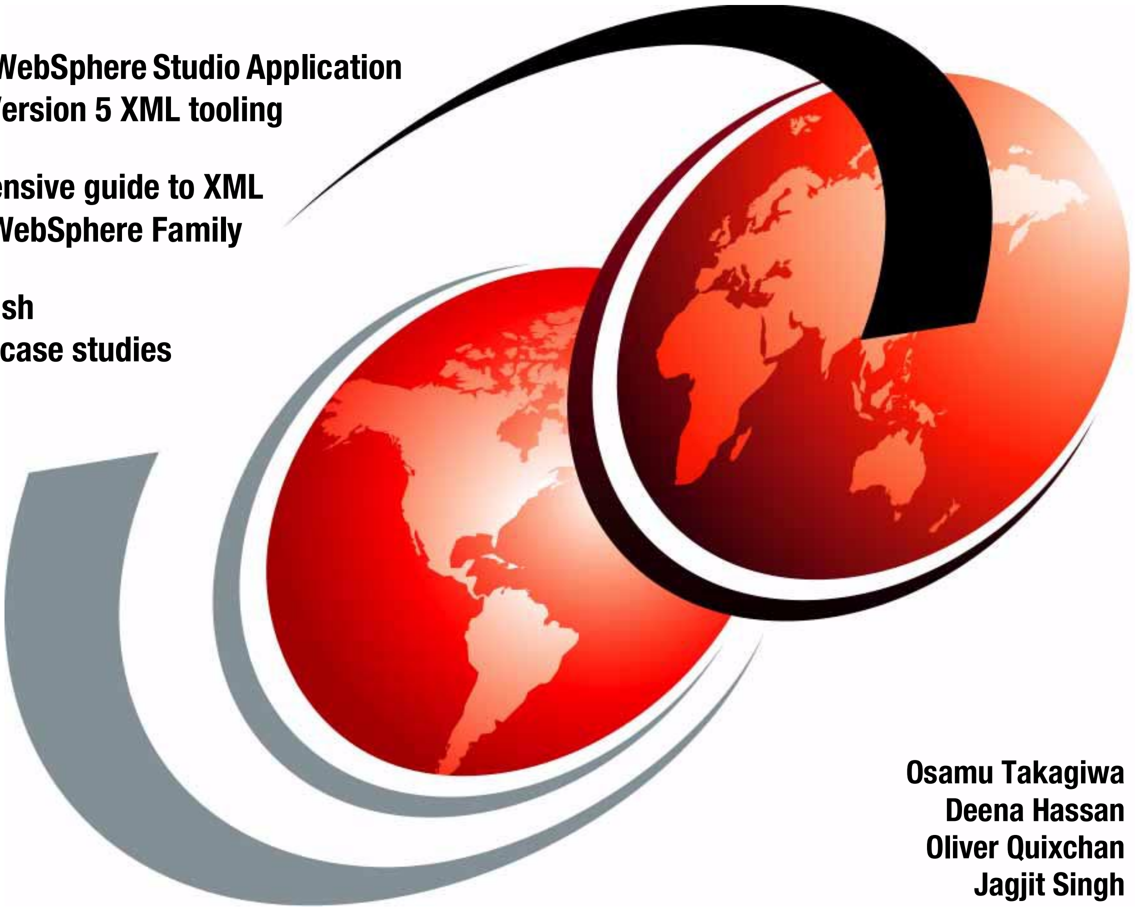
WebSphere® software

IBM

The XML Files:
Development of XML/XSL Applications Using WebSphere Studio Version 5



Introduces WebSphere Studio Application Developer Version 5 XML tooling

A comprehensive guide to XML support of WebSphere Family

Start-to-finish application case studies

Osamu Takagiwa
Deena Hassan
Oliver Quixchan
Jagjit Singh

Redbooks

ibm.com/redbooks

International Technical Support Organization

The XML Files: Development of XML/XSL Applications Using WebSphere Studio Version 5

December 2002

SG24-6586-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (December 2002)

This edition applies to Version 5 of IBM WebSphere Studio Application Developer.

Contents

Notices . . . ix
Trademarks . . . x

Preface . . . xi
The team that wrote this redbook . . . xi
Become a published author . . . xiii
Comments welcome . . . xiii

Part 1. Introduction to XML technology . . . 1

Chapter 1. XML overview . . . 3
1.1 XML concepts . . . 4
1.2 Background . . . 4
1.3 XML business benefits . . . 5
1.3.1 Information sharing . . . 5
1.3.2 XML within an organization . . . 6
1.3.3 XML in new innovations . . . 7
1.4 Technical benefits of XML . . . 7
1.5 XML history . . . 9
1.6 XML1.0 and 1.1 . . . 10
1.7 XSLT and Web applications . . . 11
1.8 Web services and XML . . . 13
1.9 XML, W3C, and IBM . . . 14

Chapter 2. Technologies in XML . . . 17
2.1 XML Processor (parser) . . . 18
2.2 DTD and XML Schema . . . 19
2.3 Schema and style using CSS, XSLT, and XSL . . . 21
2.4 XML namespaces . . . 23
2.5 Link and jump using XLink, XPointer, and XML Base . . . 25
2.6 XPath . . . 33
2.7 XML digital . . . 34
2.8 XML query language . . . 40
2.9 XSLT compilers (XSLTC) . . . 47
2.10 Java Architecture for XML Binding (JAXB) . . . 49
2.11 Cocoon . . . 49

Part 2. XML technology in IBM WebSphere . . . 53

Chapter 3. Processing XML 55
3.1 XML applications 56
3.2 Xalan 57
3.3 SAX2 58
3.4 DOM level2 62
3.5 JAXP 65

Chapter 4. Introduction to IBM WebSphere Application Developer 69
4.1 WebSphere Studio product family 70
4.2 Tools 73
4.2.1 Web development tools 73
4.2.2 Relational database tools 74
4.2.3 XML tools 74
4.2.4 Java development tools 75
4.2.5 Web services development tools 75
4.2.6 EJB development tools 76
4.2.7 Team collaboration 76
4.2.8 Debugging tools 76
4.2.9 Performance profiling tools 77
4.2.10 Server tools for testing and deployment 77
4.2.11 Plug-in development tools 78

Chapter 5. Application Developer XML Tools 79
5.1 XML perspective 80
5.2 XML perspective editors 81
5.2.1 XML editor 81
5.2.2 DTD editor 83
5.2.3 XSD editor 85
5.2.4 XSL editor 89
5.3 Namespace support 90
5.4 XPath support 92
5.5 XSL debugger 96
5.6 Web services support 98

Chapter 6. RDB and XML integration 101
6.1 The SQL to XML wizards 102
6.2 The XML to SQL wizard 119
6.3 The DDL to XML Schema wizard 122
6.4 DB2 XML Extender 126
6.4.1 XML Collection 134

Chapter 7. Generators 145
7.1 DTD <—> XSD 146
7.2 XML <—> DTD/XSD 147

7.2.1 Create an XML file from a DTD file 147
7.2.2 Create an XML file from an XSD file 148
7.2.3 Create DTD/XSD files from XML. 149
7.3 Generate a HTML from an XSD 151
7.4 JavaBeans from DTD/XSD 152
7.5 Generate XML/XSL from JavaBeans 153
7.6 Generate XML/XSL from HTML 156
7.6.1 Preparing the HTML file for generation. 157

Part 3. XML application development 161

Chapter 8. WebSphere and XML approaches 163
8.1 XML in Application development. 164
8.2 Web services. 165
8.3 Passenger List application 166
8.3.1 Solution Outline. 167
8.3.2 XML in this application 168
8.3.3 Technical implementation overview 169
8.4 Enterprise JavaBeans 172
8.5 The Customer Registration application 174
8.5.1 XML in this application 174
8.5.2 Technical overview 175

Chapter 9. Developing XML Web services 177
9.1 Passenger List application 178
9.1.1 Creating the Web tier 178
9.1.2 Create the Airline simple project 179
9.1.3 Create the Travel Web project 180
9.1.4 Design the XML Schema 180
9.1.5 Generate XML file 184
9.1.6 Design an output page 184
9.1.7 Testing the XSL. 189
9.1.8 Developing the servlet. 189
9.1.9 Test the passenger list application 191
9.1.10 Compiling XSL 191
9.2 Creating a Web service 192
9.2.1 Create the database tier 193
9.2.2 Generate DTD file 194
9.2.3 Loading DTD into XML Extender 195
9.2.4 Creating DAD file using RDB to XML mapping 196
9.2.5 Create the Web Service from DADX file 199
9.2.6 Test the Web Service 207
9.2.7 Modify passenger list application to use the Web Service 208

Chapter 10. Development of XML-based Enterprise applications 215
10.1 XML based Enterprise application architecture. 216
10.2 Solution outline for customer registration sample 217
10.3 Developing the customer registration sample 219
10.3.1 Creating the Web tier 220
10.3.2 Building the entity EJB and the database schema 238
10.3.3 Integrating the entity EJB with the Web tier 250
10.3.4 Retrieval function 253
10.4 Application deployment and testing. 253
10.4.1 Testing the registration application 254

Chapter 11. Light weight XML-based Enterprise Application 257
11.1 SQL to/from XML libraries 258
11.2 Solution outline for customer registration sample 259
11.2.1 Customer registration 259
11.2.2 Retrieving customer information 260
11.3 Developing the customer registration sample 262
11.3.1 Adding the libraries to the project 262
11.3.2 XML Document format 262
11.3.3 XMLToSQL architecture 263
11.3.4 Modifying CustomerXSLServlet 264
11.3.5 Retrieving a customer 270
11.3.6 Using datasource with SQLToXML and XMLToSQL class 277
11.3.7 Conclusion 281

Chapter 12. Deploying your Web application 283
12.1 Manual deployment 284
12.1.1 Exporting your project from Application Developer 284
12.1.2 Installing the EAR file on WebSphere AEs 285
12.1.3 Starting the WebSphere AEs Admin Console. 285
12.1.4 Installing the EAR 286
12.1.5 Testing the application 287
12.2 Publishing to a remote server (AEs) 288
12.2.1 Creating a remote server instance 288
12.2.2 Publishing to remote server 295
12.2.3 Testing the application 295

Part 4. Appendixes 297

Appendix A. Installing WebSphere Studio Application Developer 299
Things to do before installation 300
Installing Application Developer 300
Selecting your workspace 302
Verifying the installation 303

Appendix B. Installing IBM WebSphere Application Server 4.0 AEs . . . 305
Things to do before installation 306
Hardware and software prerequisites 306
Create groups and users. 306
Check that IP ports are unused. 307
Stop the Web server processes 307
Install WebSphere Application Server 307
Verifying the installation 309

Appendix C. Additional material 311
Locating the Web material 311
Using the Web material 311
System requirements for downloading the Web material 312
How to use the Web material 312

Abbreviations and acronyms 313

Related publications 315
IBM Redbooks 315
Other resources 315
Referenced Web sites 315
How to get IBM Redbooks 316
IBM Redbooks collections. 316

Index 317

Notices

Trademarks

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

Redbooks(logo)™
AIX®
alphaWorks®
CICS®
DB2®
IBM®
IMS™
Informix®
MQSeries®
Perform™
SAA®
SOM®
SP™
VisualAge®
WebSphere®

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

Lotus®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Preface

In today's technology, XML is becoming a key piece of software infrastructure. The main idea is extremely simple. It is a language like HTML and is text based, but is rigidly enforced, and therefore, can be built upon easily. XML documents may use a Document Type Definition (DTD) or an XML Schema. XML was designed to describe data and to focus on the data, unlike HTML, which was designed to display data. It was created to structure and store data.

This book is separated into three parts:

Part one introduces the eXtensible Markup Language (XML) and how it can be used in today's technology and provides the reader with an opportunity to learn about the processing of XML documents.

Part two contains an introduction to the concepts behind WebSphere Studio Application Developer, and an overview of the features of the various members of the WebSphere Studio family of tools including wizards and generators.

Part three, we provide an overview of Web Services and Enterprise JavaBeans capabilities of Application Developer. We also develop several variations of Web applications that works with RDB, Web Services, or Enterprise JavaBeans.

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.

Osamu Takagiwa is an advisory IT specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of application development. Before joining the ITSO 1.5 years ago, Osamu worked in IBM Japan as an I/T specialist.

Deena Hassan is an IT specialist at the Cairo Technology Development Center (CTDC) of IBM Egypt. She has experience in application development, particularly in the e-commerce field. She holds a degree in Computer Science from the American University in Cairo, and is about to recieve her master's degree in the field of Artificial Neural Networks. Her areas of expertise include Enterprise Java programming and e-commerce solutions.

Oliver Quixchan is an e-business consultant from Guatemala. He holds a BS in Computer Science from Universidad Francisco Marroquin, Guatemala. He has experiece in application development, Web-oriented solutions using J2EE and WebSphere software platform, banking software development, and middleware design. His areas of expertise include WebSphere Application Server, WebSphere Host Pblisher, XML, J2EE.

Jagjit Singh is an IT architect with IBM Global Services, Australia. He has 14 years of experience in application development and architecture. He holds a master's degree in Computer Science from the University of New South Wales, Sydney, Australia. His areas of expertise are in e-business, middleware, and Internet design and implementation on Windows NT and UNIX environments.

Thanks to the following people for their contributions to this project:

Maritza Marie Dubec
Emma Jacobs
International Technical Support Organization, San Jose Center

Christina Lau
IBM Canada Toronto Laboratory

Arthur Ryman
IBM Canada Toronto Laboratory

Susan Malaika
IBM Silicon Valley Laboratory

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Part 1

Introduction to XML technology

Part one introduces the eXtensible Markup Language (XML) and how it can be used in today’s technology. This highlights the XML Processor and parser shipped with Websphere Application Developer.

1 XML overview

“I think we should shoot for a goal within the developed countries of having Internet access as complete as telephone access within a fixed number of years. It will do as much as anything else to reduce income inequality.”

Bill Clinton, President of the United States of America, November 21, 1999

This chapter introduces the eXtensible Markup Language (XML) and how it can be used in today’s technology. It explains its background since it appeared on the horizon, its evolution and its benefits both within the industry and also its use within an organization. It discusses two innovations that have gained firm ground among the many that are being put onto the e-business market over the last few years. It also lists IBM contributions to the World Wide Web (W3C) group.

1.1 XML concepts

In today’s technology, XML is starting to becoming a key piece of software infrastructure. The main idea is extremely simple. It is a language like HTML and is text based, but is rigidly enforced, and therefore, can be built upon easily. XML documents may use a Document Type Definition (DTD) or an XML Schema.

XML was designed to describe data and to focus on the data, unlike HTML which was designed to display data. It was created to structure and store data.

1.2 Background

The Internet has had amazing growth during the last 8 years. However, in the last four years, its usage has increase exponentially. In the first 4 years, its foundations were laid, and in the later 4 years, a number of technologies have evolved around it. Today, it is in the mainstream of life, and everyday, every talk revolves around its usage. Nowadays, advertising mandates that a Web page address be displayed and e-mail is considered official documents.

This growth has been fueled by many factors. The cost of personal computers has decreased dramatically, network technologies are more widespread and cheaper, schools are emphasizing computer literacy at an early age. Its ease of use has been its biggest contributor. Access to the Internet is as easy as logging through the phone line and the local area network. Computer education is now part of the school curriculum at the primary level, so that today’s youngsters do not lose out on the this new revolution.

Tim Berners-Lee, who leads the World Wide Web Consortium (W3C), the inventor of HTML and HTTP, could not have envisaged this growth. Even, computer giants like IBM and Microsoft had to realign their technical and business philosophies to cater for the explosive growth. To not follow, meant to be left behind. It would be true to say, we cannot envisage what the Internet will bring in the next five years, only to say that there will be new ways we will be doing business and enjoying leisure.

This ability to manage large amount of simple and complex data in many forms, without limitations, and display in a readable format, can be based for this universal data format. As we know, this can be provided by the eXtensible Markup Language (XML).

1.3 XML business benefits

The benefits of a common base upon which the technical world can build layers upon layers of technical innovation are enormous. This will only be possible if this standard has been agreed to by all.

XML has three main applications:

Sharing of information: The main problem integrating data between any two business organizations is the interface between them. If they can at least agree upon the standard of their common meeting point and its usage, they can build upon this to start building their applications. If there is already an existing interface or infrastructure provided by industry or government standard or infrastructure, the business cost of developing it is extinguished.

Storage, transmission and interpretation of data: If the storage of information is adaptable to many mediums, its cost will be driven to the lowest cost of all mediums. XML is based on text, which obviously is accepted by all. It can be read. Its storage is cheap, compared to the storage requirements of graphics. And because the size of this text based object is small, its transmission, not withstanding cost, is cheap as well. And because it is commonly accepted, since it adheres to world wide standards, it can be easily interpreted.

Security: With the explosion of confidential information being transmitted across the Internet, there is now a need for sophisticated levels of security. Companies need to protect their documents and e-mail, banks need to allow their depositors to download their accounts and merchants need to be available their customers to enter their credit card details without compromising their security and privacy. The more secure the transmission medium, the more confidentiality it provides, the more it can be used to advertise a competitive advantage. With the evolution of XML digital signatures, the ability to protect or hide parts of a document, while sitting on a PC, server or mainframe, now covers up a security patch.

Speed and amount of content delivery: With the rapid evolution of network technologies, speed and delivery of any object has gained importance. Network companies now advertise download times of movies and CDs. Again, the first companies discovering the abilities of delivering something at an increasing speed without compromising their content will gain the competitive advantage.

1.3.1 Information sharing

XML has been readily accepted by the technical world because of its simplicity. The benefits of having a common format to share information between any two organization are obvious. Technologies and standards have been built upon XML. Consortiums and business organizations have developed industry wide

XML formats and standardization. Examples of these are inter-bank payment between banks, insurance companies and trading agencies, supply-chains between manufacturers, distributors and retailers, battlefield information between soldiers, satellites and defence analysts.

In December 2000, the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organizations for the Advancements of Standard Information Standards (OASIS) came together to initiate a project to standardize XML specifications for business. This initiative called the Electronic Business XML (ebXML) developed a technical framework that enabled XML to be utilized for all exchange of all electronic business data. The main aim of ebXML was to lower the cost and difficulties, focusing on small and medium sized business and developing nations, in order to facilitate international trade.

Another example, BizTalk, is an initiative supported by Microsoft, SAP, and Boeing, among other major vendors. However, BizTalk is not a standard, more of an community of users. Its aim to enable consistent adoption of XML to ease use, and therefore, easily adopt electronic commerce and application integration.

1.3.2 XML within an organization

With the emergence of Customer Relation Management (CRM) and Enterprise Architecture Integration (EAI), customer oriented organizations have re-engineered their systems to provide a whole new experience for the consumer.

CRM and EAI involve bringing together a multitude of applications and systems from multiple vendors to behave as a coordinate whole. The common thread between all these applications is XML. Imagine trying to integrate information from billing histories, customer personal information and credit histories, distribution systems, and workforce management systems; and then displaying them using browsers, databases, processes, and workflow systems on a variety of technical platforms like mainframes, PCs, and medium sized servers. All this information can be shared by using XML, because it is a versatile, powerful, and flexible language. It has the ability of describing complex data. It is also extensible, allowing applications to grow and develop without architectural re-engineering.

Needless to say, IBM has used XML in all its new tools. IBM’s mainstay applications: WebSphere Studio, DB2, and WebSphere Application Server are based upon XML with their extensibility being a major advantage.

1.3.3 XML in new innovations

There have such a large number of innovations based on XML, they are too numerous to list. New ideas are now based in XML. Entrepreneurial companies now cannot avoid XML and its standards.

