africa</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>ZA</td>
<td>27</td>
<td>-</td>
<td>Sun</td>
<td>South Africa</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>ZA</td>
<td>27</td>
<td>-</td>
<td>WIN</td>
<td>South Africa</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>ES</td>
<td>34</td>
<td>es_ES</td>
<td>AIX</td>
<td>Spain</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>ES</td>
<td>34</td>
<td>Es_ES</td>
<td>AIX</td>
<td>Spain</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>ES</td>
<td>34</td>
<td>es</td>
<td>Sun</td>
<td>Spain</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>ES</td>
<td>34</td>
<td>es_ES</td>
<td>Linux</td>
<td>Spain</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>ES</td>
<td>34</td>
<td>-</td>
<td>WIN</td>
<td>Spain</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>SE</td>
<td>46</td>
<td>sv_SE</td>
<td>AIX</td>
<td>Sweden</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>SE</td>
<td>46</td>
<td>Sv_SE</td>
<td>AIX</td>
<td>Sweden</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>SE</td>
<td>46</td>
<td>sv</td>
<td>Sun</td>
<td>Sweden</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>SE</td>
<td>46</td>
<td>sv_SE</td>
<td>Linux</td>
<td>Sweden</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>SE</td>
<td>46</td>
<td>-</td>
<td>WIN</td>
<td>Sweden</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>CH</td>
<td>41</td>
<td>de_CH</td>
<td>AIX</td>
<td>Switzerland</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>CH</td>
<td>41</td>
<td>De_CH</td>
<td>AIX</td>
<td>Switzerland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>CH</td>
<td>41</td>
<td>de_CH</td>
<td>Sun</td>
<td>Switzerland</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>CH</td>
<td>41</td>
<td>de_CH</td>
<td>Linux</td>
<td>Switzerland</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>CH</td>
<td>41</td>
<td>-</td>
<td>WIN</td>
<td>Switzerland</td>
<td>Y</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>----------</td>
<td>-----</td>
<td>--------------</td>
<td>--------</td>
<td>-----</td>
<td>--------------</td>
<td>-----</td>
</tr>
<tr>
<td>950</td>
<td>D-2</td>
<td>big5</td>
<td>TW</td>
<td>88</td>
<td>Zh_TW</td>
<td>AIX</td>
<td>Taiwan</td>
<td>Y</td>
</tr>
<tr>
<td>964</td>
<td>D-2</td>
<td>IBM-eucTW</td>
<td>TW</td>
<td>88</td>
<td>zh_TW</td>
<td>AIX</td>
<td>Taiwan</td>
<td>Y</td>
</tr>
<tr>
<td>950</td>
<td>D-2</td>
<td>big5</td>
<td>TW</td>
<td>88</td>
<td>big5</td>
<td>Sun</td>
<td>Taiwan</td>
<td>Y</td>
</tr>
<tr>
<td>964</td>
<td>D-2</td>
<td>cns11643</td>
<td>TW</td>
<td>88</td>
<td>zh_TW</td>
<td>Sun</td>
<td>Taiwan</td>
<td>Y</td>
</tr>
<tr>
<td>950</td>
<td>D-2</td>
<td>big5</td>
<td>TW</td>
<td>88</td>
<td>-</td>
<td>WIN</td>
<td>Taiwan</td>
<td>Y</td>
</tr>
<tr>
<td>874</td>
<td>S-20</td>
<td>TIS620-1</td>
<td>TH</td>
<td>66</td>
<td>Th_TH</td>
<td>AIX</td>
<td>Thailand</td>
<td>Y</td>
</tr>
<tr>
<td>874</td>
<td>S-20</td>
<td>TIS620-1</td>
<td>TH</td>
<td>66</td>
<td>-</td>
<td>WIN</td>
<td>Thailand</td>
<td>Y</td>
</tr>
<tr>
<td>920</td>
<td>S-9</td>
<td>ISO8859-9</td>
<td>TR</td>
<td>90</td>
<td>tr_TR</td>
<td>AIX</td>
<td>Turkey</td>
<td>Y</td>
</tr>
<tr>
<td>920</td>
<td>S-9</td>
<td>ISO-8859-9</td>
<td>TR</td>
<td>90</td>
<td>tr_TR</td>
<td>Linux</td>
<td>Turkey</td>
<td>Y</td>
</tr>
<tr>
<td>1254</td>
<td>S-9</td>
<td>1254</td>
<td>TR</td>
<td>90</td>
<td>-</td>
<td>WIN</td>
<td>Turkey</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>GB</td>
<td>44</td>
<td>en_GB</td>
<td>AIX</td>
<td>U.K.</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>GB</td>
<td>44</td>
<td>En_GB</td>
<td>AIX</td>
<td>U.K.</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>GB</td>
<td>44</td>
<td>en_UK</td>
<td>Sun</td>
<td>U.K.</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>GB</td>
<td>44</td>
<td>en_GB</td>
<td>Linux</td>
<td>U.K.</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>GB</td>
<td>44</td>
<td>-</td>
<td>WIN</td>
<td>U.K.</td>
<td>Y</td>
</tr>
<tr>
<td>1124</td>
<td>S-5</td>
<td>IBM-1124</td>
<td>UA</td>
<td>380</td>
<td>uk_UA</td>
<td>AIX</td>
<td>Ukraine</td>
<td></td>
</tr>
<tr>
<td>1251</td>
<td>S-5</td>
<td>1251</td>
<td>UA</td>
<td>380</td>
<td>-</td>
<td>WIN</td>
<td>Ukraine</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>US</td>
<td>1</td>
<td>en_US</td>
<td>AIX</td>
<td>USA</td>
<td>Y</td>
</tr>
<tr>
<td>850</td>
<td>S-1</td>
<td>IBM-850</td>
<td>US</td>
<td>1</td>
<td>En_US</td>
<td>AIX</td>
<td>USA</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO8859-1</td>
<td>US</td>
<td>1</td>
<td>en_US</td>
<td>Sun</td>
<td>USA</td>
<td>Y</td>
</tr>
<tr>
<td>819</td>
<td>S-1</td>
<td>ISO-8859-1</td>
<td>US</td>
<td>1</td>
<td>en_US</td>
<td>Linux</td>
<td>USA</td>
<td>Y</td>
</tr>
<tr>
<td>1252</td>
<td>S-1</td>
<td>1252</td>
<td>US</td>
<td>1</td>
<td>-</td>
<td>WIN</td>
<td>USA</td>
<td>Y</td>
</tr>
<tr>
<td>1163</td>
<td>S-11</td>
<td>IBM-1163</td>
<td>VN</td>
<td>84</td>
<td>vi_VN</td>
<td>AIX</td>
<td>Vietnam</td>
<td></td>
</tr>
<tr>
<td>1258</td>
<td>S-11</td>
<td>1258</td>
<td>VN</td>
<td>84</td>
<td>-</td>
<td>WIN</td>
<td>Vietnam</td>
<td>Y</td>
</tr>
</tbody>
</table>
Appendix E. Using the additional material