Voice XML

In October, 2001, the World Wide Consortium (w3C) announced the first release of a working draft for the Voice Extensible Markup Language (VoiceXML). VoiceXML has these purposes among others:

- It will hide designers and developers from low-level platform specific details.
- It will promote portability across implementation platforms.
- It will be a common language for platform providers, development tool providers and content providers.
- It would provide features to support complex dialogs, yet be easy to use for simple interactions.
- It is designed to cater for audio dialogs that would feature digitized audio, speech recognition, telephony and synthesized speech. Its goal would be to deliver advantages of Web-based development and context delivery for interactive voice response applications.

Scalable Vector Graphics

Scalable Vector Graphics (SVG) is an XML based language for describing two-dimensional graphics. Its main use is in Geographical Information Systems (GIS) where travel maps, council boundaries, forest fires and natural disasters can be processed and displayed independently of technical platforms. It will be able to integrate with non-GIS software, and also allow graphic elements to non-graphic.

The benefits of this new XML derivative are not difficult to comprehend. It uses are many. For example, insurance companies can estimate and forecast natural disaster claims; and scientists can study environmental impacts, and local and federal governments for city and town planning.

1.4 Technical benefits of XML

Fundamentally, the basics of XML have not changed. What has changed is the extent XML is being used today, and how it is incorporated into technologies for the future. XML offers many benefits, some of which are stated below.

Acceptability of use for data transfer

XML is not a programming language. It is a standard way of putting information in a format that can be processed and exchanged across hardware devices, operating systems, software applications, and the Web. It has become such a common medium of data that it enables the transmission and retrieval, and storage of information over the Internet across company boundaries, making it a natural choice for data management for e-business transactions.

Uniformity and conformity

The inability of two computer systems or applications to talk to each other is always a challenge. When two applications are integrated, the business and technical experts must decide either to integrate the two systems, or to re-architect the applications. If data from both applications, conform to a format and is easily transformed from one to another, development costs can be reduced. If this common format could be developed upon and is accepted industry-wide, then interfacing the applications to other applications is less costly.

Simplicity and openness

Information coded in XML is visually read and accepted, because it can be easily processed by computers, XML is widely accepted by major vendors in the computing world. Microsoft has indicated that it will use XML as the exchange format for its Microsoft Office software suite. Both Microsoft's and Netscape's Web browsers support XML.

XML has garnered interest because it is very simple to understand, implement, and use. It follows the Pareto principle, a 80/20 solution, meaning it supplies about 80 percent of the functionality of competing technologies with perhaps 20 percent of the effort required to build Enterprise-level solutions.

XML is not a total solution for every problems in e-business, but has made, and is making significant inroads in communications between old computer programs. That means these old programs last longer, saving money and time, which are important when both are so precious to the bottom line.

Separation of data and display

Separation of data and its display is not a new paradigm in the computing world. In the last few years, application designers have raised the concept of the Model-View-Controller (MVC) approach to building applications. There are many reasons for this. Firstly, without separation of data, re-use of that data in multiple user interfaces would be difficult. Web sites have evolved radically over the last 8 years. Evey year, Web sites have to be upgraded to compete for consumer attention. They have to have better attention getting displays and response times. If the data had to be re-hashed everytime an upgraded to the Web site was required, the development cost would rise. However, if the data could be re-used or built upon, there would be a re-development savings.

Imagine a navigation system used by consumers to move from one place to another. This system would have street maps, yellow pages information, local attractions and other information. If this information has to be displayed on a Web browser, personal device assistant (PDA) or a mobile phone, it would be a major development costs if we had to develop three separate data access systems and three data presentation systems. However, if we develop one data access systems, we also need to create three data presentation files for each system, which transforms XML using the XSLT transformation feature.

Extensibility

HTML has a major problem in that it is not extensible. It has been enhanced upon by the different software vendors. These enhancements have not been co-ordinated, and therefore, have become non-standard. It was never designed to access data from databases. To overcome this deficiency, Microsoft built Active Server Pages (ASP) and Sun produced Java Server Pages (JSP).

As the name implies, XML was designed from the beginning to allow extensions.

Industry acceptance

XML has been accepted by widely by the information and computing Industry. It is based on common concepts. As time goes on, a large number of XML tools will emerge from both existing software vendors and XML startup companies. It is readable by every operating systems, because it is in ASCII text. This implies that it can be seen by any text editor or word processor.

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.

1.5 XML history

The history of XML is really the history of another system: *Standard Generalized Markup Language* or SGML. XML is actually just a subset of SGML, and SGML has been around for many years. In fact, SGML dates back to the late 60s and the work of an IBM employee Charles Goldfarb. Goldfarb was developing a system to share documents, and together with two of his colleagues, Edward Mosher and Raymond Lorie, he put together a markup language called *Generalized Markup Language* (GML). (Of course, GML really stands for Goldfarb, Mosher, and Lorie, who invented it.)

GML also drew on work by William Tunnicliffe and Stanley Rice on building a *generic coding* system, which was done under the auspices of the Graphic Communication Association.

GML formed the basis of many IBM documentation systems including Script and Bookmaster. Later developments led to SGML, which became an ISO standard in 1986. SGML has had a great deal of success, but unfortunately, it has mainly been limited to large corporations and government departments. The reason was that SGML required a major investment, and so, only large organizations had the resources to achieve the benefits of SGML. For more information on SGML and its history, read *The SGML Handbook*, ISBN 0198537379.

In 1996, W3C sponsored a group of SGML experts to define a markup language with the power of SGML and the simplicity of HTML. The team abandoned the non-essential and cryptic parts of SGML. The remainder was a cut-down specification of 26 pages on XML. SGML had a specification of 500 and more pages. The new specification, however, managed to conceptualize the main ingredients of the older language.

Over the following years, XML evolved with the help of developers having similar problems. At that time, Chemical Markup Language (CML) and MathML were being formulated, and the eTensible Linking Language project was gaining speed. Finally in 1998, the W3C approved Version 1.0 of the XML specification and a new language was born.

Since the end of 1997, XML has grown quickly under the leadership of W3C, and in September 1999, W3C announced that XML activity was entering its third phase: Phase one built the base technology; phase two created stylesheets and namespaces; and phase three will endeavor to finish the ongoing work and introduce new specifications for an XML query standard.

Since then, as we know now, the usage of XML has exploded. We can only say that its application will continue for a long while to come. Since 1997, we have seen a multitude of tools evolving, including parsers, transformers, protocols, standards, and business-to-business integration.

1.6 XML1.0 and 1.1

The working draft for XML 1.1 was published on W3C Web pages in April 2002. XML 1.1 was formerly known as XML Blueberry.

The XML 1.0 specifications was based on the Unicode Standard. However, the Unicode standards have evolved from version 2.0 to 3.1 and beyond. XML 1.0

relied on the standard for character specifications. Characters that are not present in Version 2.0 would probably have be used in XML documents and character data. Developers would have developed workarounds for characters that were not supported in Unicode Version 2.0. These characters are not allowed in XML names such as element type, names, and attribute names, just to name a few. Also, some characters that should have been permitted in XML 1.0, but were not due to oversights and inconsistencies in Unicode 2.0.

The philosophy for names has been reversed since XML1.0. XML1.1 names have been designed such that everything that is not forbidden is permitted. In XML1.0 the philosophy was everything that was not permitted was forbidden. For example, under XML1.0, if only ‘a’, ‘b’, ‘c’, ‘d’ and ‘e’ were allowed as names, that was that could be used. In XML 1.1, we could say that we would not allow ‘a’ and ‘b’. Therefore, we could use ‘b’, ‘c’, ‘e’, ‘f’, ...’g’, ...’$’, etc. As Unicode grows past Version 3.1, changes to the XML can be avoided if nearly all characters are allowed. This will allow any kind of characters in a name.

XML1.0 discriminates against conventions used on IBM and IBM-compatible mainframes. XML documents on mainframes and not plain texts. Now, the Unicode line separator, #x2028, is also supported.

A new version of XML is being created, rather than a set of errata, because the changes affect the definition of well-formed documents. XML Processors will recognize XML 1.1 documents from XML 1.0 documents by the declaration at the start of each document.

For more details, visit W3C Web site:

`http://www.w3.org/TR/xml11/`

1.7 XSLT and Web applications

Extensible Stylesheet Language Transformations (XSLT) is designed to transform XML data into some other form. The most common form the transformation occurs to is HTML. Transformation to HTML is the final step before the user sees the Web page.

The main problem with HTML is its unorganized implementation. It evolution was driven mainly by extreme competition between Netscape and Microsoft. Each vendor tried to gain market share by its own browser specific tags and varying support for standards. To create a HTML page that would work on all browsers, developers therefore had to restrict themselves to tags that were generic to all browsers. A different approach would be to maintain a HTML page for each kind of browser.

Presently, Web services theoretically need to support a multitude a browsers. The data sent to the browsers would be common to all, but the presentation of the data could be dependent on the browser. In effect, the presentation of the data could be PDA or even a mobile phone.

An XSLT stylesheet is a XML document. A XSLT processor transforms one or more XSLT stylesheets. Typically, in a Java and XSLT based Web application, Java is used to access the data from the database. XML data is then generated dependent on the data. With the latest release in database technologies, some databases have now the ability to export the data into XML. However, usually programmers write the code to extract the data and convert it into XML.

The processes takes the XML file as one input, and the XSLT stylesheet as the second input, and produces a third output, which is usually HTML, but could be another XML document. The XML file has the data to be presented while the XSLT stylesheet details the logic of how the output data is to be presented.

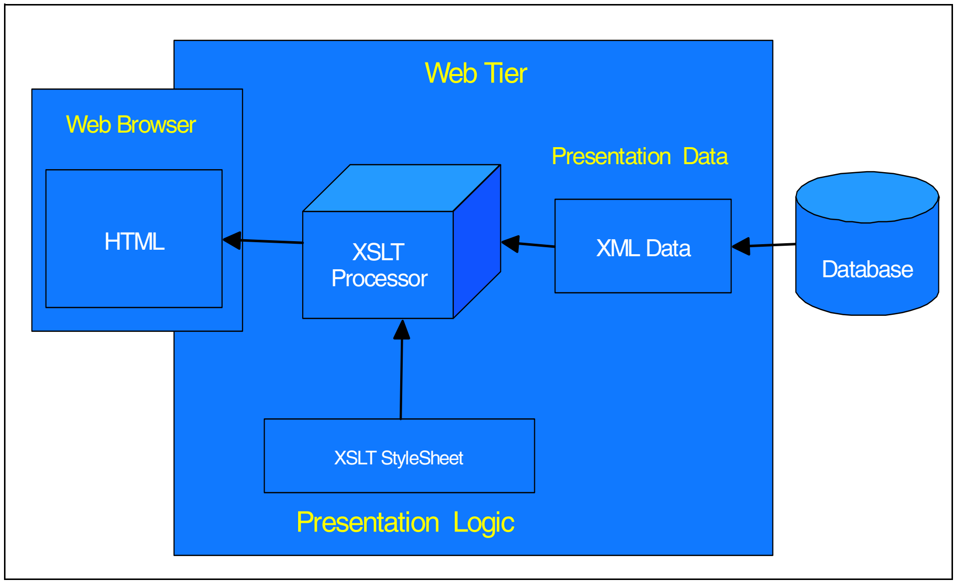


Figure 1-1   XSLT and Web applications

A Web application is an application with its presentation layer (or user interface) on the World Wide Web. These applications display graphics, text, and animation. It tends to be custom made and uses a host of technologies like browsers, databases, and programming languages to name a few. Because of the number of technologies and the skills sets required of its designers and developers, Web applications have to be modular. While the graphic designer concentrates on the HTML user interface, the database programmer focuses on extracting the correct data. A Java programmer might be working on the Web tier.

XSLT has become a critical part of Web applications, because it allows versatility in the presentation logic. W3C has published a working draft for XSLT Version 2.0 and XML Path Language (XPath) Version 2.0 in April 2002. The later had a major IBM involvement.

1.8 Web services and XML

Web services have gained prominence in the last three years. They are the new middleware that will glue all kinds of disparate applications from different vendors. Presently, there are a few major vendors touting middleware: TIBCO, BEA eLink/Tuxedo and IBM's MQ-series. Web services on the other hand are not proprietary. They are self-contained, modular applications, self explanatory applications that are published on the Web. They provide functions that encapsulate anything from single functions to complex business functions. An example of a simple function may be a calculator, spreadsheet, or a tutorial. An example complex functions could be the processing of a tax return, stock quotes, or processing credit card transactions. Once this Web service has been deployed, anyone (be it another application or Web service) would be able to locate and invoke it.

When we discuss Web services, we involve a few components:

Simple Object Access Protocol (SOAP) is an XML-based protocol that allows applications to invoke applications provided by service provides anywhere on the Web. It is supported by HTTP, and therefore, can be run on the Internet without any new requirements over existing infrastructure. It is independent off any programming language and component technology, and is object neutral. It is also independent of operating systems.

Universal Description Discovery and Integration (UDDI) is a specification for Web registries of Web services. Web users locate and discover services on a UDDI-based registry. There are registries of services distributed over the Internet, and these registry of services are described in a common XML format or schema. For a common format, searching and analysis of applications would be made much easier. A UDDI registry has been made available from IBM's alphaWorks Web site, and supports users in various department- or company-wide scenarios.

Web Services Description Language (WSDL) is a language that is used to describe a service to the world. The definition of WSDL is:

> "WSDL is an XML format for describing network services as a set of end-points operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message

format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME."

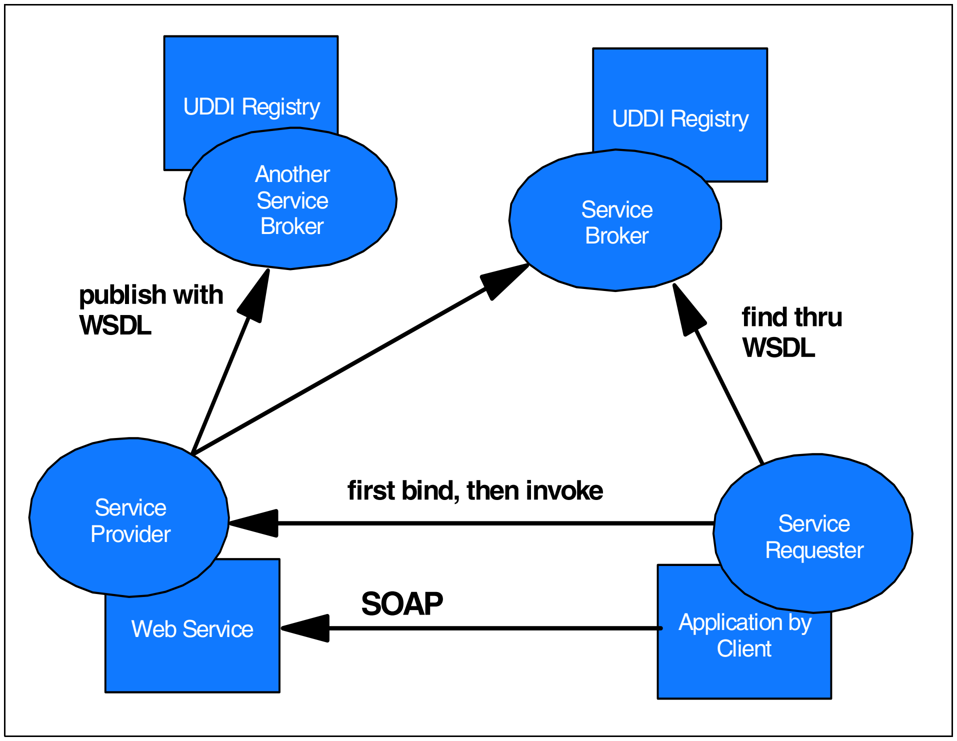


Figure 1-2 Interaction of Web services

1.9 XML, W3C, and IBM

The World Wide Web Consortium (W3C) creates Web standards for the Internet. It formed a Web service activity group in January 2002. The goal of the Web service activity is to develop technologies in order to bring Web services to their full potential. It consists of three working groups: Web Services Architecture Working Group, XML Protocol Working Group, Web Services Description Working Group.

The W3C regularly publishes technical reports and publications. IBM has made major contributions to the W3C. A list of publications where IBM researchers have been involved in follows. The list is not exhaustive.

Recommendations

"A recommendation is work that represents consensus within W3C and has the director's stamp of approval. W3C considers that the ideas or technology specified by a recommendation are appropriate for widespread deployment and promotes W3C's mission."

- Extensible Stylesheet Language (XSL) Version 1.0
- Synchronized Multimedia Integration Language (SMIL 2.0)

Candidate recommendations

"A candidate recommendation is work that has received significant review from its immediate technical community. It is an explicit call to those outside of the related Working Groups or the W3C itself for implementation and technical feedback."

- XML encryption syntax and processing
- Decryption transform for XML signature

Working drafts

"The following working drafts have been submitted for review by W3C members and other interested parties. These are draft documents and may be updated, replaced, or made obsolete by other documents at any time. It is inappropriate to use W3C working drafts as reference material or to cite them as other than work in progress."

- SOAP Version 1.2 Part 1: Messaging framework
- SOAP Version 1.2 Part 2: Adjuncts
- Voice Extensible Markup Language (VoiceXML) Version 2.0
- XForms 1.0
- XMl events

Working drafts in development

- XML protocol usage scenarios
- XML protocol requirements
- XML query use cases
- XML Path Language (XPath) 2.0
- XQuery 1.0: An XML query language
- XQuery 1.0 formal semantics

2

Technologies in XML

This chapter discusses some of the latest technologies being introduced in the XML arena. Some of these have been around for the last three years or so, and are undergoing enhancements and the others are fairly new.

The material is intended for introduction only, where detailed investigation of each of these technologies is a subject by itself. The material forms a good groundwork for concepts required in developing of many applications in WebSphere Studio Application Developer. The following technologies are discussed:

- XML Processor
- DTD and XML Schema
- CSS, XSLT and XSL
- XLink, XPointer and XBase
- XML Digital Signature
- XML Query Language
- XSLT Compilers (XSLTC)
- Java Architecture for XML Binding
- Cocoon

2.1 XML Processor (parser)

An XML Processor can either be a validating or non-validating parser. Both kinds of parsers report violations on an XML document. According to the XML 1.0 specification:

`http://www.w3.org/ TR/REC-xml#proc-types`

> "Validating processors must, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML Processors must read and process the entire DTD and all external parsed entities referenced in the document."

Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to process all the declarations they read in the internal DTD subset and in any parameter entity that they read. This is done up to the first reference to a parameter entity that they do not read; that is to say, they must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. Except when `standalone="yes"`, they must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

From the definition above, a validating parser must read the entire DTD and check the XML document against it. A non-validatiing parser may not need the DTD must still check the XML against default values for attributes. Both parsers check for the well-formedness of the document.

Most parsers can be run in validating and non-validating mode. Validating of XML documents is crucial in the development and testing stage of the software development life cycle. However, running validation has a performance cost. In production, when the reliability of the data of a system is already established, and they are expected to have complex DTDs and XML Schemas, the validating can be turned off. Some parsers are non-validating by default.

Parsers can be of two types: tree-based parsing or event-based parsing. These will be further discussed in Chapter 3, however, here is an overview:

Tree-based parsing

In tree-based parsing, the parsers attempts to create an hierarchal structure for the entire document. For a hugh document, this will be extremely memory-sensitive. The parser will make the elements and attributes available only after it has parsed the whole document. However, once the document has been created in memory, it can be navigated and changed. A DOM parser would be a tree-based parser.

Event-based parsing

These parsers process the document as it encounters the tags of the document. It is a data-centric view of the XML. Whenever an element or tag is encountered, it (or its contents) can be processed. However, it cannot backtrack once the tag has been passed. The parser returns the element, its attributes and the contents. The event-based parser never attempts to build a structure of the data, and therefore, its memory requirements are less. It comes in useful, when one is looking in the document only for certain elements. A SAX parser would be an example of a event-based parser.

The most popular XML parsers on the market is the Apache XML Project's Xerces. The parsers provides XML parsing and generation, and are fully-validating parsers available for both Java and C++, implementing the W3C XML and DOM (Level 1 and 2) standards, as well as SAX (Level 2) standard. The parsers also support for XML Schema. This parser has been incorporated into the IBM set of products (WebSphere, Application Studio and DB2).

Another parser is IBM's XML Parser for Java (XML4J and XML4C). The XML4J is a validating XML parser written in 100% pure Java, whereas XML4C is a validating XML parser written for C++. It provides classes for parsing, generating, manipulating, and validating XML documents. Both parsers are support the XML 1.0 Recommendation and associated standards (DOM 1.0, SAX 1.0, DOM 2.0). XML4J contains implementations of the DOM Level 2, the SAX Level 2 implementations, and parts of W3C schema, but these are experimental at this stage. XML4C is supported on most operating systems including AIX and Linux.

Both parsers are open source and have the same code base, where the XML4J parser has the latest code enhancements, while Xerces has been through production level testing.

2.2 DTD and XML Schema

DTDs and XML Schema are both used to describe structured information, however, in the last two years acceptance of XML Schema has gained momentum. Both DTDs and schemas are building blocks for XML documents and consists of elements, tags, attributes, and entities

XML Schemas evolved to overcome limitations in DTDs. W3C has three documents published, the latest update being in May 2001:

- XML Schema Part 0: This is a primer, it is intended to provide an easily readable description of the XML Schema facilities, and is oriented towards quickly understanding how to create schemas using the XML Schema language.
- XML Schema Part 1: Structures specify the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility.
- XML Schema Part 2: Datatypes 2: It defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications.

DTDs consists of elements that text string, text string with other child elements and a set of child elements. DTDs also offer limited support for types and namespaces. Lastly, the syntax in DTDs is not XML.

XML Schema is more powerful than DTD. Advantages of XML Schema over DTDs are:

- Defines data types for elements and attributes, and their default and fixed values. Some of the data types can be of string, decimal, integer, boolean, date, time or duration. Altogether there are 19 built-in primitive data types and 23 built-in derived data types. Primitive data types are not defined in terms of any other data types, whereas derived data types are defined in terms of other data types.
- Apply restrictions to elements, by stating minimum and maximum values, (for example, on age from 1 to 90 years), or restrictions of certain values (eg. redbooks, residencies, Redpieces with no other values accepted, such as in a drop-down list box). Restrictions can also be applied to types of characters and their patterns (eg. only accepting values ‘a’ to ‘z’ and also specifying that only three letters can be accepted). The length of the data can also be specified. (Eg. passwords must be between 4 and 8 characters.)
- Specify complex element type. Complex types may contain simple elements and other complex types. Restrictions can be applied to the sequence and their frequency of their occurrences. These complex types can then be used in other complex type elements.
- Since schemas are written in XML, they are also extensible. The also implies that the learning curve for learning another language has been eliminated, the available parsers need not be enhanced, transformation can be carried out using XSLT, and also its manipulation can be carried out using XML DOM.
- With XML Schemas being extensible, they can be re-used in other schemas, we can reference multiple schemas from the same document, and also have the ability to create our own data types from standard data types.

2.3 Schema and style using CSS, XSLT, and XSL

The eXtensible Style Language (XSL) is a language defined by the W3C for expressing stylesheets. It has three parts:

- XSL Transformations (XSLT), which is used for transforming XML documents.
- the XML Path Language (XPath), which is a language used to access or refer to parts of an XML document.
- XSL-FO, which is a vocabulary for specifying formatting semantics.

A transformation in XSLT must be well-formed document and must conform to the namespaces in XML, which can contain elements that may or may not be defined by XSLT. XSLT-defined elements belong to a specific XML namespace. A transformation in XSLT is called a stylesheet.

XSL uses a XML notation and works on two principles: pattern matching and templates. It operates on an XML source document and parses it into a source tree, it specifies the tansformation of the source tree to a result tree, and then it outputs the result tree to a specified format. In constructing the result tree, the elements can be reordered or filtered and also other structures can be added. The result tree can be completely different from the source tree.

A stylesheet contains a set of template rules. A rule consists of two parts. The first is a pattern that matches against nodes in the source tree, and the second is a template, which is instantiated to form part of the tree. Therefore, documents that have similar source tree structures can use one stylesheet.

The result tree is constructed by finding the template rule for the root node and instantiating its template. This creates part of the tree. The template, in turn, can contain elements from XSLT namespaces that are instructions for creating parts of the tree. Each instruction in the template is carried out and that instruction would result in a part of the tree being created. When the transformer is looking for a applicable template rule, it may find more than one that matches it for an element. Since only one rule can be applied, conflict resolution rules will be applied. The rules are a combination of import precedence and priority. If the transformer cannot resolve the conflict, it will signal an error.

XSLT makes use of the expressing language defined by XPath for deciding elements for processing and generating text.

XSL transformation can be carried out either on a client or a server. On the client side, Javascript could be used to determine the browser type of any operating system characteristics and then applying different style sheets according to browser and user needs. Obviously, the browser must support an XML parser.

To make the XML data available to all kinds of browsers, we could transform the XML document on the server and send it as a more generic form of HTML to the browser. XSL transformations on the server will experience major growth as the specialized browser market expands. The would include browsers for Braille, aural browsers, Web printers, handheld devices and other kinds.

The W3C has published a working draft for a new set of requirements for XPath on February 2001. It has set of number of goals. It has stated that XML must:

- Simplify manipulation of XML Schema-types content
- Simplify manipulation of string-content
- Support related XML standards
- Improve ease of use
- Improve interoperability
- Improve i18n (International Language Support)
- Maintain backward compatibility
- Enable improved processor efficiency

For more information, visit W3C XSL Transformation (XSLT) Version 1 at:
`http://www.w3.org/TR/xslt#section-Introduction`

W3C Xpath requirements Version 2.0 at:
`http://www.w3.org/TR/xpath20req`

Cascading Style Sheet (CSS)

Cascading Style Sheets were designed to help separate presentation from data with HTML. In the *early days,* Netscape and Microsoft continued to customize their browsers, by adding new tags, to add functionality to the HTML. For developers, it became difficult to create Web sites where the content of the HTML pages was clearly separated from the presentation layout. To alleviate this problem, the W3C created Style Sheets. Style Sheets are usually saved in files external to HTML.

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.

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 shipping browser that supports this feature is Microsoft Internet Explorer 5.0.

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/TR/CSS1`

The Cascading Style Sheet, Level 1 (CSS1) was made a full recommendation in 1996, while Cascading Style Sheet, Level 2 (CSS2) was made a full recommendatory in 1998. CSS2 built upon CSS1, but had made no major changes.

It can be said that CSS1 was a simple specification. It has a few limitations: It did not say anything about tables on a HTML page. CSS2 did try to rectify this, by introducing a new set of properties and behaviors, but these have not been supported.

CSS1 does not incorporate absolute positioning within a table. It is possible to define position relatively. The specification for CSS2 has devoted a number of chapters for visual rendering, which includes the positioning of elements.

W3C has also published a working draft on CSS3 on its Web site in May 2001. The main aim was to modularized the CSS specification. It is intended to clarify the relationships between the different parts of the specification, and to reduce its size.

2.4 XML namespaces

Namespaces are used 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:

Example 2-1 A namespace example

```
<?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>John</authr:firstname>
      <authr:lastname>Smith</authr:lastname>
      <authr:title>Mr</authr:title>
   </authr:author>
 </bk:book>
</library-entry>
```

In the example above, 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. 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. There is not guarantee that the URI will point to a valid URI.

In the 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.

Example 2-2 A namespace example using namespace prefix

```
<?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.

XML namespaces are used extensively in the XML arena, but, during the last two years, there has not been much technological advancement in this area.

2.5 Link and jump using XLink, XPointer, and XML Base

Anyone surfing the Internet knows the joys of moving from one document to another seemlessly. Links are easily embedded within one document to a pre-defined location another. As demands for linking functionality, more demands are required on the technology for more capabilities.

XLink, formerly known as XLL (the eXtensible Linking Language) provide advanced *linking* capabilities, while XPointer provides ways describing *locations* in XML documents. Xpointer is another layer built above XPath and is used to locating data and ranges of data. XBase complements XLink and XPointer by allowing developers to specify a document's base URI. These can use the URI do point to documents using relative parts.

XML documents should have the capability of being easily linked to one another, and these links should be bi-directional. The basic unidirectional link is not enough for future needs. The W3C published a XML Linking Language Version 1 recommendation in June 2001.

In the future, we would need some, if not all of the following capabilities:

- Multi-directional links: In today's technology, we only have unidirectional links. The only way to return to the location of the called document is to select the **Go Back** button. With a multi-directional link, users could return to the original location through a link at the first link's destination.
- Link with multiple locations: There should be choice for different locations or documents from a single link.
- Placing content inline from a linked document: Presently, we cannot present portions of two documents that are interlinked with one another.

The Xlink framework provides for a complex linking structures as will as unidirectional links. Besides specifying the relationships of the two resources to be linked, it can also associate meta data for that link.

The XLink type attribute may have one of the following attributes:

- Title: A description for another linking element
- Simple: A simple link
- Extended: A multi-resource link. An extended attribute can have locator, arc, resource, and title as its child element types.

- Locator: Pointer to a external source. A locator can have a title as its child element type.
- Arc: A rule between resources. An arc element can have a title as its child element type.

The *simple* and *extended* attributes are considered as linking elements. The other attributes describe the link. A *resource* is any available information or service that can be located by one means or another. The links, therefore, can link any two resources: files, documents, images, query results, and programs. A resource can refer to a portion of a resource, and may not have to refer to the whole file or document. A local resource is specified *by value* and a remote resource is specified *by reference*.

A *transversal* is a path from one resource to another. An *arc* describes the traversal and application behavior between two resources. A link is multi-directional if is coded such that the resources switch places at the starting and ending resources. This is not the same as *going back* on the link. An arc is *out-bound* if the starting resources is local and the ending resource is remote. An example would be the HTML *A* element. Conversely, if the ending resource is local, but the starting resource was remote, the arc is said to be *in-bound*. A remote resource is one, which we do not have write access to or one we cannot embed linking constructs.

The arc is said to be *third-party* if both the starting and ending resource are remote. Typically, at any one time, the arc is in-bound, out-bound, or third-party.

Files or documents that contain a collection of in-bound and third-party links are called linkups or link databases.

An example of a simple link is found in Example 2-3.

Example 2-3 A simple link

```
<passengers:FlightReference
  xlink:href="passengerList.xml"
  xlink:role="http://www.airline.com/linkproperties/passengerlist"
  xlink:title="Passenger List">
  xlink:show="replace"
  xlink:actuate="onRequest">
  List of Students for the Flight
</passengers:FlightReference>
```

A simple link in its simplest form provides an outbound link for two resources. This would be very similar to the HTML-style A and IMG links. A click (for the example shown) on the simple link will open the resource pointed to in an existing window.

An example of an XML document showing extended links is in Example 2-4.

Example 2-4 An extended link

```
<PassengerList>
<passengers
    xlink:href="passengers/passengersConfirmed.xml"
    xlink:label="passengersConfirmed020627"
    xlink:role="http://www.airline.com/linkproperties/passConfirmed"
    xlink:title="Confirmed Passengers" />

  <passengers
    xlink:href="passengers/passengersStandBy.xml"
    xlink:label="passengersStandBy020627"
    xlink:role="http://www.airline.com/linkproperties/passStandBy"
    xlink:title="Stand By Passengers" />
  <!-- more remote resources for Passengers, etc. -->

  <flight
    xlink:href="flight/united/Sydney020627.xml"
    xlink:label="UA 0862"
    xlink:title="United Airlines Ua 0862" />
  <!-- more remote resources for courses, seminars, etc. -->

<go
    xlink:from="UA 0862"
    xlink:arcrole="http://www.airline.com/linkproperties/confirmed"
    xlink:to="passengersConfirmed020627"
    xlink:show="new"
    xlink:actuate="onRequest"
    xlink:title="Economy Class Confirmed Passengers" />
  <go
    xlink:from="UA 0862"
    xlink:arcrole="http://www.airline.com/linkproperties/standby"
    xlink:to="passengersStandBy020627"
    xlink:show="replace"
    xlink:actuate="onRequest"
    xlink:title="Economy Class Stand By Passengers" />
</PassengerList>
```

When the entended link has a number of arcs, the specification does not say how the target documents are to be treated. One option would have a pop-up menu that lists all links and the resources found.

For more details, visit W3C XML Linking language (XLink) V1.0 at:
http://www.w3.org/TR/xlink/#N854.

XML Base

XML Base allows developers to specify a document’s base URI. Other links within the same document can then specify links relative to this base. These links could then point to applets, style sheets, images and other files. The syntax for XML Base consists of a single XML attribute named `xml:base`.

A simple example follows for xml:base in a document with Xlink follows.

Example 2-5 An example of extended links using xbase

```
<?xml version="1.0"?>
<doc xml:base="http://airlines.org/tomorrow/"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <head>
    <title>Passenger List</title>
  </head>
  <body>
    <paragraph>Pick <link xlink:type="simple" xlink:href="passengers.xml">your
meal for the flights</link>!!!</paragraph>
    <paragraph>Pick your meals for your flight tomoorw !</paragraph>
    <meallist xml:base="/mealchoices/">
      <item>
        <link xlink:type="simple" xlink:href="choice1.xml">Choice 1: Beef
Vindalooo</link>
      </item>
      <item>
        <link xlink:type="simple" xlink:href="choice22.xml">Choice 2: Chicken
Laksa</link>
      </item>
      <item>
        <link xlink:type="simple" xlink:href="choice3.xml">Choice 3: United
Airlines Burger</link>
      </item>
    </meallist>
  </body>
</doc>
```

The URIs resolving to full URIs would be:

- “Your meal for the flights” resolves to the URI `"http://airlines.org/tomorrow/passengers.xml"`

- “Choice 1: Beef Vindalooo" resolves to the URI `"http://airlines.org/mealchoices/choice1.xml"`
- “Choice 2: Chicken Laksa" resolves to the URI `"http://airlines.org/mealchoices/choice2.xml"`
- "Choice 3: United Airlines Burger" resolves to the URI `"http://airlines.org/mealchoices/choice2.xml"`

As in XML, only Unicode characters are allowed in xml:base. But for the URI, non-ASCII characters, except for the hash, percentage and square brackets, are disallowed. All disallowed characters can be used provided they are converted to UTF-8 or hexadecimal notation of the byte value, or if the character is replaced by the character sequence.

The URI is resolved in the following priority:

1. The URI entry specified in the document
2. The URI of the encapsulating entity
3. The base URI is the URI used to retrieve the entity, or is defined by the application.

   The base URI may consists three parts in this order:

   a. The base URI dictated by the xml:base attribute
   b. The base URI of the element’s parent
   c. The base URI of the document entity containing the element

It is best to provide an xml:base attribute as close to where the relative paths are specified. If not, the xml:base attribute of the parent document comes into affect. If the parent document is not available, then there might by difficult to determine the absolute URI. For best results, the xml:base value should be provided either directly or via default attributes declared in the internal subset of the DTD. Visit W3C Web page to read more:
`http://www.w3.org/TR/xmlbase/`

XML Pointer

The XML Pointer specification is the final part of the XLink specification. Where XLink governs how you can insert links into your XML document and where the link can point to anything, XPointer allows the user to point to specific part of the document or a range, or area of the document. The user can then link the address from one point of the document to another point in the same document.

IN HTML, using the ‘A’ and ‘IMG’ allows you to point from one document to another, and only to a pre-determined spot. In an XML document, given its tree structure, you should be able to navigate down to a child element or to a part of a tree or from one part of a tree to another part of the tree.

The XML Pointer Language (XPointer), the language defined to express fragment identifiers for any URI reference that locates a resource whose Internet media type is one of text/xml, application/xml, text/xml-external-parsed-entity, or application/xml-external-parsed-entity.

Some examples of Xpointers follow. Each of these selects a particular element in a document.

The example finds the element with the ID United:

```
http://www.airlines.com/airline.xml#xpointer(id("United"))
```

The example finds the second language element in the document:

```
http://www.airlines.com/airline.xml#xpointer(descendant::language[position()=2]
)
```

The third example is a shorthand form of finding the element with the ID United.

```
http://www.airlines.com/airline.xml#United
```

The next two examples are more involved. It shows how one navigates down a, say a DOM tree. Here is shows the navigation from the root node, to the 'spec' child, and from it to all the second language elements from any of the child elements:

```
http://www.airlines.com/airline.xml#xpointer(/child::spec/child::body/child::*/
child::language[2])
http://www.airlines.com/airline.xml#xpointer(/spec/body/*/language[2])
```

The last example finds the second child element of the fourteenth child element of the root element:

```
http://www.airlines.com/airline.xml#/1/14/2
```

The final URI also points to the element with the ID United. However, if no such element is present, it then finds the element with the ID UNITED:

```
http://www.airlines.com/airline.xml#xpointer(id("United"))xpointer(id("UNITED")
)
```

XPointer paths and steps

A XPointer path is made up from a number of steps. From a context node, each step will relatively locate a specific point in the document. A location step has three points: axis, node test and an optional predicate. This is in the form:

```
axis::node-test[predicate]
```

For aircraft::ROW[position()=34]:

- The axis is aircraft

- The node test is ROW
- The predicate is [position()=34]

This example located the 34th ROW element along from the content node.

The user can also specify the absolute location steps that do not depend on the context node.

The location path of the XPointer is:

`/child::BOEING747/child::ROW[position()=3].`

This path is built from two location steps:

`/child::BOEING747 and child::ROW[position()>34]`

The first step is an absolute step that selects all child elements of the root node whose name is BOEING747. This should return to exactly one such element. The second step is then applied relative to the BOEING747 element returned by the first step. All of its child nodes are considered. Those that satisfy the node test, that is, elements whose name is ROW are returned. There might be 50 nodes. Therefore, all nodes from 35 to 50 are returned.

XPointer range functions

A range describes a contiguous part of a document. Location paths are identified by location paths. To specify a range, the user appends `/range-to`(end-point) to a location path specifying the start point of the range. An example would be:

```
xpointer(/child::BOEING747/child::ROW[position()=22]/range-to(/child::BOEING747
/child::ROW[position()=last()]))
```

Other range functions are:

`range(location-set)`: Returns a set containing one range for each location specified the argument. It is the minimum range necessary to cover the entire location. In essence, it converts locations to ranges.

`range-inside(location-set)`: Returns a set of locations inside the element with the start and end tags not included. If the input location is a range or point, than it points to that range or point.

`start-point(location-set)`: Returns a set that contains the first point of each location in the input location set. For example, `start-point(//ROW[1])` returns the point immediately after the first `<ROW>` start tag in the document. `start-point(//ROW)` returns the set of points immediately after each `<ROW>` start tag.

end-point(location-set) - acts the same as `start-point()` but returns the points just after each location in its input.

XPointer string functions

There are some basic string matching capabilities through this function:

`string-range()` - It takes a location set to search and a substring to search for. The result is a location set having a single range. The `index` and `length` arguments specify the number of characters where the match should start from and the number of characters it should search for. Strings to be searched for are case sensitive. Markup characters are ignored.

The basic syntax is:

```
string-range(location-set, substring, index, length)
```

The first argument is a location from which the document to be searched for the matching string. The second argument is the actual string to search for. The index argument must be a positive number to start after the beginning of the match. The length argument can specify how many characters to include in the range. If the last two arguments are not specified, then all characters are assumed to require processing.

This example finds all occurrences of the string Boeing767:

```
xpointer(string-range(/,"Boeing767"))
```

The first argument can specify what nodes you want to look in. For example, this finds all occurrences of the string Boeing767 in the `UNITED` elements:

```
xpointer(string-range(//UNITED,"Boeing767"))
```

This example finds only the first occurrence of the string Boeing737 in the document:

```
xpointer(string-range(/,"Boeing737")[position()=1])
```

This final position is immediately before the word Boeing737 in the document's `UNITED` element. This is not the same as pointing at the entire `UNITED` element as an element-based selector would do.

A third argument, which is numeric, targets a particular position within the string. This example targets the point between the I and 'g' in the first occurrence of the string Boeing737 because 'g' is the sixth letter:

```
xpointer(string-range(/,"Boeing737",6)[position()=1])
```

The optional fourth argument specifies the number of characters to select. The example below, selects the ‘737’ from the first occurrence of the entire string Boeing737:

```
xpointer(string-range(/,"Boeing737",7,3)[position()=1])
```

If the first string argument in the node test is empty, then relevant positions in the context node's text contents are selected. In the example below, the first six characters are picked up:

```
xpointer(string-range(/,""1,6)[position()=1])
```

See Related Publications.

2.6 XPath

XPath is to address parts of an XML document. 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 is shown below.

Let us consider an XML document (Library.xml) which describes a Library System. This sample document will be used throughout XPath and XPointer for examples.

Example 2-6 An XPath example

```
<? 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]`

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".