This redbook also contains additional material on the Internet. See the appropriate section below for instructions on using or downloading each type of material.

E.1 Locating the additional material on the Internet

The CD-ROM, diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246130

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the Additional materials and open the directory that corresponds with the redbook form number.

E.2 Using the Web material

The additional Web material that accompanies this redbook includes the following (see Table 22):

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG246130.zip</td>
<td>Zipped Code</td>
</tr>
</tbody>
</table>

Table 22. Contents of SG246130.ZIP

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadMe.txt</td>
<td>Describes any updates to the book.</td>
</tr>
<tr>
<td>SG246130.html</td>
<td>Sample file with links to the applications. This file can be copied to your Web server document root.</td>
</tr>
<tr>
<td>\collections</td>
<td>Files for Chapter 8, “XMLapp: installing XML Columns version in WAS” on page 249</td>
</tr>
<tr>
<td>\columns</td>
<td>Files for Chapter 9, “XMLapp: installing XML Collections version in WAS” on page 263</td>
</tr>
<tr>
<td>\vaj</td>
<td>Files for Appendix A, “XMLapp: installing XML Columns version in VAJ” on page 269</td>
</tr>
</tbody>
</table>
E.2.1 How to use the Web material

Create a subdirectory (folder) on your workstation and copy the contents of the Web material into this folder. This folder should be on drive C: and called SG246130.
Appendix F. Special notices