The W3C has published a working draft for a new set of requirements for XPath on February 2001. It has set of number of goals. It has stated that XML must:

- Simplify manipulation of XML Schema-types content
- Simplify manipulation of string-content
- Support related XML standards
- Improve ease of use
- Improve interoperability
- Improve i18n (International Language Support)
- Maintain backward compatibility
- Enable improved processor efficiency

2.7 XML digital

Any document, XML or otherwise, can be encrypted entirely and be transmitted to one or more recipients. Suppose we would only want to encrypt parts of the document of the same document. Also, we would like only different classes of users to have access to different parts of the document. A airline agent may need to know a passenger's customer name and address, but does not need to know the details of their credit card. A passenger boarding officer does not need to have access to the passenger's personal details, while the airline would want to know more information about the passenger for marketing purposes.

It is fairly easy to encrypt a whole document, however, difficulty arises when parts of a document needs to be signed by different people, and when this is to be done with selective encryption.

One of the strengths of XML languages is that searching is clear and unambiguous: The DTD or schema provides information syntax of the XML document. If a document subsection including tags is encrypted as a whole, then we are unable to search for data relevant for those tags. Also, the tags may sometimes need to be hidden, and if they are known, could compromise security.

When sending secure data across the Internet, we need four things:

- Confidentially: No one else can access or copy the data.
- Integrity: The data is not altered as it gets transmitted from the sender to the receiver.
- Authentication: The document actually came from the purported sender.
- Nonrepudiability: The sender cannot deny that they sent it, and the sender also cannot deny the contents of the data.

The first three functions are provided for the Secure Sockets Layer (SSL). The last function is provided for by the XML Security Suite.

The XML Security Suite provides several important functions:

- XML Signatures: This implementation is based on the XML-Signature Core Syntax and Processing specification being developed by W3C and the Internet Engineering Task Force (IETF).
- An implementation of the W3C's Canonical XML working draft
- Element-level encryption

The XML signature and XML encryption are two initiatives designed to both account for and take advantage of the special nature of XML data. These initiatives are currently progressing through the standardization process. The XML Signature initiative is a joint effort between the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF), and XML Encryption is solely W3C effort.

Examples of XML encryption

The main elements in the XML encryption syntax are the `EncryptedData` element and the `EncryptedKey` element, which derive from the `EncryptedType` abstract type. The data to be encrypted can be of an type, being an XML document, element or element content. The result of encrypting data is an XML encryption element that contains or references the cipher data. When an element or element content is encrypted, the `EncryptedData` element replaces the element or content in the encrypted version of the XML document.

When the data is encrypted, the `EncryptedData` element may become the root, or the child of the XML document. When an whole XML document is encrypted, then the `EncryptedData` element may become the root of a new document. However, the `EncryptedData` cannot be the parent or child of another `EncryptedData` element, but the actual data encrypted can be anything including existing `EncryptedData or EncryptedKey` elements.

A simple example follows in Example 2-7.

Example 2-7 Information on passenger John Smith

```
<?xml version='1.0'?>
   <Credit Info xmlns='http://creditOrg.org/bills'>
      <Name>John Smith<Name/>
      <CreditCard Limit='21,000' Currency='AUD'>
         <Credit Card Number>3760 098675 3245</Credit Card Number>
       <Issuer>Wells Rago</Issuer>
      <Expiry date>05/06</Expiry date>
   </CreditCard>
</CreaditInfo>
```

Here we have selectively encrypted John Smith's credit card details. The `EncryptedData` elements now takes the place of the credit card element.

Example 2-8 Encrypted Information on passenger John Smith

```
<?xml version='1.0'?>
<CreditInfo xmlns='http://creditOrg.org/bills'>
    <Name>John Smith<Name/>
        <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
xmlns='http://www.w3.org/2001/04/xmlenc#'>
            <CipherData>
                <CipherValue>W78G12I67</CipherValue>
            </CipherData>
        </EncryptedData>
</CreditInfo>
```

In other cases, other information might be encrypted. Here are some of the credit card details that have been left un-encrypted. This could happen if the credit limit and the currency of the user's details must be left visible to other parties.

Example 2-9 Encrypting only the credit card number

```
<?xml version='1.0'?>
       <CreditInfo xmlns='http://creditOrg.org/bills'>
         <Name>John Smith<Name/>
         <CreditCard Limit='21,000' Currency='AUD'>
           <Number>
             <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
              Type='http://www.w3.org/2001/04/xmlenc#Content'>
                 <CipherData>
                      <CipherValue>P45K98W67</CipherValue>
                 </CipherData>
             </EncryptedData>
           </Number>
           <Issuer>Wells Fargo</Issuer>
           <Expiry date>05/06</Expiry date>
         </CreditCard>
       </CreditInfo>
```

Sometimes, it would be appropriate the encrypt the whole document.

Example 2-10 Encryption of the whole document

```
<?xml version='1.0'?>
<EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
Type='http://www.airline.com/flights/data/media-types/text/xml'>
    <CipherData>
        <CipherValue>I89H56V34</CipherValue>
    </CipherData>
</EncryptedData>
```

The `CipherData` element can either envelop or reference the raw encrypted data. In the first case, that raw data is shown by the contents of the `CipherValue` element, while in the second a `CipherReference` element is used, and this encloses a URI, which points to the location of the encrypted data.

Example of a XML digital signature

The information that is signed is within the `SignedInfo` element. The algorithms used in calculating the `SignatureValue` element are referenced within the signed section, but that element itself is in the `SignatureMethod` element. The `SignatureMethod` references an algorithm used to convert the canonicalized

SignedInfo into the SignatureValue. It is a combination of a key-dependent algorithm and a digest algorithm, here DSA and SHA-1. The KeyInfo element indicates the key used to validate the signature. This element is not mandatory.

Example 2-11 An XML digital signature

```
<Signature Id="UnitedSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
        <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
             <Transforms>
                <Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
             </Transforms>
             <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
             <DigestValue>kgqwequetuwetqwetteuqteuyyey</DigestValue>
          </Reference>
        </SignedInfo>
       <SignatureValue>iu7e876werew776er</SignatureValue>
       <KeyInfo>
       <KeyValue>
          <DSAKeyValue>
             <p>...</p><Q>...</Q><G>...</G><Y>...</Y>
          </DSAKeyValue>
       </KeyValue>
       </KeyInfo>
</Signature>
```

Transforms

When a document or parts of a document are decrypted, we say it is transformed into a decrypted form. The user may need to encrypt parts of a document that already has parts of it that have been encrypted by another user. This user may not be able, or may not need to, decrypt those parts that he has no authority of interest over. The W3C published a candidate recommendation on Decryption Transform for XML Signature in March 2002 that addresses this situation.

In the following example, some data (as in line 11) has already been encrypted, and the user needs to further encypted data of his own.

Example 2-12 Part encryption of an XML document

```
[01]<ticket Id="EXTYGH">
[02]  <passengers>
```

```
    <name>Jenny Smith</name>
    <cost>230.0</cost>
    <seats>4</seats>
  </passengers>
  <cardcard>
    <name>John Smith</name>
    <expirydate>02/2006</expirydate>
    <cardnumber>3760 234567 76547</cardnumber>
  </cardcard>
  <EncryptedData
Id="enc1"xmlns="http://www.w3.org/2001/04/xmlenc#">...</EncryptedData>
</ticket>
```

After this user signs and encrypts this document, it would look like Example 2-13.

Example 2-13 The final XML document after encryption

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    ...
    <Reference URI="#order">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/2001/04/xmlenc#decryption">
          <DataReference URI="#enc1"
xmlns="http://www.w3.org/2001/04/xmlenc#"/>
        </Transform>
        <Transform
Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
      </Transforms>
      ...
    </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
  <Object>
    <ticket Id="EXTYGH">
      <passengers>
        <name>Jenny Smith</name>
        <cost>230.0</cost>
        <seats>4</seats>
      </passengers>
      <EncryptedData Id="enc2" xmlns="http://www.w3.org/2001/04/xmlenc#">
        ...</EncryptedData>
      <EncryptedData Id="enc1" xmlns="http://www.w3.org/2001/04/xmlenc#">
        ...</EncryptedData>
    </ticket>
  </Object>
```

```
[26] </Signature>
```

The signature element now encompasses the whole order, and the ticket elements is now embedded within it (lines 16 to 24). We have now encrypted the credit card details that were between lines 7 to 11 of the previous listing. The Transform element on line 6 to 10, indicates that there are two transform references. The first, decryption (in lines 6 to 8) and canonizations (in line 9). The Decryption Transform, decrypts all the data, except for that on line 7, “enc1”, as specified in the DataReference element. Once this decryption in the EncryptedData element has taken place, the signature is verified. This signature verification information is in the signature value element.

Other security specifications

XML security is still inadequate, and has some way to go before it will be fully accepted. The other specifications that have been raised to address various issues are:

- SAML :Security Assertion Markup Language - "XML security standard for exchanging authentication and authorization information."
- XACML : eXtensible Access Control Markup Language - A language used for define rules and access privileges for XML documents.
- XKMS : W3C’s XML Key Management Specification published in March 2001. This document specifies protocols for distributing and registering public keys

Visit following Web sites to read more details.

W3C Signature Work Group at:
`http://www.w3.org/Signature`

W3C Decryption Transform for XML Signature at:
`http://www.w3.org/TR/xmlenc-decrypt`

Enabling XML Security: An Introduction to XML encryption and XML Signature by Murdoch Mactaggart at:
`http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html`

2.8 XML query language

In February 2001, W3C published a working draft for the XML query language. In April 2002, another working draft was published. These papers had heavy IBM involvement. When the publication is a *working draft,* it can be updated, replaced and made obsolete by other documents at any time. Because it contains many open issues it is considered to be fully stable.

The first draft introduced the XQuery language and provided examples, while the second draft introduced function signatures and other expressions. The second draft exemplified the semantics of element and attributes, and how the underlaying data was operated upon.

The XML query language was specified to be one that made use of XML structure, and have the ability to, “conduct queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware.”

This section of the book does not go into the details of XQuery, but provides examples to show the main flow of the language. This will provide an overview of what XQuery is likely to provide. The examples and explanations that follow are taken from the first draft.

The XQuery language consists of the forms below. Explanations of some are given:

- Path expressions
- Element expressions
- FLWR (“FOR, LET, WHERE, and RETURN clauses”)
- Expressions involving operators and functions
- Conditional expressions
- Quantified expressions
- List constructors
- Expressions that test and modify datatypes

Path expressions

A path expression is a series of steps, where one step of the path may serve as an endpoint, or may have multiple values. Each step may serve as a starting point of the next step, if they exist. Here are some queries expressed in the English form, and then the XQuery form:

In the second chapter of the document named “zoo.xml”, find the figure(s) with caption “Tree Frogs”.

```
document("zoo.xml")/chapter[2]//figure[caption = "Tree Frogs"]
```

In this example, the first step finds the document, the second locates Chapter 2, and the last step filters all captions in that chapter for “Tree Frogs”.

Find all the figures in Chapters 2 to 5 of the document named "zoo.xml."

```
document("zoo.xml")/chapter[RANGE 2 TO 5]//figure
```

This example demonstrates use of the RANGE predicate. The first element always has a ordinal number of 1.

Find captions of figures that are referenced by <figref> elements in the chapter of "zoo.xml" with title "Frogs".

```
document("zoo.xml")/chapter[title = "Frogs"] //figref/@refid->fig/caption
```

XQuery has allowed for a de-reference operator("->"). In HTML, IDREF and IDREFS values refer to values of other elements' ID attributes. An IDREF value is a single ID while an IDREFS value is a space-separated list of IDs. IDREF and IDREFS are case-sensitive. In this case, the deference operator follows the IDREF-type attribute, and returns elements referenced by the attribute. In the example above, the de-reference operator locates, in the "figref" element, in the "refid" attribute, the caption of the "fig" element.

List the names of the second-level managers of all employees whose rating is "Poor".

```
/emp[rating = "Poor"]/@mgr->emp/@mgr->emp/name
```

Here the query locates, where the employees having a "poor" rating, the name of the employees by looking up another emp element having a "mgr" attribute.

In the document "zoo.xml", find <tiger> elements in the namespace defined by `www.abc.com/names` that contain any supplement in the namespace defined by `www.xyz.com/names:`

```
NAMESPACE abc = "www.abc.com/names"
NAMESPACE xyz = "www.xyz.com/names"
document("zoo.xml")//abc:tiger[xyz:*]
```

Here query provides syntax for URIs. A default namespace can also be specified as in this last example:

```
NAMESPACE DEFAULT = "www.abc.com/names"
NAMESPACE xyz = "www.xyz.com/names"
document("zoo.xml")//tiger[xyz:*]
```

Element constructors

The typical use of an element constructor is that it is nested inside another expression that binds one or more variables.

Generate an <emp> element containing an "empid" attribute and nested <name> and <job> elements. The values of the attribute and nested elements are specified by variables that are bound in other parts of the query:

```
<emp empid = $id>
   <name> $n </name> ,
   <job> $j </job>
</emp>
```

Generate an element with a computed name, containing nested elements named <description> and <price>:

```
<$tagname>
   <description> $d </description> ,
   <price> $p </price>
</$tagname>
```

In this example, the tagname, with its end-tag itself is a variable.

FLWR(“FOR, LET, WHERE and RETURN clauses”) expression

A FLWR expression binds values to and more variables, and then uses these variables to construct a result. For the following example, the XML file would follow the structure shown:

```
<book>
  <title>...</title>
  <author>.....</author>
  <author>....</author>
  <publisher>...</publisher>
  <price>.......<price>
  <year>....<year>
</book>
```

List the titles of books published by Morgan Kaufmann in 1998:

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann"
AND $b/year = "1998"
RETURN $b/title
```

List each publisher and the average price of its books:

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")
   /book[publisher = $p]/price)
RETURN
   <publisher>
      <name> $p/text() </name> ,
      <avgprice> $a </avgprice>
   </publisher>
```

Conditional expressions (IF THEN ELSE)

Conditional expressions are used when the information returned from a query depends on some condition.

Make a list of holdings, ordered by title. For journals, include the editor; and for all other holdings, include the author:

```
FOR $h IN //holding
RETURN
   <holding>
      $h/title,
      IF $h/@type = "Journal"
      THEN $h/editor
      ELSE $h/author
   </holding> SORTBY (title)
```

Quantifiers

Sometimes we have to test if elements exist that must satisfy a condition.

Find titles of books in which both sailing and windsurfing are mentioned in the same paragraph:

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
   contains($p, "sailing")
   AND contains($p, "windsurfing")
RETURN $b/title
Eg 12. Find titles of books in which sailing is mentioned in every
paragraph.
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
   contains($p, "sailing")
RETURN $b/title
```

Filtering

This filter function takes two operands, each of which is an expression that evaluates to a ordered set of notes, and returns copies of some of the nodes in the first operand, while preserving their hierarchic and sequential relationships. The second operand is the *filter* that trims the hierarchical tree. Both operands must have the same node, not only two nodes of the same value. If the two operands do have a common root, the result of the filter function is an empty list.

The following example illustrates this process by computing a table of contents for a document, which contains many levels of nested sections. The query filters the document and retaining only section elements, title elements nested directly inside section elements, and the text of those title elements. Other elements, such as paragraphs and figure titles, are eliminated, leaving only the *skeleton* of the document.

The first argument of filter is the root of a document, and the second argument is a path expression that identifies the nodes to be preserved from the original document.

Prepare a table of contents for the document "cookbook.xml", containing nested sections and their titles:

```
LET $b := document("cookbook.xml")
RETURN
   <toc>
      filter($b, $b//section | $b//section/title | $b//section/title/text() )
   </toc>
```

Querying relational data

The best way to demonstrate XQuery is showing how it can be used to query a relational database. In Table 2-1 we have a very simple database.

Table 2-1 Example schema for relational database and Xquery comparison

Table name	Relational data	XML representation
S	SNO SNAME	<s> <s_tuple> <sno> <sname>
P	PNO DESCRIP	<p> <p_tuple> <pno> <descrip>
SP	SNO PNO PRICE	<sp> <sp_tuple> <sno> <pno> <price>

SQL is the standard relational database language. In many cases, SQL queries can be converted to XQuery syntax in a straightforward way by mapping SQL query-blocks into FLWR-expressions. We illustrate this mapping by the following query:

Find the part numbers of gears in numeric order:

```
SQL version:
SELECT pno
FROM p
WHERE descrip LIKE 'Gear'
ORDER BY pno;
```

XQuery version:

```
FOR $p IN document("p.xml")//p_tuple
WHERE contains($p/descrip, "Gear")
```

```
RETURN $p/pno SORTBY(.)
```

Grouping in XQuery

Find the part number and average price for parts that have at least three suppliers.

SQL version:

```
SELECT pno, avg(price) AS avgprice
FROM sp
GROUP BY pno
HAVING count(*) >= 3
ORDER BY pno;
```

XQuery version:

```
FOR $pn IN distinct(document("sp.xml")//pno)
LET $sp := document("sp.xml")//sp_tuple[pno = $pn]
WHERE count($sp) >= 3
RETURN
   <well_supplied_item>
      $pn,
      <avgprice> avg($sp/price) </avgprice>
   </well_supplied_item> SORTBY(pno)
```

The `$pn` represents an individual part number, but `$sp` represents a set of records.

Joins

Joins combine data from multiple sources. We will present a simple example with an inner join.

Return a "flat" list of supplier names and their part descriptions, in alphabetic order:

```
FOR $sp IN document("sp.xml")//sp_tuple,
    $p IN document("p.xml")//p_tuple[pno = $sp/pno],
    $s IN document("s.xml")//s_tuple[sno = $sp/sno]
RETURN
   <sp_pair>
      $s/sname ,
      $p/descrip
   </sp_pair> SORTBY (sname, descrip)
```

In conclusion, XQuery is designed to support many types of queries. The more versatile the Xquery is, the more it will provide usability for XML for applications.