This publication is intended to help Web and database developers to take advantage of XML technology and develop XML applications with DB2 XML Extender, IBM DB2 Text Extender, IBM VisualAge for Java, and IBM WebSphere Application Server. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM DB2 XML Extender, IBM DB2 Text Extender, IBM VisualAge for Java or IBM WebSphere Application Server. See the PUBLICATIONS section of the IBM Programming Announcement for the above products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

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, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the
customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

IBM ©

AIX
AS/400
AT
CT
Current
DB2
DB2 Extenders
DB2 Universal Database
IBM
Manage. Anything. Anywhere.
Netfinity
RS/6000
SP
SP1
System/390
VisualAge
WebSphere
Wizard
XT
400
Lotus
Notes
Tivoli
Manage. Anything. Anywhere.
TME
NetView
Cross-Site
Tivoli Ready
Tivoli Certified
The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.
Integrating XML with DB2 XML Extender and DB2 Text Extender
Appendix G. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

G.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 307.

- *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, SG24-5479
- *Servlet and JSP Programming with IBM WebSphere Studio andVisualAge for Java*, SG24-5755
- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*, SG24-5754
- *WebSphere Application Servers: Standard and Advanced Editions*, SG24-5460
- *Developing an e-business Application for the IBM WebSphere Application Server*, SG24-5423

G.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System/390 Redbooks Collection</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>IBM Networking Redbooks Collection</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>IBM Transaction Processing and Data Management Redbooks Collection</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>IBM Lotus Redbooks Collection</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>IBM AS/400 Redbooks Collection</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>IBM Netfinity Hardware and Software Redbooks Collection</td>
<td>SK2T-8046</td>
</tr>
<tr>
<td>IBM RS/6000 Redbooks Collection</td>
<td>SK2T-8043</td>
</tr>
<tr>
<td>IBM Application Development Redbooks Collection</td>
<td>SK2T-8037</td>
</tr>
<tr>
<td>IBM Enterprise Storage and Systems Management Solutions</td>
<td>SK3T-3694</td>
</tr>
</tbody>
</table>
G.3 Other resources

These publications are also relevant as further information sources:

- *IBM DB2 Universal Database XML Extender Administration and Programming*. (Note: This manual is supplied as a file, db2sx.pdf, which is shipped with the product)
- *IBM DB2 Universal Database Text Extender Administration and Programming*, SC26-9930
- *IBM DB2 Universal Database Administration Guide*, SBOF-8934
- *IBM DB2 Universal Database SQL Reference*, SBOF-8933
- *IBM DB2 Universal Database Application Development Guide*, SC09-2949
- *Character Data Representation Architecture Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

G.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- http://www.iso.ch
  ISO — International Organization for Standardization
- http://www.unicode.org
  Unicode Standard
- http://wcp.oclc.org
  OCLC Online Computer Library Center, Inc.
- http://www.w3.org/XML
  W3C — World Wide Web Consortium — XML
- http://www.schema.net
  SCHEMA.NET Portal
- http://www.oasis-open.org
  OASIS — Organization for the Advancement of Structured Information Standards
- http://www.w3.org/TR/xmlschema-2
  W3C — World Wide Web Consortium — XML Schemas
- http://www.w3.org/TR/xmlschema-1
  W3C — World Wide Web Consortium — XML Schemas
• http://www.jclark.com/dsssl
  Jim Clarks's — Document Style Semantics and Specification Language

• http://www.w3.org/Style/CSS
  W3C — World Wide Web Consortium — Cascading Style Sheets

• http://www.w3.org/TR/WD-xsl
  W3C — World Wide Web Consortium — XSL

• http://www.w3.org/TR/WD-xslt
  W3C — World Wide Web Consortium — XSLT

• http://www.ibm.com/developer/xml
  IBM — developerWorks — XML Zone

• http://www.xml.org/xmlorg_registry/index.shtml
  The XML Industry Portal

• http://www.megginson.com/SAX
  Megginson Technologies Ltd. — SAX

• http://www.w3.org/DOM
  W3C — World Wide Web Consortium — DOM

• http://www.w3.org/TR/xml-stylesheet
  W3C — World Wide Web Consortium — Associating Style Sheets with XML documents

  IBM — Creating DataSources in the VisualAge for Java WebSphere Test Environment

  IBM — Configuring the VisualAge for Java WebSphere Test Environment

• http://www.ibm.com/software/data/db2/extenders/xmlext/library.html
  IBM — DB2 XML Extender Documentation
How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- Redbooks Web Site ibm.com/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- E-mail Orders

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

  | In United States or Canada | pubscan@us.ibm.com |
  | Outside North America     | Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl |

- Telephone Orders

  | United States (toll free) | 1-800-879-2755 |
  | Canada (toll free)        | 1-800-IBM-4YOU  |
  | Outside North America    | Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl |

- Fax Orders

  | United States (toll free) | 1-800-445-9269      |
  | Canada                   | 1-403-267-4455      |
  | Outside North America    | Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl |

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

---

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.
IBM Redbooks fax order form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First name       Last name
Company
Address
City              Postal code   Country
Telephone number  Telefax number  VAT number

☐ Invoice to customer number
☐ Credit card number

Credit card expiration date  Card issued to  Signature

*We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.*
**Glossary**

**applet.** A Java program designed to run within a Web browser. Contrast with application.

**application.** In Java programming, a self-contained, stand-alone Java program that includes a main() method. Contrast with applet.

**application server.** A server program that allows the installation of application specific software components, in a manner so that they can be remotely invoked, usually by some for of remote object method call.

**Application Service Provider (ASP).** An ASP is an agent or broker that aggregates, facilitates and brokers IT services to deliver IT-enabled business solutions across a network via subscription-based pricing.

**attribute.** In XML, a name="value" pair that can be placed in the start tag of an element. The value must be quoted with single or double quotes.

**Cascading Style Sheet (CSS).** CSS defines a stylesheet language for HTML 4.0. CSS allows a Web page designer to separately specify style elements of a Web page, such as colors, fonts and font styles.

**case-sensitive.** Indicates whether an application, processor, or operating system distinguishes between upper and lower case. If it does, it is case-sensitive. XML tags are case-sensitive, but HTML tags are not.

**class.** An aggregate that defines properties, operations, and behavior for all instances of that aggregate.

**CLASSPATH.** In your deployment environment, the environment variable that specifies the directories in which to look for class and resource files.

**constructor.** A special class method that has the same name as the class and is used to construct and possibly initialize objects of its class type.

**content model.** In XML, the expression specifying what elements and data are allowed within an element.

**Document access definition (DAD).** An XML document where you specify how DB2 XML Extender should handle the XML documents.

**double-byte character set (DBCS).** A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Compare with single-byte character set.

**Document Object Model (DOM).** This allows the representation and manipulation of an XML document in memory as a programming object. DOM is defined by the World-Wide Web Consortium.

**DOM.** (see Document Object Model).

**DOM Tree.** A DOM Tree is an in-memory representation of an XML Document.

**Document Type Definition (DTD).** A DTD is a definition of which Elements and Attributes are acceptable in a specific XML file. The DTD therefore defines a subset of XML which may be used for a particular application.

**BNF.** Extended Backus-Naur Form. A formal set of production rules that comprise a grammar defining another language, such as XML.

**Electronic data interchange.** The automatic machine-to-machine transfer of trading documents (e.g., invoices and purchase orders) using electronic networks such as the internet. Originally conducted only through value-added networks, EDI is gradually moving to the Internet.

**element.** In XML, a start tag and its end tag, plus the content between the tags. An empty tag is also an element.

**empty declaration.** In XML, the DTD declaration for an empty tag. For example, if
<foo/> is an empty tag, the empty declaration looks like: <!ELEMENT foo EMPTY>.

empty tag. In XML, a start and end tag combined in one tag. The tag has a trailing slash, so an XML parser can immediately recognize it as an empty tag and not bother looking for a matching end tag. For example, if foo is an empty tag, it looks like <foo/>.

entity. In XML, an entity declaration provides the ability to have constants or replacement strings, which are expanded by a pre-processor. An entity declaration maps some token to a replacement string. Later the token can be prefixed with the & character and the replacement string is put in its place.