For more detail, visit XQuery 1.0: An XML Query Language (April 2002) at:
`http://www.w3.org/TR/xquery/`

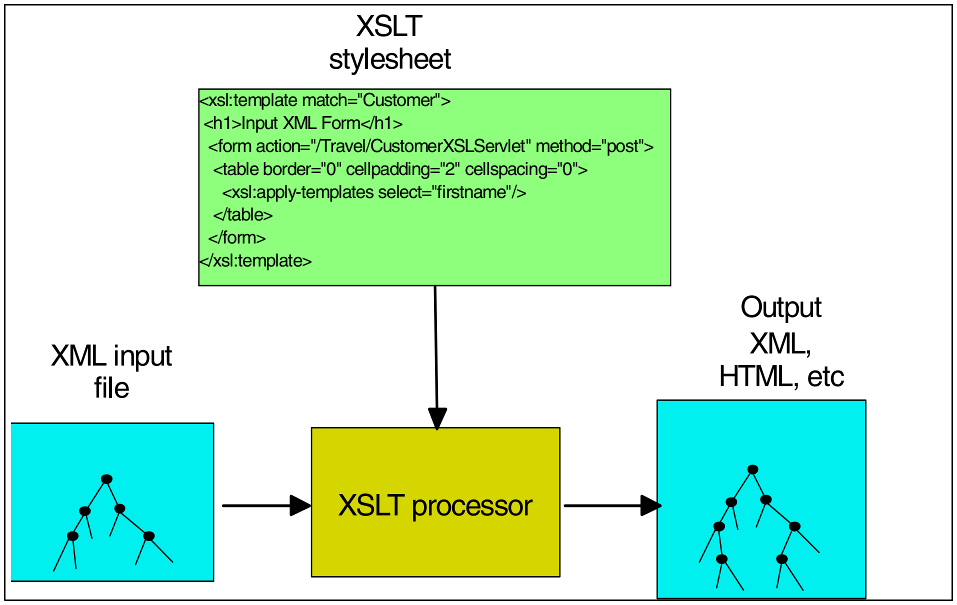
XQuery: A Query Language for XML (Feb 2002) at:
`http://www.w3.org/TR/2001/WD-xquery-20010215/#section-Introduction`

2.9 XSLT compilers (XSLTC)

As the use of XML documents climbs, so will the use of XSL transformations increase. For any project, this increase will not only be in terms of the number of transformations required, but also in the size of each transformation. There are a number of XSLT engines on the market today, but these are all interpreters (Figure 2-1).

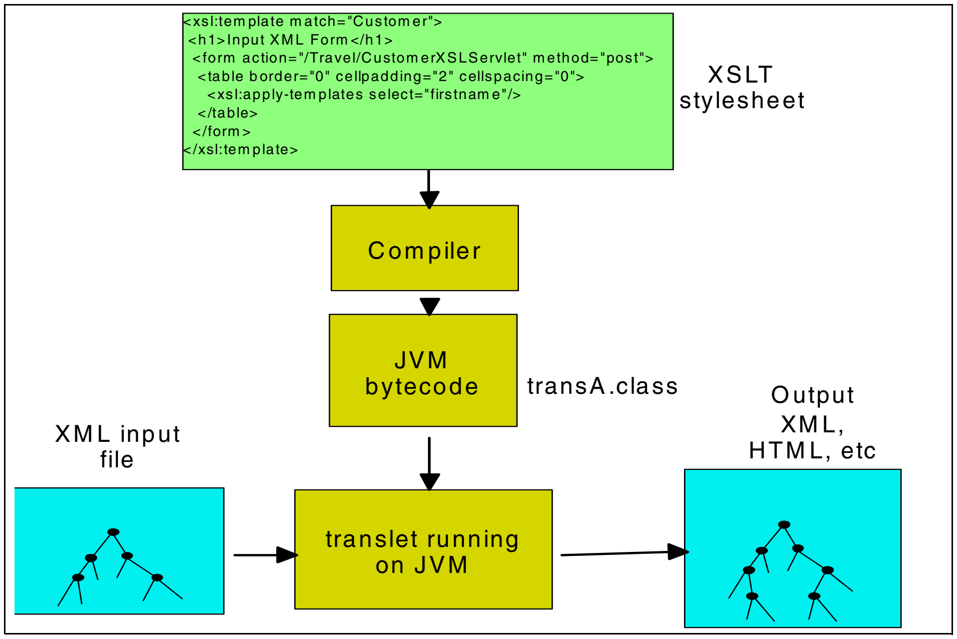
Transformations are generally static and can be used multiple times over different XML documents. If these transformations could be pre-compiled, we would realize performance levels in speed and for memory (Figure 2-2).

Java classes that offer translation capabilities are called translets. Translets can offer significant speed in server-side transformations, and also the possibility of client-side transformation of XML into other formats. Translets are expected to be small, in the 100 Kilobytes range.



```
<xsl:template match="Customer">
 <h1>Input XML Form</h1>
  <form action="/Travel/CustomerXSLServlet" method="post">
   <table border="0" cellpadding="2" cellspacing="0">
     <xsl:apply-templates select="firstname"/>
   </table>
  </form>
</xsl:template>
```

Figure 2-1 XSLT interpreters



```
<xsl:template match="Customer">
 <h1>Input XML Form</h1>
  <form action="/Travel/CustomerXSLServlet" method="post">
   <table border="0" cellpadding="2" cellspacing="0">
     <xsl:apply-templates select="firstname"/>
   </table>
  </form>
</xsl:template>
```

Figure 2-2 The XSLT compiler

The Apache XML project (xml.apache.org) provides XSLTC, which is a compiler and runtime processor. The compiler can be used to compile an XSL stylesheet into a set of Java classes called *translets*. The classes than can be deployed, just like any other Java classes, along with the xsltc.jar and the runtime.jar. The compiler and the translets can be run from the command line, and also the TrAX/JAXP API. TrAX provides a framework and a standard API for performing XML transformation, and it uses systems properties to locate the Transformer and the XML parser.

The JAXP API can be found in the Java XML Pack, which can be downloaded from the Sun Web site. Besides the two methods, the XSLTC can also be used with the native Java API.

Xalan-Java Version 2 is an XSLT processor built upon on SAX 2 and DOM 2. It implements the W3C recommendations for XSLT and the XMl Path language (XPath). It can be used from the command line, in a Java class (applet or servlet), or as a module in another program.

For more details, visit *Using XSLTC:The Apache XML Project* at:
`http://xml.apache.org/xalan-j/xsltc_usage.html`

Xalan-Java version 2.4.D1: The Apache XML project at:
`http://xml.apache.org/xalan-j/index.html`

2.10 Java Architecture for XML Binding (JAXB)

The Java Architecture for XML Binding provides an API and tools that automate the mapping between XML documents and Java objects. These are available as an early access release. The public version of the JAXB specifications and a pre-release version of the reference implementation will be available by the third quarter 2002. The final draft is scheduled to be published in the forth quarter of the same year.

The JAXB API compiles an XML Schema into Java classes. The Java classes handle all the details of XML parsing and formatting. The generated classes ensure that the constraints expressed in the DTD are enforced in the methods and Java technology language data types. The early access release only works on DTD, but later versions are being produced to support other schemes.

The classes generated by JAXB perform at a greater efficiency than the SAX and DOM parsers, because the classes are more customized than the generic parsers. The generated application validates structure and content with the help of the Java classes. The structure and content validation classes perform better than the SAX, because the derived classes are precompiled. Unlike the DOM parser, it does not need to create the whole tree. The content tree created is specific to one schema, and the classes do not need any redundant tree-manipulation functionality.

The JAXB expert group is now working on the new specification. Among the features are:

- Support for a subset of W3C XML Schema and XML namespaces
- More flexible unmarshalling and marshalling functionality
- Enhanced validation capabilities, such as allowing validation to be turned on for development purposes, and turned off for deployment.

To understand JAXB in depth, visit Java Architecture for XML Binding (JAXB) at Java.sun.com:
`http://www.java.sun.com/xml/jaxb/`

2.11 Cocoon

Cocoon provides an XML publishing framework that is designed around the SAX API. It can interact with common data sources, including files, relational databases, and XML databases. The content delivery can be customized to different devices such as HTML, WML, RTF, PDF and others. Its main aim is to provide a platform for building applications with distinct separation between content, logic, and presentation.

The latest version is Cocoon 2 and is available for download from:
`http://www. xml.apache.org/cocoon/`

It is a Web based application and must be run upon a Java Servlet 2.2 compliant engine, and also Jakarta Tomcat 4.0.1

In Cocoon, everything can be perceived as part of a pipeline. A pipeline consists of an input, some processing steps and then an output. Cocoon 2 makes use of the SAX events between each processing step. Parts of the pipeline include a generator and a serializer. The generator is used for input, the serializer produces output and the transformer is used for processing intermediate steps.

These components can be grouped together into several distinct types, depending on the roles they play within a pipeline:

Generators and reader: pipeline inputs

Generators are used for inputing data source and passing it on as a series of SAX events. The simplest generator is therefore a SAX parser. Generally, any data source that can be mapped to a series of SAX events can become the basis for a generator.

Transformers and actions: processing steps

The main processing steps in the pipeline are handled by the transformers (Figure 2-3). They accept SAX events as input, perform a number of useful processing steps, and then the results produced are passed down the pipeline as SAX events. The most common transformer one can find is the XSLT Transformer. Input is fed into an XSLT processor which performs an XSLT transformation. The results of the transform are then fed back into the pipeline as SAX events.

Serializer: pipeline outputs

Serializers render the output from a stream of SAX events produced from a generator or a transformer. The format of the output is dependent on the serializer specified. *Serializers* are the endpoints in Cocoon pipelines. The most basic serializer is the XML serializer, which simply turns the SAX events back into an XML document, while other serializer produce HTML, PDF documents, and images to name a few. The serializers need to have the SAX events stream to be of a particular XML vocabulary: A HTML Serializer turns XHTML into valid HTML, a PNG images and PDF Serializer: Turns XSL-FO into a PDF document and a SVG Serializer turns SVG into JPEG.

Matchers and selectors: conditional processing

Conditional statements are part of any programming language. Output can be dependent on many factors, such as the users' browser, request parameters, or operating system.

Matchers are the equivalent of If statements. If a condition is true, then a section of a pipeline is processed, or otherwise is passed by.

A selector is used when there are few options available, and it is similar to an If-then-else statement. These are used to create conditional sections in the pipeline, while matchers are used to test if a particular pipeline is to be processed.

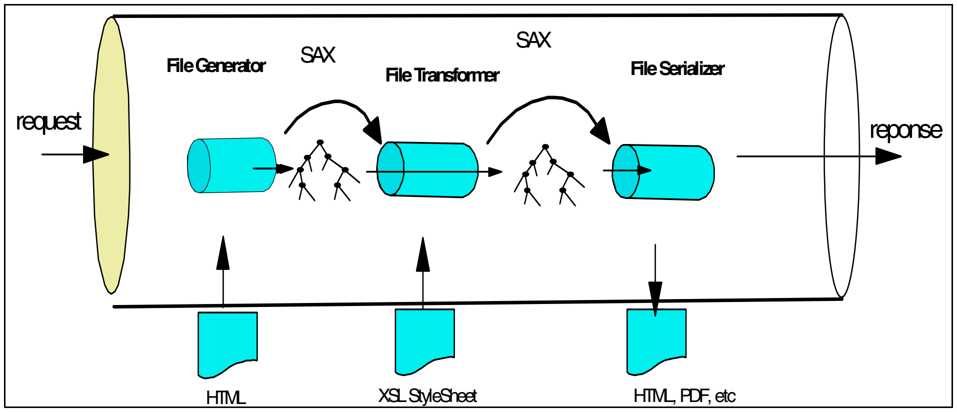


Figure 2-3 Cocoon components

Sitemap

The Cocoon sitemap fulfills two functions:

- Components (matchers, generators, serializers, transformers, etc.) are declared here before being used in pipelines.
- Where pipelines are declared using the declared components.

It consists of configuration data for the Cocoon engine.

Example 2-14 Sample sitemap for Cocoon

```
<map:match pattern="airline.html"">
   <map:generate src="docs/flight_data.xml"/>
   <map:transform src=stylesheets/page/airline.xsl"/>
   <map:serialize type="html"/>
</map:match>
```

The sitemap is a XML file and is responsible for declaring individual components. It is used to define how those components are used to construct pipelines. Declaring components within the sitemap provides Cocoon with a great deal of extensibility, allowing the plug-and-play addition of new implementations.

The sitemap XML file has the structure shown below:

Example 2-15 Components of a sitemap XML file

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <map:components>
   <!-- component declared here -->
   <map:generators/>
   <map:readers/>
   <map:transformers/>
   <map:actions/>
   <map:serializers/>
   <map:actions/>
   <map:matchers/>
   <map:selectors/>
</map:components>
<map:pipelines>
   <!-- pipeline definitions declared here-->
   </map:pipelines>
</map:sitemap>
```

It refers to a specific namespace: ‘`"http://apache.org/cocoon/sitemap/1.0"`, which is used to identify all the elements. As mentioned earlier, the file is divided into two sections: `map:components` and `map: pipelines`, reflecting its responsibilities.

This section of this chapter has only served as an introduction to Cocoon. Any further discussions would have involved starting a new chapter. Like any other emerging technology, it tries to compartmentalize data from logic and presentation. Its further acceptance will depend on how much it can keep up with the ever changing requirements of rapidly changing industry.

Part 2

XML technology in IBM WebSphere

This part introduces IBM WebSphere Studio family and introduces the XML development capabilities of Websphere Studio Application Developer using XML perspective and wizards.

3

Processing XML

Chapter three provides the reader with an opportunity to learn about the processing of XML documents. The chapter highlights the XML Processor and parser shipped with WebSphere Application Developer.

In this chapter, the following topics are described:

- Xalan XSLT Processor
- SAX2
- DOM Level2
- Java API for XML Processing (JAXP)

3.1 XML applications

Today’s organizations’ rapid movement towards e-business brings new demands on defining flexible systems architectures. Systems need to be powerful, scalable, robust, and most of all, capable of meeting new business requirements. With that in mind, applications often need to be able to support multiple client types, all with different capabilities. The dominant client type for Web applications is currently the desktop browser, but that will not last forever. Pervasive Computing is rapidly evolving, introducing the use of cellular phones, PDAs, and other front-end devices, all with different XML capabilities. So as time passes by, each application gets a much broader audience, using a variety of new devices for sending and receiving information.

The most recent edition of any browser that might have XML support cannot be a prerequisite for using an XML based 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.

Extensible Stylesheet Language Transformations (XSLT) is designed to transform XML data into some other XML form. An XSLT processor, such as Apache’s Xalan, performs transformations using one or more XSLT stylesheets, which are also XML documents. Typically, in an XSLT- and Java-based Web application, XML data is generated dynamically based on database queries. Although some databases can export data directly as XML, you will often write custom Java code to extract data using JDBC and convert it into XML. In order to display this XML data on most browsers, it must first be converted into HTML. The XML data is fed into the processor as one input, and as XSLT stylesheet is provided as a second input. The output is then sent directly to the Web browser as a stream of HTML. The XSLT stylesheet produces HTML formatting instructions, while the XML provides raw data.

So in conclusion, at the heart of every XML application is an XML Processor that parses an XML document, so that the document elements can be retrieved and transformed into a presentation understood by the target client. The other responsibility of the parser is to check the syntax and structure of the XML document. The focus of this chapter is on the Xalan Java processor with all the aspects involved in processing an XML document.

3.2 Xalan

Xalan is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath). It provides a standard API for performing XML Transformations. It builds on SAX2, DOM Level2, and the XML parser API in Java API for XML Parsing 1.0 (JAXP). It may be configured to work with any XML parser, such as Xerces, that implements JAXP. It can process Stream, SAX or DOM input, and output to a Stream, SAX or DOM.

The diagram in Figure 3-1 shows a high level model of operation. A transformation expressed in XSLT describes the rules for transforming a source tree into a result tree. A term tree represents the structure of an XML document, whether it is a DOM tree, or a series of parse events coming from a SAX2 ContentHandler. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered.

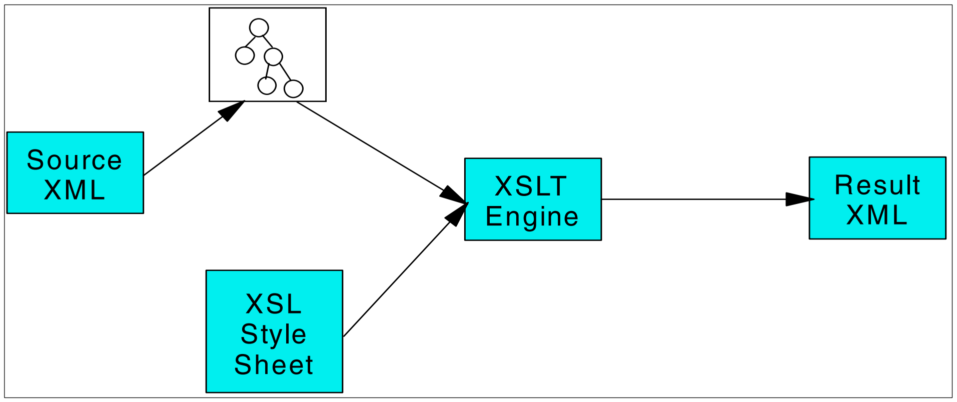


Figure 3-1 Xalan model of operation

The primary interface for Xalan for external usage is javax.xml.transform. These interfaces define a standard and powerful interface to perform tree-based transformations. These interfaces have no dependencies on SAX or the DOM standard as shown in Figure 3-2. It achieves this by defining source and result interfaces, which a user can use to define instances of whatever input and output desired, whether SAX, DOM, or stream.

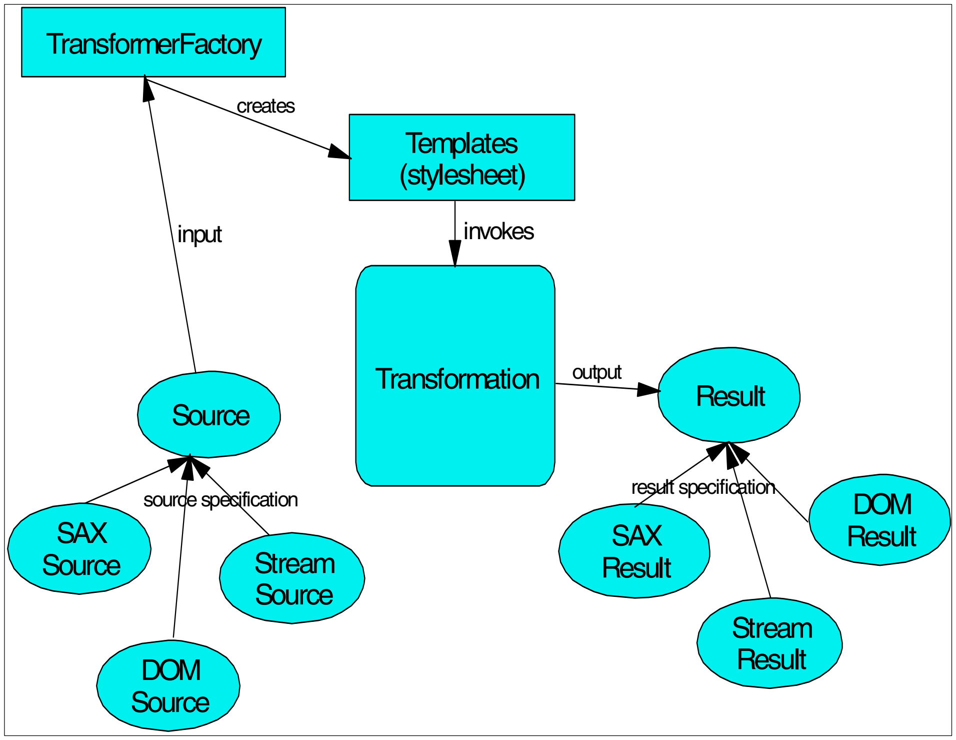


Figure 3-2 javax.xml.transforms interfaces operation

3.3 SAX2

SAX is the Simple API for XML, originally a Java-only API. SAX was the first wisely adopted API for XML in Java. SAX APIs are event-based APIs, which report parsing events (such as start and end elements) directly to the application through callbacks, and does not usually build an internal tree. These event-driven APIs are used for accessing XML documents and extracting information from them. They cannot be used to manipulate the internal structures of XML documents. As the XML document is parsed, the application using SAX receives information about the various parsing events. The application implements handlers to deal with these different events, much like handling events in a graphical user interface. The logical structure of an application using SAX API with the parser is shown in Figure 3-3. You can parse documents much larger than your available system memory, and you can construct your own data structures using your callback event handlers.

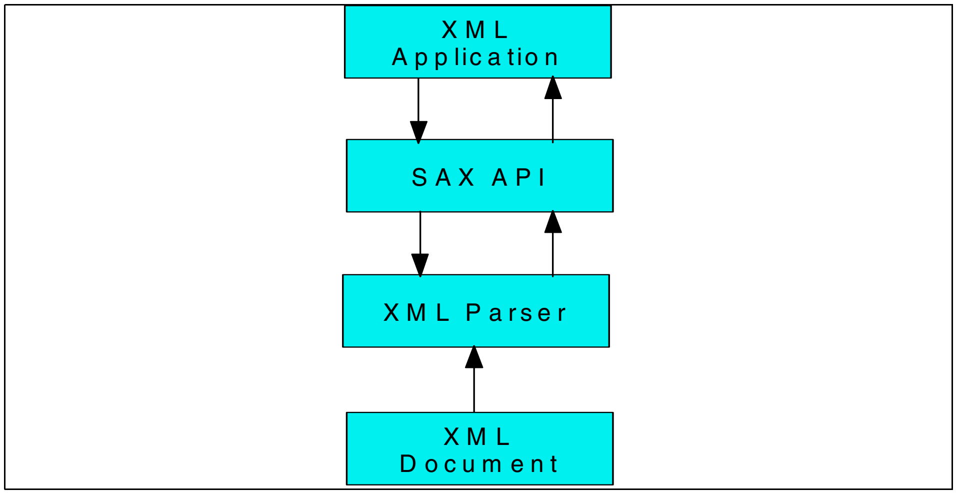


Figure 3-3 SAX application components

To understand how an event-based API can work, consider the sample document in Example 3-1.

Example 3-1 Sample XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<Customer>
    <name>Mary Smith</name>
    <membership>10001</membership>
</Customer>
```

An event-based interface will break the structure of this document down into a series of linear events, such as those shown in Example 3-2.