Enterprise JavaBeans (EJB). The Enterprise JavaBeans specification defines a way of building transactionally aware business objects in Java.

Hypertext Markup Language (HTML). The basic language that is used to build hypertext documents on the World Wide Web. It is used in basic, plain ASCII-text documents, but when those documents are interpreted (rendered) by a Web browser such as Netscape, the document can display formatted text, color, a variety of fonts, graphics images, special effects, hypertext jumps to other Internet locations, and information forms.

Hypertext Transfer Protocol (HTTP). The protocol for moving hypertext files across the Internet. Requires an HTTP client program on one end, and an HTTP server program on the other end.

instance. Synonym for object, a particular instantiation of a data type.

Java. A programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to your computer through the Internet and immediately run without fear of viruses or other harm to your computer or files. Using small Java programs (called applets), Web pages can include functions such as animation, calculators, and other fancy tricks. We can expect to see a huge variety of features added to the Web through Java, because you can write a Java program to do almost anything a regular computer program can do and then include that Java program in a Web page.

Java archive (JAR). A platform-independent file format that groups many files into one. JAR files are used for compression, reduced download time, and security. Because the JAR format is written in Java, JAR files are fully extensible.

JavaBeans. In JDK 1.1, the specification that defines the platform-neutral component model used to represent parts. Instances of JavaBeans (often called beans) may have methods, properties, and events.

Java Database Connectivity (JDBC). In JDK 1.1, the specification that defines an API that enables programs to access databases that comply with this standard.

Java Naming and Directory Interface (JNDI). The Java standard API for accessing directory services, such as LDAP, COS Naming, and others.

Java Server Page (JSP). Java Server Pages are Web pages that include dynamic tags which are executed on the server. JSPs are the presentation layer for Web-based applications built in Java.


method. A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

object. (1) A computer representation of something that a user can work with to perform a task. An object can appear as text or an icon. (2) A collection of data and methods that operate on that data, which together represent a logical entity in the system. In object-oriented programming, objects are grouped into classes that share common data definitions and methods. Each object in the class is said to be an instance of the class. (3) An instance of an object class consisting of attributes, a data structure, and operational methods. It can represent a person, place, thing, event, or concept. Each instance has the same properties, attributes, and methods as other instances of the object class, although it has unique values assigned to its attributes.

object class. A template for defining the attributes and methods of an object. An object class can
contain other object classes. An individual representation of an object class is called an object.

**overloading.** An object-oriented programming technique that allows redefinition of methods when the methods are used with class types.

**package.** A program element that contains related classes and interfaces.

**private.** In Java, an access modifier associated with a class member. It allows only the class itself to access the member.

**project.** In VisualAge for Java, the topmost kind of program element. A project contains Java packages.

**server.** A computer that provides services to multiple users or workstations in a network; for example, a file server, a print server, or a mail server

**Servlets.** Servlets are Java objects which execute on the server in response to a browser request. They can either generate HTML or XML directly, or call a JSP to produce the output.

**single-byte character set (SBCS).** A set of characters in which each character is represented by a 1-byte code.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** The basic programming foundation that carries computer messages around the globe through the Internet. The suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

**Unicode.** A character coding system designed to support the interchange, processing, and display of the written texts of the diverse languages of the modern world. Unicode characters are normally encoded using 16-bit integral unsigned numbers.

**URI/URL.** A Uniform Resource Identifier (URI) and Uniform Resource Locator (URL) uniquely defines a location on the Web. URIs are familiar to anyone who browses the Web (for example http://www.ibm.com), and the term URI is a more general term which also incorporates other schemes for identifying resources.

**valid.** An XML document is valid if its content conforms to the rules in its DTD.

**WAP.** Wireless Application Protocol. Offers internet browsing from wireless handsets

**Web Application.** A WebSphere Web application is a collection of static pages, JSPs and Servlets that share a common URL prefix, and together make a complete application.

**well-formed.** An XML document is well-formed if there is one root element, and all its child elements are properly nested within each other. Start tags must have end tags, and each empty tag must be designated as such with a trailing slash. Also, all attributes must be quoted, and all entities must be declared.

**white-space.** In XML, characters that are not visible, but used in formatting documents or programs. These characters include the SPACE, TAB, NEWLINE, and CARRIAGE-RETURN characters.

**XSL Stylesheet.** The eXtensible Stylesheet Language defines stylesheets for XML Documents. It is composed of two parts: the formatting objects, and XSLT (see below). XSL is defined by the WorldWide Web Consortium.

**XSLT eXtensible Stylesheet Language Transformations.** This defines the part of the XSL specification which allows the stylesheet to reformat and reorganize the XML data. It is most often used to transform XML into XSL.
Integrating XML with DB2 XML Extender and DB2 Text Extender
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>BSF</td>
<td>Bean Scripting Framework</td>
</tr>
<tr>
<td>CAE</td>
<td>client application enabler</td>
</tr>
<tr>
<td>DLL</td>
<td>dynamic link library</td>
</tr>
<tr>
<td>DAD</td>
<td>document access definition</td>
</tr>
<tr>
<td>DNS</td>
<td>domain name server</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GUI</td>
<td>graphical user interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
</tr>
<tr>
<td>IDE</td>
<td>integrated development environment</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>JAR</td>
<td>Java archive</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>NLS</td>
<td>National Language Support</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>PIN</td>
<td>personal identification number</td>
</tr>
<tr>
<td>RAD</td>
<td>Rapid Application Development</td>
</tr>
<tr>
<td>RDBMS</td>
<td>relational database management system</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SQL</td>
<td>structured query language</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Locator</td>
</tr>
<tr>
<td>VAJ</td>
<td>VisualAge for Java</td>
</tr>
<tr>
<td>XLink</td>
<td>relational database management system</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XPointer</td>
<td>XML Pointer Language</td>
</tr>
<tr>
<td>XSL</td>
<td>eXtensible Style Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>XSL Transformations</td>
</tr>
<tr>
<td>WAS</td>
<td>WebSphere Application Server</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
</tbody>
</table>
Integrating XML with DB2 XML Extender and DB2 Text Extender
Index

A
ASCII 131
attribute_node 73, 93
Attributes 13

B
B2B 33, 155, 190, 197, 201
B2C 33, 201
Basket bean 180, 183, 184
BasketBean 179, 183
BrowseSupplierServlet
See also servlets

C
C2B 155
CCSID 138
CDATA 14
CDRA 138
clearOrder() 184
code page 131, 138
coded character set 131
Column enablement 116
Comments 16
connection pool 204
connection pooling 170, 272
Content UDF 64
control tables 102
createEmptyBasket() 184
CRM 5
CustomerStorageServlet 258
See also servlets

D
DAD 35, 49
DAD creation 49
DAD_file 93
Data Definition Language (DDL) 162
database 161
Database enablement 113
databases
A118 250, 253, 270
L301 170, 172, 250, 251
ORDER 264
PROFILES 250, 253, 264, 270
SUP001 253
SUPPLIERS 162
XMLTEST 68
datasource 170, 172, 270, 272
DataSource type 172
DAXSL 254
See also servlets
DB2CODEPAGE 139
db2xml schema 102
db2xml.xml_usage 57
decompose XML documents 95
default_view 58
DefaultApplyXSL 183, 249, 254, 272
See also stored procedures
DHTML 29
Disabling the database for XML 102
Disabling the XML column 65
DOCTYPE 11
Document Access Definition 35, 49
Document Object Model (DOM) 25, 29, 32
API 29, 32
DocumentFragment 29
DOM HTML 29
Document Type Declaration 11, 12
Document Type Definition 9, 35
DTD 9, 10, 35, 50, 67
DTD files
dad.dtd 216, 223
order.dtd 230
OrderDetail.dtd 239
OrderList.dtd 237
product.dtd 223
Supplier.dtd 232
supplier.dtd 234
DTD ID 56
DTD_REF 50, 53
dxxadm 58
dxxEnableColumn 60
dxxGenXML()
See also stored procedures
dxxRootID 58
dxxShredXML()
See also stored procedures
dynamic queries 91

E
ECMAScript 29
Electronic Data Interchange (EDI) 4, 33
element_node 86, 99
element_nodes 72
Elements 12
ENABLE DATABASE 113
ENABLE TEXT COLUMN 116
ENABLE TEXT TABLE 116
enable_column 58
Enabling the database to XML 42
Enabling the XML column 57
encoding 131
encoding declaration 147
Entities 14
environment variable 111
external DTD 12
extract 197
extraction UDF 61