Example 3-2 Sample event breakdown

```
start document
start element: Customer
start element: name
characters: Mary Smith
end element: name
start element: membership
characters: 10001
end element: membership
end element: Customer
end document
```

An application handles these events just as it would handle events from a graphical user interface; there is no need to cache the entire document in memory or secondary storage.

Currently, there are two versions of SAX: 1.0 and 2.0. Many changes were made in version 2.0. The focus of this chapter is on SAX version 2.0. Most SAX parsers should support the older 1.0 classes and interfaces, however, you will receive deprecation warnings from the Java complier if you use these older features.

SAX2 classes and interfaces

The following interfaces and classes are provided by SAX2:

- org.xml.sax.XMLReader which replaces SAX1 Parser
- org.XML.sax.XMLFilter
- org.xml.sax.ContentHandler which replaces SAX1 DocumentHandler
- org.xml.sax.Attributes which replaces SAX1 AttributeList
- org.xml.sax.SAXNotSupportedException
- org.xml.sax.SAXNotRecognizedException
- org.xml.sax.helpers.AttributesImpl which replaces SAX1 AttributeListImp
- org.xml.sax.helpers.NamespaceSupport
- org.xml.sax.helpers.XMLFilterImpl
- org.xml.sax.helpers.ParserAdapter
- org.xml.sax.helpers.XMLReaderAdapter
- org.xml.sax.helpers.DefaultHandler which replaces SAX1 HandlerBase

SAX2 contains complete namespace support, which is available by default from any XMLReader. AN XML reader can also optionally supply raw XML 1.0 names. Have a look at “XML namespace support” on page 61 for more details about SAX2 support for XML namespaces.

The ContentHandler and Attribute interfaces are similar to the deprecated DocumentHandler and AttributeList interfaces, but they add support for the namespace related information. ContentHandler also adds a callback for skipped entities, and Attributes adds the ability to look up an attribute’s index by name.

The ContentHandler interface is regarded as the most important of the all, as it has methods such as startDocument(), startElement(), characters(), endElement(), and endDocument(). During the parsing process, startDocument() is called once, then startElement() and endElement() are called once for each tag in the XML data. The characters() method provides the text value of the element. This process continues until the end of the document, at which time endDocument() is called.

Since ContentHandler is an interface, it is up to your application code to somehow implement this interface and subsequently do something when the parser invokes its methods. SAX does provide a class called DefaultHandler that implements the ContentHandler interface. To use DefaultHandler, create a subclass and override the methods that interest you. The other methods can safely be ignored, since they are just empty methods.

The ParserAdapter class makes a SAX1 Parser behave as a SAX2 XMLReader. The XMLReaderAdapter class makes a SAX2 XML reader behave as a SAX1 parser. The two classes should ease the transition from SAX1 to SAX2 by allowing SAX1 drivers and clients to co-exist with SAX2 drivers and clients in the same application.

XML namespace support

SAX2 adds XML namespace support. Every implementation of the SAX2 XMLReader interface is required to support namespace processing in its default state. Additionally, many XML readers allow namespace processing to be modified or disabled. Namespace processing changes only element and attribute naming, although it places restrictions on some other names. Each XML element and attribute has a single name called the qName which may contain colons. With namespaces, elements and attributes have two-part name, sometimes called the universal or expanded name, which consists of a URI, and a localName.

SAX2 is capable of supporting either of these views or both simultaneously. Similarly, documents may use both views simultaneously. SAX2 XMLReader implementations are required to report the namespace style names when documents use them.

Namespace support affects the ContentHandler and Attributes interfaces. In SAX2, the `startElement` and `endElement` callbacks in a content handler look like Example 3-3 on page 61.

Example 3-3 SAX2 callback methods

```
public void startElement (String uri, String localName,String qName,
                Attributes atts) throws SAXException;
public void endElement (String uri, String localName, String qName)
                                                    throws SAXException;
```

By default, an XML reader will report a namespace URI and a local name for every element that belongs to a namespace, in both the start and end handler. Consider Example 3-4 on page 61.

Example 3-4 Sample namespace

```
<html:hr xmlns:html="http://www.ibm.com"/>
```

With the default SAX2 namespace processing, the XML reader would report a start and end element event with the namespace URI `http://www.ibm.com` and the local name `hr`. Most XMLReader implementations also report the original qName `html:hr`, but that parameter might simply be an empty string (except for elements that are not in the namespace).

For attributes, you can look up the value of a named attribute using the `getValue` method, and you can look up the namespace URI or local name of an attribute by its index using the `getURI` and `getLocalName` methods. If the URI were the empty string, you would normally use the qName to identify the attributes.

In addition to those events, SAX2 reports the scope of namespace declarations using the `startPrefixMapping` and `endPrefixMapping` methods, so that applications can resolve prefixes in attribute values or character data if necessary.

3.4 DOM level2

While XML is a language to describe tree-structures data, the Document Object Model (DOM) defines a set of interfaces to access tree-structures 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.

DOM level2, 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. The DOM Core API allows the creation and population of a Document object using only DOM API calls; loading a Document and saving it persistently is left to the product that implements the DOM API.

DOM hierarchy

The DOM provides a set of standard object interfaces that an XML parser can use to expose the contents of a document to a client application. These interfaces provide access to all the information from the original document, organized in a hierarchical tree structure. The base interface for navigating this tree structure is the `Node` interface, 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. Every specific document structure is represented in the DOM by one of the following specialized interfaces:

- Document

- Attr
- Element
- Text
- Comment
- CDATASection
- DocumentType
- Notation
- Entity
- EntityReference
- ProcessingInstruction

These specialized interfaces all inherit the basic attributes and methods provided by the `Node` interface. They also provide specialized access to unique information associated with each specific XML document item. The resulting specialized nodes are stored in a list of lists structure that has parent_child and sibling-to-sibling links. For example, the following document in Example 3-5 would produce the tree of DOM nodes in memory shown in Figure 3-4 on page 64. The structure can be traversed using the parent, child, and sibling links available through the node interface.

Example 3-5 Sample XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<Customer>
    <name>Mary Smith</name>
    <membership>10001</membership>
</Customer>
<Customer>
    <name>Dave Johnson</name>
    <membership>12345</membership>
</Customer>
```

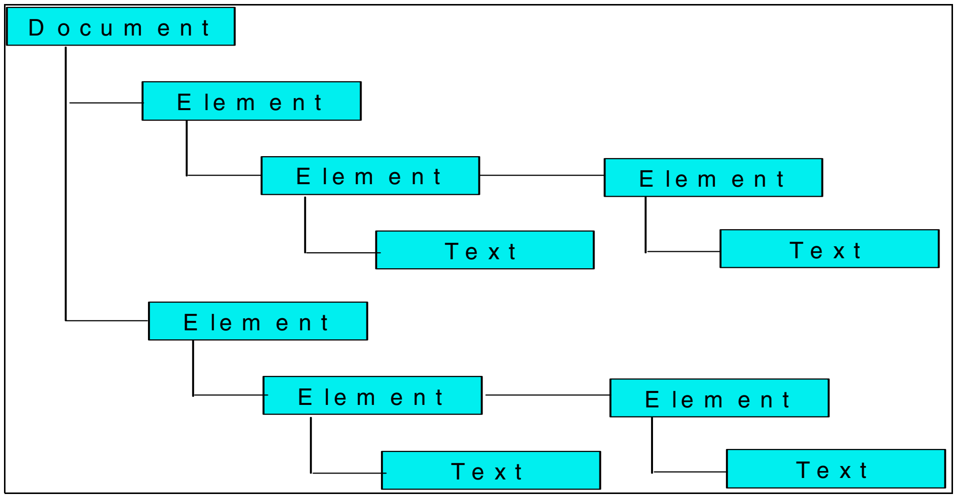


Figure 3-4 Sample generated DOM tree

`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. It should be noticed that while a DOM application reads an XML document and an object representation is 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.

XML namespace support

The DOM Level2 supports XML namespace. It allows creating and manipulating elements and attributes associated to a namespace. As far as the DOM is concerned, special attributes used for declaring XML namespaces are exposed and can be manipulated just like any other attribute. However, nodes are permanently bound to namespace URIs as they get created. Consequently, moving a node within a document, using the DOM, in no case results in a change of its namespace prefix or namespace URI. Similarly, creating a node with a namespace prefix and namespace URI, or changing the namespace prefix of a node, does not result in any addition, removal, or modification of any special attributes declaring the appropriate XML namespaces. Namespace validation is not enforced; the DOM application is responsible.

DOM Level2 does not perform any URI normalization. The URIs given to the DOM are assumed to be valid. Absolute URI references are treated as strings and compared literally. How relative namespace URI references are treated is undefined. To ensure inter operability, only absolute namespace URI references should be used. Note that the empty string will be treated as a real namespace

URI in DOM Level2 methods. Applications must use the value null as the namespace URI parameter for methods if they wish to have no namespace.

3.5 JAXP

The Java API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX, and XSLT. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation. Depending on the needs of the application, developers have the flexibility to swap between XML Processors (such as high performance vs. memory conservative parsers) without making application code changes. Thus, application and tools developers can rapidly and easily XML-enable their Java applications for e-commerce, application integration, and dynamic Web publishing. Just added into the JAXP 1.2 reference implementation is support for XML Schema and an XML compiler (XSLTC).

First released in March 2000, Sun's JAXP 1.0 utilized XML 1.0, XML Namespaces 1.0, SAX 1.0, and DOM Level 1. JAXP is a standard extension to Java, meaning that SUn provides a specification through its Java Community Process (JCP) as well as a reference implementation. JAXP 1.1 follows the same basic design philosophies of JAXP 1.0, adding support for DOM Level2, SAX2, and XSLT 1.0. A tool like JAXP is necessary because the XSLT specification defines only a transformation language; it says nothing about how to write a Java XSLT processor. Although they all perform the same basic tasks, every processor uses a different API and has its own set of programming conventions.

So simply JAXP is an API, but it is more accurately an abstraction layer. It does not provide a new means for parsing XML, add to SAX or DOM, or provide new functionary to Java and XML handling. Instead, it makes it easier to deal with some difficult tasks with DOM and SAX. Without SAX, DOM, or another XML parsing API, you cannot parse XML.

The key to JAXP's design is the concept of plug-ability. Figure 3-5 on page 66 illustrates the high-level architecture of JAXP. These layers provide consistent Java interfaces to the underlying SAX, DOM, and XSLT implementations. In order to utilize one of these APIs, you must obtain a factory class without hard-coding Xalan or any other processor code in your application. This is achieved via a lookup mechanism that relies on Java system properties. Since three separate plug-ability layers are used, you can use a DOM parser from one vendor, a SAX parser from another vendor, and yet another XSLT processor from someone else. In reality, you will probably need to use a DOM parser compatible with your XSLT processor if you try to transform the DOM tree directly.

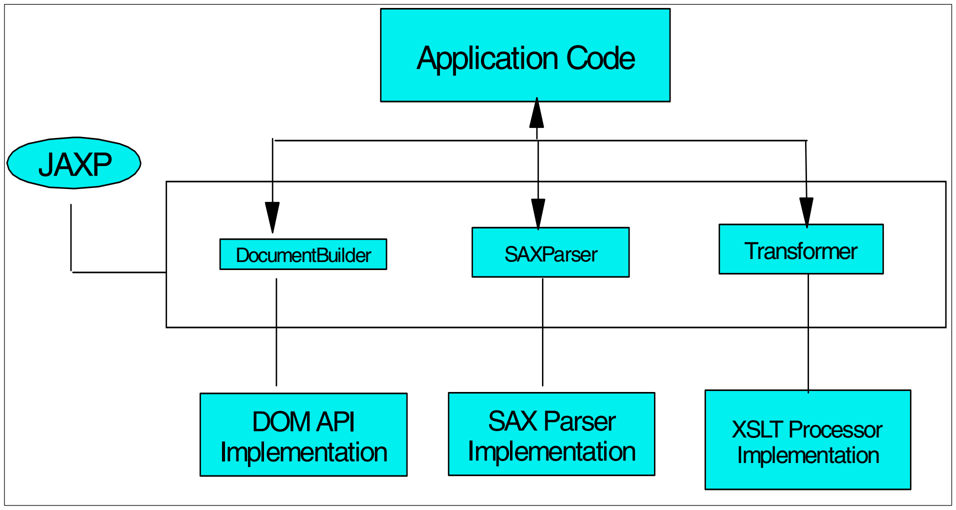


Figure 3-5 JAXP architecture

As shown, application code does not deal directly with specific parser or processor implementations. Instead, you write code against abstract classes that JAXP provides. This level of indirection allows you to pick and choose among different implementations without even recompiling your application.

Using JAXP

In this section, we show a very simple example that illustrates the usage of JAXP to invoke the process of transforming an XML document, using an XSLT stylesheet.

Example 3-6 Sample XML to HTML transformation code

```
File xml = new File("fileName.xml");
File xslt = new File("fileName.xsl");
File html = new File("fileName.html");

javax.xml.transform.Source xmlSource =
   new javax.xml.transform.stream.StreamSource(xml);
javax.xml.transform.Source xsltSource =
   new javax.xml.transform.stream.StreamSource(xslt);
javax.xml.transform.Result result =
   new javax.xml.transform.stream.StreamResult(html);

//create an instance of TransformerFactory
javax.xml.transform.TransformerFactory transFact =
   javax.xml.transform.TransformerFactory.newInstance();
javax.xml.transform.Transformer trans =
   transFact.newTransformer(xsltSource);
trans.transform(xmlSource, result);
```

Studying Example 3-6, notice that JAXP can read directly from a `File` object, so three files objects are created for the source XML, the XSLT stylesheet, and the target HTML file. The Source interface is used to read both the XML file and the XSLT file. The Source interface can have many implementations. In this example, we are using `StreamSource`, which has the ability to read a File object, an `InputStream`, a `Reader`, or a system ID. Later we will examine additional Source implementations that use SAX and DOM as input. Just like `Source`, `Result` is an interface that can have several implementations. In this example, a `StreamResult` sends the output of the transformations to a target file.

Next an instance of `TransformerFactory` is created. The `TransformerFactory` is responsible for creating `Transformer` and `Template` objects. In our example, we create a `Transformer` object. `Transformer` objects are not thread-safe, although they can be used multiple times. For a simple example like this, we will not encounter any problems. In a threaded servlet environment, however, multiple users cannot concurrently access the same `TransformerFactory` instance. JAXP also provides a templates interface, which represents a stylesheet that can be accessed by many concurrent threads.

The transformer instance is then used to perform the actual transformation. It applies the XSLT stylesheet to the XML data, sending the result to the target file.

XSLT support packages in JAXP

JAXP support for XSLT is broken down into the packages listed below:

- `javax.xml.transform`: defines a general purpose API for XML transformations without any dependencies on SAX or DOM. The Transformer class is obtained from the TransformerFactory class. The Transformer transforms from a source to a result.
- `javax.xml.transform.dom`: defines how transformations can be performed using DOM. It also provides implementations of `Source` and `Result`: `DOMSource` and `DOMResult`.
- `javax.xml.transform.sax`: supports SAX2 transformations. It defines SAX versions of `Source` and `Result`: `SAXSource` and `SAXResult`. It also defines a subclass of `TransformerFactory` that allows SAX2 events to be fed into an XSLT processor.
- `javax.xml.transform.stream`: defines I/O stream implementations of `Source` and `Result`: `StreamSource` and `StreamResult`.

The heart of JAXP XSLT support lies in the `javax.xml.transform` package, which lays out the mechanics and overall process for any transformation that is performed.

Stylesheet compilation

Before a stylesheet can be processed, it must be converted into some internal machine readable format. Usually the stylesheet is read into memory using an XML parser, translated into machine format and then preserved in memory for repeated use. This is called stylesheet compilation.

Different XSLT processors implement stylesheet compilation differently, so JAXP includes the `javax.xml.transform.Templates` interface to provide consistency. The `newTransformer()` method of this interface is widely used. It allows you to obtain a new instance of a class that implements the `Transformer` interface. It is this `Transformer` object that actually allows you to perform XSLT transformations. Since the implementation of the `Templates` interface is hidden by JAXP, it must be created by the following method on `javax.xml.transform.TransformerFactory`:

```
public Template newTemplates(Source source) throws
TransformeConfigurationException
```

As mentioned earlier, the `Source` may obtain the XSLT stylesheet from one of many locations, including a filename, a system identifier, or even a DOM tree. Regardless of the original location, the XSLT processor is supposed to compile the stylesheet into an optimized internal representation.

Introduction to IBM WebSphere Application Developer

This chapter contains an introduction to the concepts behind Application Developer, and an overview of the features of the various members of the WebSphere Studio family of tools.

4.1 WebSphere Studio product family

The WebSphere Studio family of products is built on the top of the Eclipse Workbench as a set of plug-ins conforming to the Workbench's open standard APIs. These products are then follow-on technology for WebSphere Studio Advanced Edition V3 and VisualAge for Java Enterprise Edition V4.

The WebSphere Studio family currently has the following members Figure 4-1:

- WebSphere Studio Site Developer Advanced
- WebSphere Studio Application Developer
- WebSphere Studio Application Developer Integration Edition
- WebSphere Studio Enterprise Developer

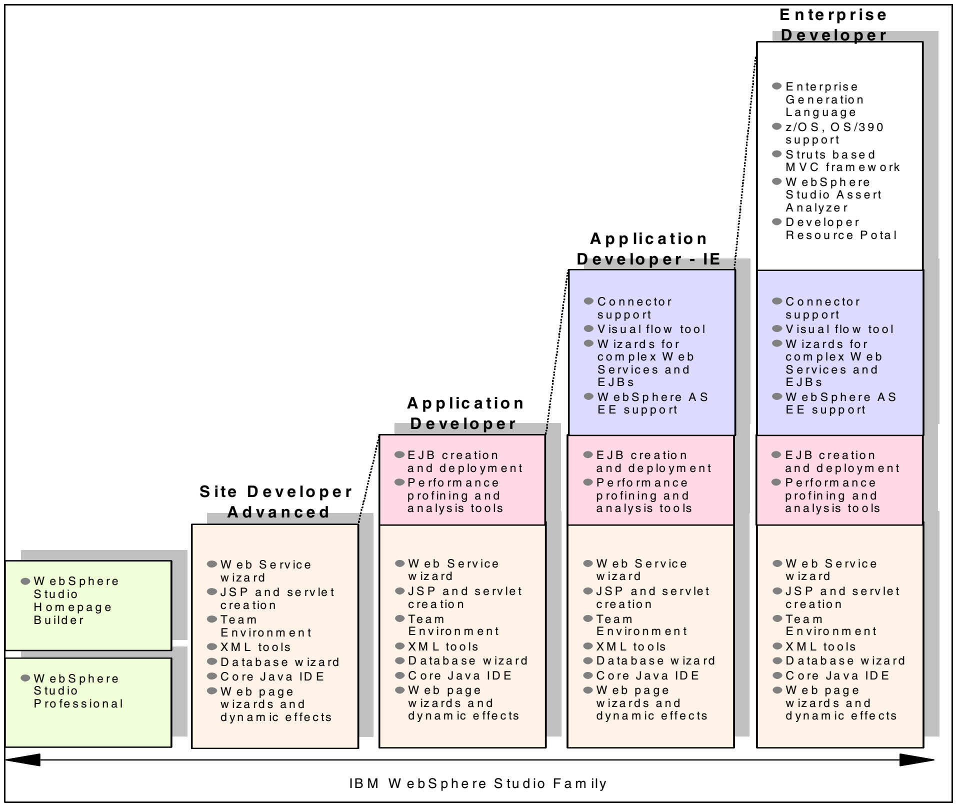

WebSphere Studio Homepage Builder
WebSphere Studio Professional
Site Developer Advanced
Web Service wizard
JSP and servlet creation
Team Environment
XML tools
Database wizard
Core Java IDE
Web page wizards and dynamic effects
Application Developer
EJB creation and deployment
Performance profining and analysis tools
Application Developer - IE
Connector support
Visual flow tool
Wizards for complex Web Services and EJBs
WebSphere AS EE support
Enterprise Developer
Enterprise Generation Language
z/OS, OS/390 support
Struts based MVC framework
WebSphere Studio Assert Analyzer
Developer Resource Potal
IBM WebSphere Studio Family

Figure 4-1 WebSphere Studio family

These products provide support for end-to-end development, testing, and deployment of Web and J2EE applications.

The WebSphere Studio family of products provide integrated development tools for all e-business development roles including Web developers, Java developers, business analysts, architects, and Enterprise programmers. The customizable, targeted and role-based approach of the Workbench will be a common characteristic of future products in the WebSphere Studio family.

WebSphere Studio Site Developer Advanced

Site Developer Advanced is an IDE intended for Web developers who develop and manage complex Web sites. It is an easy-to-use toolset that minimizes the time and effort required to create, manage, and debug multi-platform Web sites. It is designed according to the J2SE and J2EE specifications, and supports JSPs, servlets, HTML, JavaScript, and DHTML. It further includes tools for developing images and animated GIFs.

Site Developer Advanced enables Web developers to use their favorite content creation tools in conjunction with the built-in local and remote publishing capabilities.

Using Site Developer Advanced, you can develop Web applications that use the following technologies.

- **JSPs:** A simple, fast and consistent way to extend Web server functionality and create dynamic Web content. JSPs enable rapid development of Web applications that are server and platform-independent.
- **Servlets:** Server code that executes within a Web Application Server.
- **Web services:** Self-contained, modular applications that can be described, published, located, and invoked over the Internet or within intranets.

WebSphere Studio Application Developer

Application Developer is designed for professional developers of Java and J2EE applications, who require integrated Web, XML, and Web services support.

It includes all of the features of Site Developer Advanced, and adds tools for developing EJB applications, as well as performance profiling and logging tools for both local and remote execution.

Developers can quickly build and test business logic and enhance the presentation artifacts with built-in Web creation tools inside the Application Developer IDE before deploying to a production server.

Using the performance profiling and tracing tools makes it possible to detect application performance bottlenecks earlier in the development cycle. Furthermore, the built-in test environment for WebSphere Application Server and advanced tools for code generation help to shorten the test cycle.

WebSphere Studio Application Developer Integration Edition

Integration Edition includes all of the functionality in Application Developer, plus:

- Powerful graphical tools to help you quickly and easily build custom application adapters to integrate your J2EE application with your back-end systems, helping you save time and money by reusing existing resources.
- Visual flow-based tools that increase developer productivity by allowing them to visually define the sequence and flow of information between application artifacts such as adapters, Enterprise JavaBeans components, and Web services.
- Wizards that help in building and deploying complex Web services out of adapters, EJB components, flows, and other Web services.
- Support for the full set of Enterprise services provided by WebSphere Application Server Enterprise Edition such as Business Rule Beans, internationalization, and work areas that deliver additional integration capabilities, developer productivity, and business agility.

WebSphere Enterprise Developer

Enterprise Developer includes all of the functionality in WebSphere Studio Application Developer Integration Edition, plus:

Enterprise Developer can be used to implement Struts-based MVC applications using connectors and the *Enterprise Generation Language* (EGL).

The ability to connect components is the first step in modernizing the application portfolio of Enterprises. It supports creating and connecting Web applications to Enterprise business logic using the Struts-based Model-View-Controller framework and associated tooling.

Two other core technologies are integrated within Enterprise Developer.

- **WebSphere Studio Asset Analyzer (WSAA)**: Identifies application processes and connecting points, and provides the ability to generate components from existing code
- **Developer Resource Portal (DRP):** Provides collaborative capabilities across the entire development process

Enterprise Developer addresses the needs of large Enterprises, providing a model based paradigm for building applications in a Struts-based Model-View-Controller framework. It provides a visual construction and assembly based environment supporting the implementation of Enterprise level applications, including support for the multiple developer roles and technologies required by those applications. Some examples of technologies supported are HTML, JSPs, servlets, EJBs, COBOL, EGL, PL/I, and connectors.

4.2 Tools

The WebSphere Studio family of products includes the following basic tools:

- Web development tools
- Relational database tools
- XML tools
- Java development tools
- Web services development tools
- Team collaboration tools
- Integrated debugger
- Server tools for testing and deployment
- Enterprise JavaBean development tools (not in Site Developer Advanced)
- Performance profiling tools (not in WSSD)
- Plug-in development tools

4.2.1 Web development tools

The professional Web development environment provides the necessary tools to develop sophisticated Web applications consisting of static HTML pages, JSPs, servlets, XML deployment descriptors, and other resources.

Wizards are available to generate running Web applications based on SQL queries and JavaBeans. Links between Web pages can be automatically updated when content changes. There are also tools for creating images and animated GIFs.

The Web development environment bring all aspects of Web application development into one common interface. Everyone on your Web development team including content authors, graphic artists, programmers, and Web masters, can work on the same projects and access the files they need.

Such an integrated Web development environment makes it easy to collaboratively create, assemble, publish, deploy, and maintain dynamic, interactive Web applications.

The Web development tools provide the following features:

- Support for latest Web technology with an intuitive user interface
- Advanced scripting support to create client-side dynamic applications with VBScript or JavaScript
- Web Art Designer to create graphic titles, logos, buttons, and photo frames with professional-looking touches
- Animated GIF designer to create life-like animation from still pictures, graphics, and animated banners

- Over 2,000 images and sounds in the built-in library
- Integrated, easy-to-use visual layout tool for JSP and HTML file creation and editing
- Web project creation, using the J2EE-defined hierarchy
- Creation and visual editing of the Web application deployment descriptor (web.xml) file
- Automatic update of links as resources are moved or renamed
- A wizard for creating servlets
- Generation of Web applications from database queries and JavaBeans
- J2EE WAR/EAR deployment support (not in Site Developer Advanced)
- Integration with the WebSphere unit test environment

4.2.2 Relational database tools

The database tools provided with the WebSphere family products allow you to create and manipulate the data design for your project in terms of relational database schemas.

You can explore, import, design, and query databases working with a local copy of an already existing design. You can also create an entirely new data design from scratch to meet your requirements.

The database tools provide a metadata model that is used by all other tools that need relational database information. This includes database connection information. In that way tools, although that unaware of each other are able to share connections.

The SQL statement wizard and SQL query builder provide a GUI-based interface for creating and executing SQL statements. When you are satisfied with your statement, you can use the SQL to XML wizard to create an XML document, as well as XSL, DTD, XSD, HTML, and other related artifacts.

The relational database tools support connecting *to*, and importing *from* several database types including DB2, Oracle, SQL Server, Sybase, and Informix.

4.2.3 XML tools

The comprehensive XML toolset provided by the WebSphere Studio family of products includes components for building DTDs, XML Schemas, and XML files. With the XML tools, you can perform all of the following tasks:

- Create, view, and validate DTDs, XML Schemas, and XML files.

- Create XML documents from a DTD, from an XML Schema, or from scratch.
- Generate JavaBeans from a DTD or XML Schema.
- Define mappings between XML documents and generate XSLT scripts that transform documents.
- Create an HTML or XML document by applying an XSL style sheet to an XML document.
- Map XML files to create an XSL transformation script and to visually step through the XSL file.
- Define mappings between relational tables and DTD files, or between SQL statements and DTD files, to generate a document access definition (DAD) script, used by IBM DB2 XML Extender. This can be used to either compose XML documents from existing DB2 data or decompose XML documents into DB2 data.
- Generate DADX, XML, and related artifacts from SQL statements, and use these files to implement your query in other applications.

4.2.4 Java development tools

All WebSphere Studio family of products provide a professional-grade Java development environment with the following capabilities:

- Application Developer v5.0 ships with JDK 1.3
- Pluggable run-time support for JRE switching and targeting of multiple run-time environments from IBM and other vendors
- Incremental compilation
- Ability to run code with errors in methods
- Crash protection and auto-recovery
- Error reporting and correction
- Java text editor with full syntax highlighting and complete content assist
- Refactoring tools for reorganizing Java applications
- Intelligent search, compare, and merge tools for Java source files
- Scrapbook for evaluating code snippets

4.2.5 Web services development tools

Web services represent the next level of function and efficiency in e-business. Web services are modular, standards-based e-business applications that businesses can dynamically mix and match in order to perform complex transactions with minimal programming. The WebSphere Studio family of products that include the Web services feature, help you to build and deploy Web services-enabled applications across the broadest range of software and hardware platforms used by today's businesses. These tools are based on open, cross-platform standards such as Universal Description Discovery and Integration (UDDI), Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL).

4.2.6 EJB development tools

The WebSphere Studio family (except Site Developer Advanced) feature full EJB support (Application Developer v5.0 supports EJB2.0 and EJB 1.1), an updated EJB test client, an enhanced unit test environment for J2EE, and deployment support for Web application archive (WAR) files and Enterprise application archive (EAR) files. Entity beans can be mapped to databases, and EJB components can be generated to tie into transaction processing systems. XML provides an extended format for deployment descriptors within EJB.

4.2.7 Team collaboration

Team developers do all of their work in their individual workbenches, and then periodically make changes to a *team stream*. This model allows individual developers to work on a team project, share their work with others as changes are made, and access the work of other developers as the project evolves. At any time, developers can update their work space by retrieving the changes that have been made to the team stream or by submitting changes to the team stream.

All products of the WebSphere Studio family support the Concurrent Versions System (CVS) and the Rational ClearCase LT products among others.

Other software configuration management (SCM) repositories can be integrated through the Eclipse Workbench SCM adapters. SCM adapters for commercial products are provided by the vendors of those products.

4.2.8 Debugging tools

The WebSphere Studio family products include a debugger that enables you to detect and diagnose errors in your programs running either locally or remotely. The debugger allows you to control the execution of your program by setting breakpoints, suspending launches, stepping through your code, and examining the contents of variables.

You can debug live server-side code as well as programs running locally on your workstation.

The debugger includes a debug view that shows threads and stack frames, a process view that shows all currently running and recently terminated processes, and a console view that allows developers to interact with running processes.

There are also views that display breakpoints and allow you to inspect variables.

4.2.9 Performance profiling tools

The WebSphere Studio family except Site Developer Advanced provide tools that enable you to test the performance of your application. This allows you to make architectural and implementation changes early in your development cycle, and significantly reduces the risk of finding serious problems in the final performance tests.

The profiling tools collect data related to a Java program's run-time behavior, and present this data in graphical and non-graphical views. This assists you in visualizing program execution and exploring different patterns within the program.

These tools are useful for performance analysis, and for gaining a deeper understanding of your Java programs. You can view object creation and garbage collection, execution sequences, thread interaction, and object references. The tools also shows you which operations take the most time, and help you find and plug memory leaks. You can easily identify repetitive execution behavior and eliminate redundancy, while focusing on the highlights of the execution.

4.2.10 Server tools for testing and deployment

The server tools provide a unit test environment where you can test JSPs, servlets and HTML files, (EJB testing is supported in Application Developer and Enterprise Developer). You also have the capability to configure other local or remote servers for integrated testing and debugging of J2EE applications.

The following features are included:

- ► A copy of the complete WebSphere Application Server Developer Edition (AEs) run-time environment
- ► Stand-alone unit testing
- ► Ability to debug live server-side code using the integrated debugger
- ► Support for configuring multiple servers

The server tools support the following run-time environments:

- ► WebSphere Application Server AEs, which can be installed locally or remotely. It supports testing of both EJBs and Web applications.

- Apache Tomcat, which can be installed only locally and supports testing of Web applications.

4.2.11 Plug-in development tools

The WebSphere Studio family (except for Site Developer Advanced) include the PDE (Plug-in Development Environment) which is designed to help you develop platform plug-ins while working inside the platform workbench, and it provides a set of platform extension contributions (views, editors, perspectives, etc.) that collectively streamline the process of developing plug-ins inside the workbench. The PDE is not a separate tool, but it is a one of perspectives. PDE blends with the platform and offers its capabilities through a new perspective.

The following project types are supported:

- **Plug-in project**

  WebSphere Studio Application Developer is based on the concept of plug-ins, which have a clearly defined structure and specification. This project supports creating, testing , and deploying a plug-in *in* the PDE.

- **Fragment project**

  A plug-in fragment is used to provide additional plug-in functionality to an existing plug-in after it has been installed. Fragments are ideal for shipping features like language or maintenance packs, which typically trail the initial products by a few months.

- **Plug-in component**

  PDE attaches a special component nature to plug-in and fragment projects to differentiate them from other project types. The project must have a specific folder structure and a component manifest. The project must be set up with references to all of the plug-in and fragment projects, which will be packaged into the component.

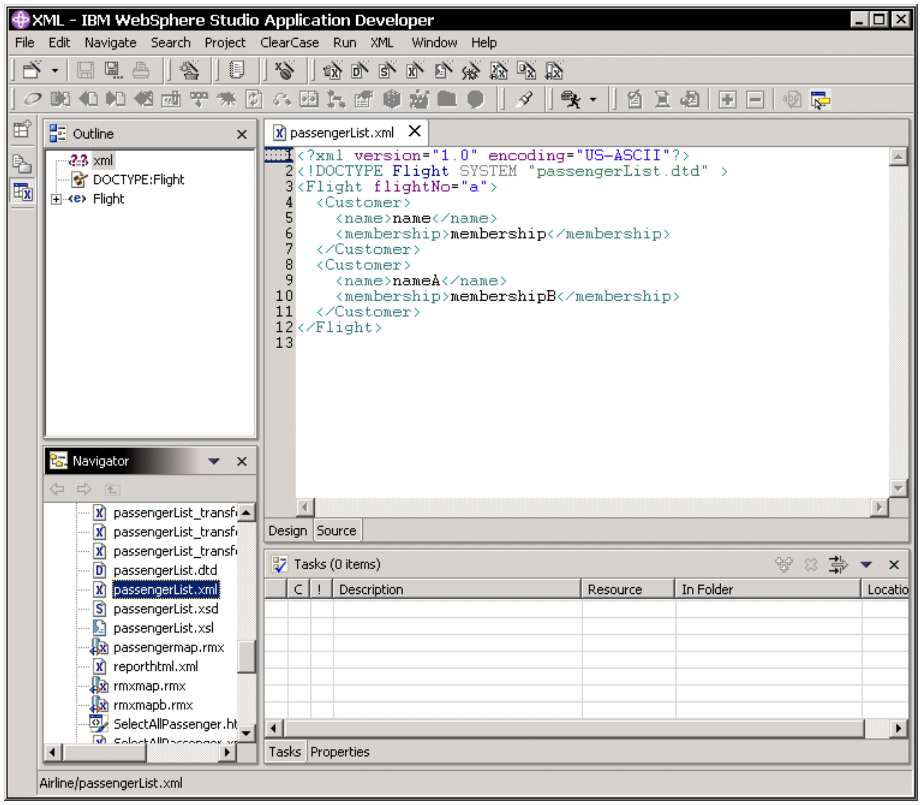
5 Application Developer XML Tools

Chapter four introduces the XML development capabilities of Websphere Application Developer. In particular, the chapter covers the following topics:

- XML perspective
- XML editors
- XML support features in Websphere Studio Application Developer

5.1 XML perspective

Perspectives are a way to look through different views to a a project. Depending on the role of the developer (whether a Web developer, or a Java developer, or an EJB developer, etc.). And also depending on the tasks that the developer must perform, the developer uses a different perspective. The XML perspective is the perspective for XML development in the Application Developer. It contains several editors and views that can help a developer in building XML files, XML Schemas, DTDs, stylesheets, and integrating between data extracted from relational databases and XML. To open the XML perspective select **Window—>Open Perspective—>XML**