G
generate_unique 71
generateOrderList() 194
getConnection() 172
getOrderXMLString() 184, 187

H
HTML 3, 4, 8
HTTP 4
HttpServletRequest 172, 183
HttpSession 177

I
index 108
index characteristics 114
INITIAL_CONTEXT_FACTORY 171
Insert, delete, extract, and update functions 60
Installation 36
ISO
  IEC 10646 136
  IEC 2022 134
  IEC 646 132
  IEC 8859-1 135

J
Java Server Pages 155
JNDI 170
jndiname 204

L
L301
  See databases
Linguistic index 109
Locale 139
location path 36, 50
logon and registration 162
LogonServlet 258
  See also servlets

M
mapping schemes 67
mime-filter 167
MIME-TYPE 167

N
namespace 172
namespace prefix 17
Namespaces 16
National Language Support (NLS) 138
NCSA 7
NGram index 110
NLS 138
NO_OF_MATCHES 119
NO_OVERRIDE 91
Node 29
Notations 15

O
object ID 71
object-oriented database (OODB) 34
operators 92
order tracking 156, 162
orderbasket 180
orderby attribute 87
ordering mechanism 156
ORDERXML 162
ORDERXML1 162, 195
override 91
overridetype 91

P
Parameter Entities 15
parse 198
PCDATA 12
performTask() 193
Persistent Name Server 273
Precise index 110
processing characteristics 114
processing instruction 167
prolog 72
PROVIDER_URL 171

R
RDB_node 87, 93
RDB_node mapping 84, 86, 95
RegisterChangeOutputServlet 172, 258
See also servlets
RegisterChangeServlet 258
See also servlets
RegisterServlet 170, 258
See also servlets
registration process 156
RegistrationServlet 163
See also servlets
relational 67
retrieval UDF 64
Root_id 58
root_node 72

S
SAX 200
AttributeList 26
DocumentHandler 26
DTHandler 26
ErrorHandler 26
Locator 26
Parser 26
SAXHandler 200, 206
SAXParser 206
scalar extracting UDFs 62
search engine 162
search mechanism 156
search-argument 120
SERunner 274
ServletEngine 274
servletEngine 254
servlets
BrowseOrderServlet 192, 193, 205
CustomerStorageServlet 197, 258
DAXSL 254
DefaultApplyXSL 166, 167, 254
LogonPageServlet 213
LogonServlet 172, 258
MainMenuServlet 209, 268
RegisterChangeOutputServlet 258
RegisterChangeServlet 172, 258
RegisterFormServlet 211
RegisterServlet 170, 172, 212, 258
RegistrationServlet 163
SupplierOrderServlet 268
SupplierStorageServlet 258
UpdateB2BServlet 206, 258
UpdateServlet 205, 258
setProductQuantity() 185
SG246130 263
SG246130.ZIP 249, 251, 263, 264, 270
SG246130VAJ 269
shopping basket 156, 177
shopping basket and ordering subsystem 162
shopping-cart 155
dside table 49
side tables 57, 61
Simple API for XML 25
simple location path 92, 93
SQL mapping 69
SQL_OVERRIDE 91
Storage and access methods 34
storage UDF 61
stored procedures
dxxGenXML 223
dxxGenXML() 74, 88, 90, 99
dxxInsertXML() 99
dxxRetrieveXML() 90, 93
dxxShredXML() 98, 99
SupB2B 203
SupMainMenu 203
SupMainMenu.jsp 204
supplier side 162
SupplierStorageServlet 258
See also servlets

T
table extracting UDFs 62
tables
DTD_REF 35, 65
ORDERXML 162, 187, 197
ORDERXML1 162
PRODUCTS 162, 197
RESULT_TAB 75
SUPPLIER_SHIPPING 162
SUPPLIER_STORAGE 162
USERPROFILE 162, 172
XML_USAGE 35, 65
Tablespace 58
tag
  attribute_node 72
column 73
element_node 72, 73
SQL_stmt 73
text_node 72
template 169
text characteristics 114
Text Extender 103
Text Extender installation 103
Text Extender instance 105, 106
Text Extender UDF 119
Text Extender UDFs
  CONTAINS 119
  RANK 119
text_node 73, 87, 93
Topology 254
topology 253, 267
triggers 57, 65

U
UCS-2 136
UCS-2 database 144
UDF 36, 146
Unicode 136
Uniform Resource Identifier 16
Update UDF 64
update UDF 63
UpdateB2BServlet 258
  See also servlets
UpdateServlet 258
  See also servlets
URI 16
URL connection 187
usage_count 58, 65
user-defined functions (UDF) 102
user-defined types (UDT) 102
USERPROFILE 162
UTF-8 137, 146

V
validity 25
views 67
VisualAge for Java 159, 269

W
WebSphere 257, 259
WebSphere Application Server 159, 167, 171,
  249, 250, 252, 254, 266
WebSphere Studio 20
WebSphere Test Environment 269, 271, 273
WebSphereAdminDomain 254
well-formed 9
well-formedness 25

X
XML 65
XML collection 34, 67, 71, 155
XML column 34, 44, 155, 186
XML files
  LogonError.xml 214
  LogonForm.xml 213
  LogonOk.xml 214
  MainMenu.xml 209
  RegisterError.xml 212
  RegisterForm.xml 211
  RegisterInvalid.xml 212
  RegisterOk.xml 212
  XMLAppConfig.xml 249, 252
XML parsers 8, 9
XML Schemas 17
XML_OVERRIDE 91, 93
XML_USAGE 93
XMLCLOB 46, 61, 62, 64
XMLClobFromFile 50
XMLCLOBFromFile() 61
XMLConfig 252, 267
XMLFILE 46
XMLFile 61, 62
XMLFileFromCLOB 61
XMLFileFromVarchar 61
xml-stylesheet 167
XMLVarchar 46, 61, 62, 64
XMLVarcharFromFile() 61
XPath 18, 50, 92, 93, 200
XSL 20
XSL files
  basket.xsl 181, 182, 190
  LogonForm.xsl 168, 209, 211, 213
  orderDetail.xsl 235, 238
  orderedetail.xsl 193
  orderList.xsl 235, 236
orderlist.xsl 193
product.xsl 221
result.xsl 173
sample.xsl 167
user.xsl 215
XSLTInputSource 194
320  Integrating XML with DB2 XML Extender and DB2 Text Extender
IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.

- Use the online Contact us review redbook form found at ibm.com/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

<table>
<thead>
<tr>
<th>Document Number</th>
<th>SG24-6130-00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redbook Title</td>
<td>Integrating XML with DB2 XML Extender and DB2 Text Extender</td>
</tr>
</tbody>
</table>

Review

What other subjects would you like to see IBM Redbooks address?

Please rate your overall satisfaction:

- O Very Good
- O Good
- O Average
- O Poor

Please identify yourself as belonging to one of the following groups:

- O Customer
- O Business Partner
- O IBM, Lotus or Tivoli Employee
- O Solution Developer
- O None of the above

Your email address:

The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.

O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.

Questions about IBM's privacy policy?

The following link explains how we protect your personal information.

ibm.com/privacy/yourprivacy/

© Copyright IBM Corp. 2000
Integrating XML with DB2 XML Extender and DB2 Text Extender
Integrating XML with DB2 XML Extender and DB2 Text Extender

This IBM Redbook shows how to use XML technology efficiently in business applications, and explains how to integrate it with DB2 Universal Database, DB2 XML Extender and Text Extender, and WebSphere Application Server. This book will help developers to set up the environment and to create and process XML documents that can be stored and recovered using SQL.

In Part 1, we introduce the theory of XML, including some advantages of its use. We show its positioning in real-life situations and explain how to process XML using Java. Following this, we cover DB2 XML Extender, from its installation to enabling/disabling the database for XML. Then we cover DB2 Text Extender, also from its installation to the actual use of data. Finally, we offer some considerations on code pages and National Language Support.

In Part 2, we describe our XMLApp application, including two different implementations using XML Columns and XML Collections. We explain how to install both implementations in WebSphere Application Server, and how to install one of them in VisualAge for Java.

Anyone wanting to use XML technology to build better business solutions will benefit from reading this book.