```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE Flight SYSTEM "passengerList.dtd" >
<Flight flightNo="a">
  <Customer>
    <name>name</name>
    <membership>membership</membership>
  </Customer>
  <Customer>
    <name>nameA</name>
    <membership>membershipB</membership>
  </Customer>
</Flight>
```

Figure 5-1 XML perspective

The XML perspective (Figure 5-1) contains four sections:

- The view on the top left of the perspective shows the Outline view for the active editor; in this case the XML editor is active.
- The view on the top right shows the active editors

- The view on the bottom left shows the Navigator view, which displays the folders, and files of the project
- The view on the bottom right shows the Tasks view, which shows the problems and errors to be fixed

5.2 XML perspective editors

In this section, we demonstrate the editing capabilities of the XML components editors available in the Application Developer.

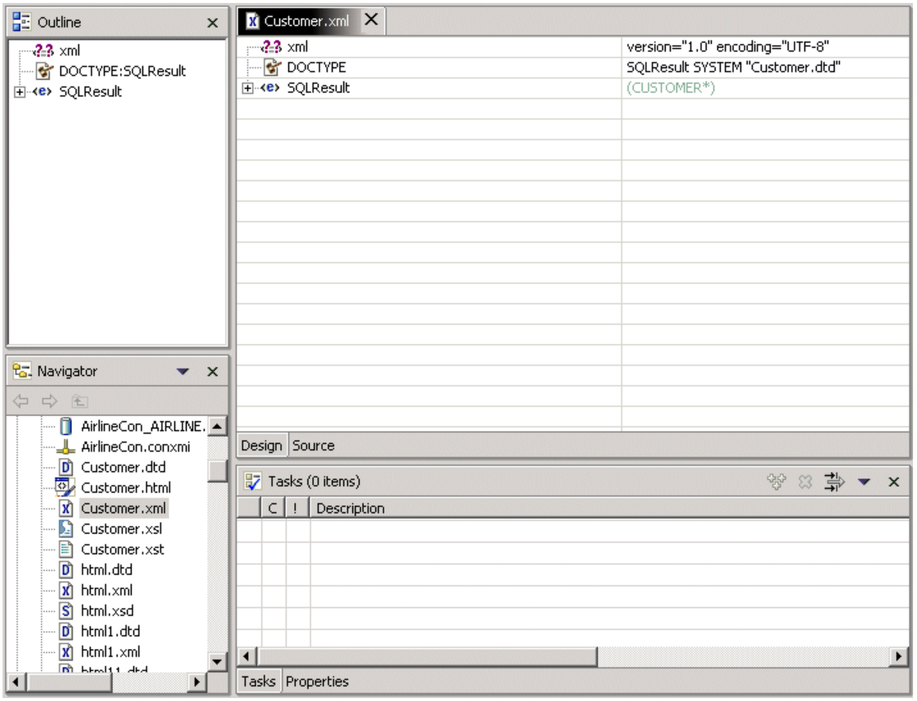
5.2.1 XML editor

This is a tool for creating and viewing XML files. You can use it to edit the contents of new XML files, either created from scratch or from existing DTDs, or XML Schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and validate them.

Creating an XML file from scratch

In this section, we show how to create a new XML file. To create a new file from scratch, use the New XML wizard as follows:

- Select **File—>New—>Project—>Simple —>Project** to bring up the New Project wizard to create a simple project.
- Select **File—>New—>XML** to launch the new XML wizard.
- Select the option **Create XML file from scratch**.
- Select **Next**.
- In the File name field, type your XML file name.
- Click **Finish**. The XML file is created and the XML Editor is automatically opened for you, as shown in Figure 5-2.



xml	version="1.0" encoding="UTF-8"
DOCTYPE	SQLResult SYSTEM "Customer.dtd"
SQLResult	(CUSTOMER*)

Figure 5-2 XML editor

Editing an XML file

The XML Editor has two main views: Source view, Design view, in addition to utilizing the Outline view, and Task view. The Source view is a text editor that lets you directly edit the source of the XML document. It has several text editing features, such as syntax highlighting, and content assist, which uses the information in a DTD or schema content model to provide a list of acceptable continuations depending on where the cursor is located in an XML file, or what has just been types. The XML editor Source view also includes a smart double-clicking behavior. If your cursor is placed in an attribute value, one double-click selects that value, another double-click selects the attribute-value pair, and a third double-click selects the entire tag. This makes it easier to copy and paste commonly used pieces of your XML code.

The Design view displays the XML document as a tree enabling you to manipulate the document by adding, removing, and editing tree nodes. This also makes navigation easier. Note that in the left column of the Design view, we see the elements, attributes, and other nodes of the XML document's tree. The right column is used to display the values associated with these nodes. Content and attribute values can be edited directly in the table cells, while right-click menus on the tree nodes give alternatives that are valid for that location. The right column is also used to display content model information associated with elements.

The Outline view provides an overview of the XML document that can be useful when navigating large documents. And finally the Task view displays error messages that may be associated with the XML document.

Note the editor's page tabs at the bottom of the top right pane. These tabs are used to switch between the Design view and the Source view of the XML Editor. The views are synchronized so that a change made in one view will be automatically reflected in the other view.

Validating the XML file

Another useful feature of the XML editor is the incremental validation feature. At any point during your development of an XML document, you can invoke the validate process to validate the file. Just right-click on the file name in the Navigator view, and select Validate XML File option. In addition to the manual invocation of the validation process, the validation is also automatically run when you save the document, or when you finish typing in the Source view (as indicated by switching focus to a different view). Any validation errors will be reported in the Task view with a little red marker for the corresponding line in the Source view and the corresponding object in the Outline view.

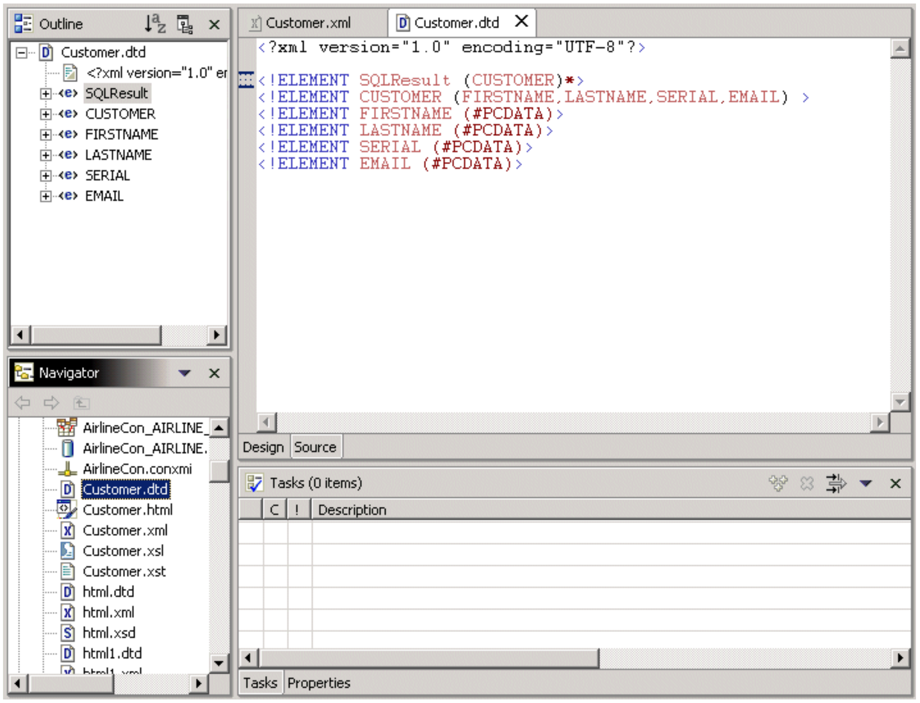
5.2.2 DTD editor

This is a tool for creating and viewing DTDs. You can use it to edit the contents of new DTD files, either created from scratch or from existing XML Schemas. You can also use it to edit existing DTD files, and validate them.

Creating a DTD from scratch

In this section, we show how to create a new DTD file. To create a new DTD from scratch, use the New DTD wizard as follows:

- Select **File—>New—>Project—>Simple —>Project** to bring up the New Project wizard to create a simple project.
- Select **File—>New—>DTD** to launch the New DTD wizard.
- In the File name field, type your DTD file name.
- Click **Finish**. The DTD file is created and the DTD editor is automatically opened for you, as shown in Figure 5-3 on page 84.



```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT SQLResult (CUSTOMER)*>
<!ELEMENT CUSTOMER (FIRSTNAME,LASTNAME,SERIAL,EMAIL) >
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT SERIAL (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
```

Figure 5-3 DTD Editor

In order to handle the contents of a DTD file, you can use the Outline view to add or remove components of your DTD. When you select an object in the Outline view, the Design view will display the properties that are associated with that DTD component object. You can use the Design view to enter values for the selected object. You can switch to the Source view to edit the DTD source directly. The DTD editor also uses the Task view from the workbench for errors reporting.

Using the Outline view to add DTD components

The DTD specification defines a large number of components such as elements, entities, notations, comments, attributes, etc. To create a valid DTD, you must understand the containment relationships between these components.The XML Schema Editor removes the burden to remember these details for you. You can use the Outline view to add DTD components via the pop-up menu as shown in Figure 5-4 on page 85. The pop-up menu will only display the list of objects that are relevant for the selected object. It will also add the object at the correct location in the DTD.

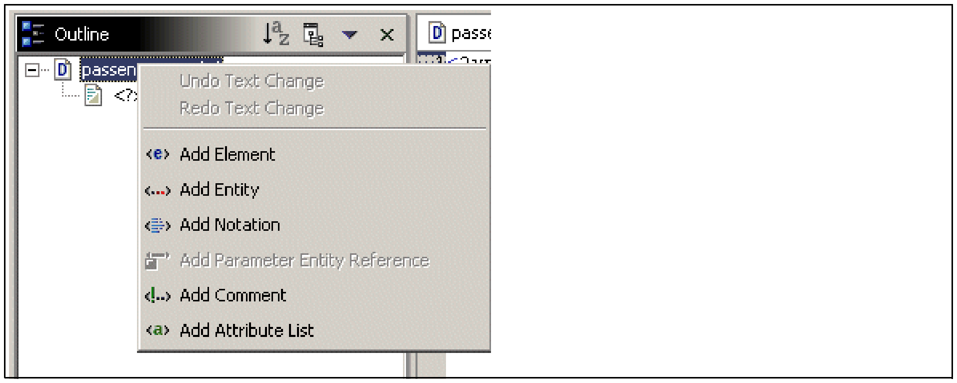


Figure 5-4 Pop-up on DTD file in the Outline view

Validating DTD

Another useful feature of the DTD editor is the incremental validation feature. At any point during your development of a DTD, you can invoke the Validate process to validate the DTD. Just right-click on the file name in the Navigator view, and select Validate DTD option. In addition to the manual invocation of the validation process, the validation is also automatically run when you save the document, or when you finish typing in the Source view (as indicated by switching focus to a different view). Any validation errors will be reported in the Task view with a little red marker for the corresponding line in the Source view and the corresponding object in the Outline view.

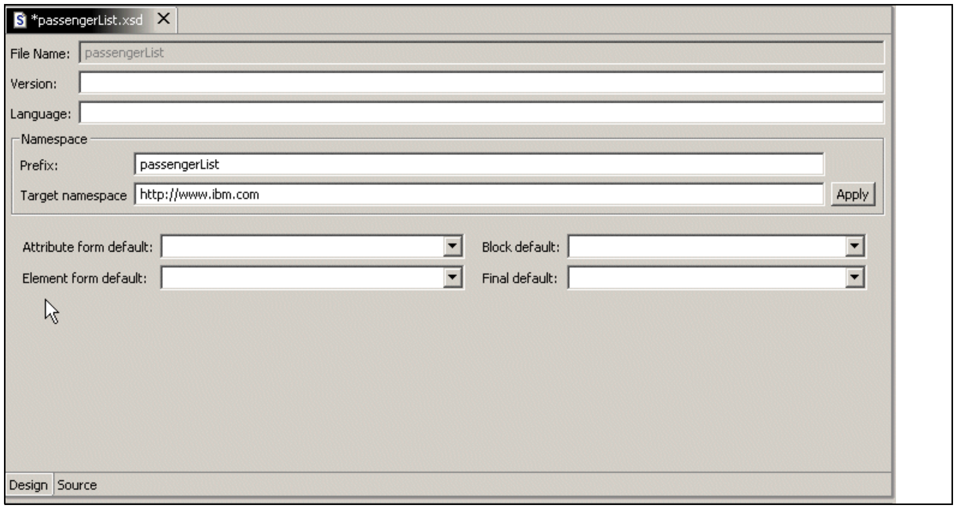
5.2.3 XSD editor

An XML Schema editor is a tool for creating, viewing, and validating XML Schemas. You can use the XML Schema editor to perform tasks such as creating XML Schema components, importing and viewing XML Schemas.

Creating a schema from scratch

In this section, we show how to create a new XML Schema. To create a new schema from scratch, use the New XML Schema wizard as follows:

- Select **File—>New—>Project—>Simple —>Project** to bring up the New Project wizard to create a simple project.
- Select **File—>New—>XML Schema** to launch the New XML Schema wizard.
- In the File name field, type your schema file name.
- Click **Finish**. The schema file is created and the XML Schema Editor is automatically opened for you, as shown in Figure 5-5 on page 86.


*passengerList.xsd

File Name: passengerList

Version:

Language:

Namespace

Prefix: passengerList

Target namespace http://www.ibm.com

Figure 5-5 XML Schema Editor

In order to handle the contents of a schema file, you can use the Outline view to add, remove, or rearrange components of your schema. When you select an object in the Outline view, the Design view will display the properties that are associated with that schema component object. you can use the Design view to enter values for the selected object. You can switch to the Source view to edit the schema source directly. The XML Schema editor also uses the Task view from the workbench for errors reporting.

Using the Outline view to add schema components

The XML Schema specification defines a large number of components such as schema, complexType, simpleType, group, annotation, include, import, element, and attribute, etc. To create a valid schema, you must understand the containment relationships between these components. For example, an attribute can only be added to a complex type, but not a simple type. A group can only be defined at the schema level, but can be referenced by a complex type, etc.

The XML Schema editor removes the burden to remember all these details for you. You can use the Outline view to add schema components via the pop-up menu, as shown in Figure 5-6 on page 87. The pop-up menu will only display the list of objects that are relevant for the selected object. It will also add the object at the correct location in the XML Schema.

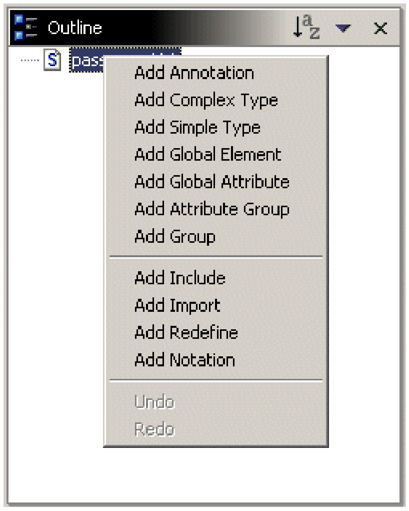


Figure 5-6 Pop-up on schema file in the Outline view

Making changes and referential integrity

As a schema becomes bigger and more complex, there will be more type definitions, and references to those types. So what happens after you have defined a type, created ten references to that type, and you want to change the name of the type? The XML Schema editor has a built-in referential integrity mechanism that will propagate the changes automatically, freeing you from the tedious and error-prone task of doing the manual updates.

Let us assume that in your xsd file, you have defined a simple type called `SimpleType1`. There is a reference to this type in the complex type named `ComplexType1`. Let us say we want to change the simple type name from SimpleType1 to `SimpleType2`.The following illustrates how to do that:

- Switch to the Design view.
- Select the **SimpleType1** in the Outline view. In the Design view, change it to `SimpleType2`.
- Now switch over to the Source view. Notice how all references to `SimpleType1` change to `SimpleType2` automatically.

The XML Schema editor's referential integrity mechanism is not limited only to name change. The same rule applies when you delete a schema component. For example, if you delete a type, all references to that type will automatically be reset to the default string data type. Whenever such an automatic update occurs, an information message will be displayed in the Task view. You can always invoke the undo action if you want to change your mind.

One thing to note is that the built-in referential integrity mechanism will only be enforced if the change is made from the Design view. If you make the changes directly by typing in the Source view, then it is your responsibility to ensure that you make all the changes correctly. Any change you made in the Source view will automatically be reflected in the Design and Outline view.

Namespace

Namespace provides a way to identify where an element or attribute comes from. For example, two elements from two different schemas might have the same name (for example, `SampleElement`). To identify them, let us say Schema A's `SampleElement` vs. Schema B's `SampleElement`; we can use the XML Namespaces mechanism to distinguish them.

This section provides a quick introduction on how to define namespace for your schema in the XML Schema editor. In the xsd file, the target namespace is by default is `http://www.ibm.com`. This is indicated by the targetNamespace attribute in the schema element. This means that all the types that are defined in this schema belong to the target namespace `http://www.ibm.com`.

Usually, your schema has a prefix for its target namespace. To refer to any type defined in your schema, you must use this defined prefix. If you want to change the namespace prefix or the target namespace for your schema, you can use the Design view to do this. The following demonstrates this feature:

- Select the file object, that is your schema.
- In the Design view, change the Prefix field to any prefix you desire.
- In the Design view, change the Target namespace field to any namespace of your choice, for example `http://www.your application name.com`.
- Because such a change has a global impact to your entire document, you must click the **Apply** button to make this global change.
- Switch over to the Source view. You will notice that the attributes on the schema element and all the prefixes for the types are automatically changed for you.

In order to have a more detailed view about XML namespaces and their support in the Application Developer, have a look at 5.3, “Namespace support” on page 90.

Validating schema

Another useful feature of the XML Schema editor is the incremental validation feature. At any point during your development of an XML Schema, you can invoke the validate process to validate the schema. The validation is also automatically run when you save the document, or when you finish typing in the

Source view (as indicated by switching focus to a different view). Any validation errors will be reported in the Task view with a little red marker for the corresponding line in the Source view, and the corresponding object in the Outline view.

Also, when making changes in the Design view, pay attention to the status bar. It will contain hints for potential errors in the schema that you are developing.

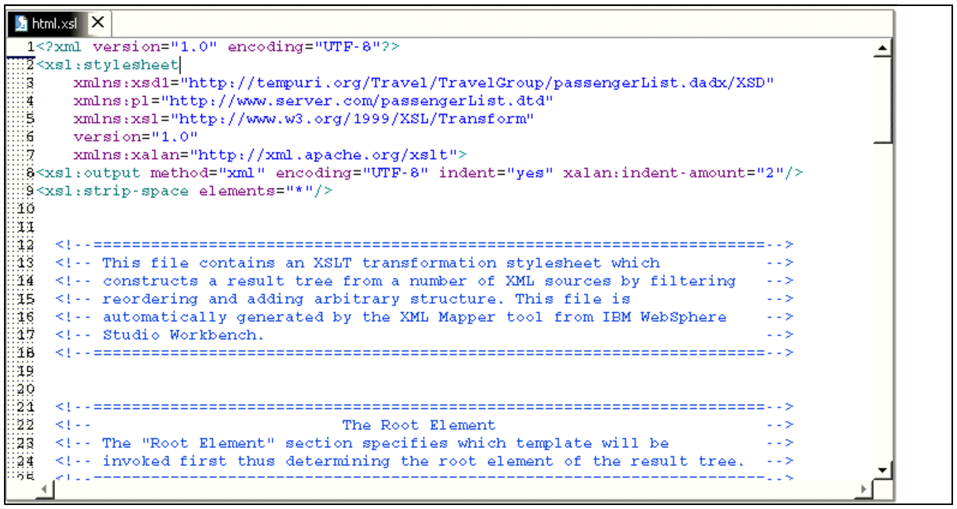
5.2.4 XSL editor

This is a tool for creating and viewing XSL files. You can use it to edit the contents of new XSL files created from scratch. Moreover, you can also use it to edit existing ones that are created using the XML to XML mapping wizard.

Creating an XSL file from scratch

In this section, we show how to create a new XSL file. To create a new file from scratch, use the New XSL wizard as follows:

- Select **File—>New—>XSL** to launch the New XSL wizard.
- In the File name field, type your XSL file name.
- Click **Finish**. The XSL file is created and the XSL editor is automatically opened for you, as shown in Figure 5-7.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsd1="http://tempuri.org/Travel/TravelGroup/passengerList.dadx/XSD"
    xmlns:pl="http://www.server.com/passengerList.dtd"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt">
<xsl:output method="xml" encoding="UTF-8" indent="yes" xalan:indent-amount="2"/>
<xsl:strip-space elements="*"/>


  <!--=====================================================================-->
  <!-- This file contains an XSLT transformation stylesheet which          -->
  <!-- constructs a result tree from a number of XML sources by filtering  -->
  <!-- reordering and adding arbitrary structure. This file is             -->
  <!-- automatically generated by the XML Mapper tool from IBM WebSphere   -->
  <!-- Studio Workbench.                                                    -->
  <!--=====================================================================-->


  <!--=====================================================================-->
  <!--                           The Root Element                          -->
  <!-- The "Root Element" section specifies which template will be         -->
  <!-- invoked first thus determining the root element of the result tree. -->
```

Figure 5-7   XSL editor

Editing an XSL file

The XSL editor provides a text editor to handle the source code of the XSL file. It has several text editing features, such as content assist and syntax highlighting. The content assist feature helps you in writing you XSL code, as it is aware of the proper XSL grammar rules. When editing the XSL document, content assist can be invoked to prompt the you with a list of possible XSL constructs to use. Concerning the syntax highlighting feature, you will notice that whenever you select any of the XSL file components listed in the Outline view, the corresponding XSL code for that component will be selected, and vice versa.

Validating the XSL file

Another useful feature of the XSL editor, like the rest of the editors, is the incremental validation feature. At any point during your development of an XSL file, you can invoke the Validate process to validate the file. Just right-click on the file name in the Navigator view, and select the **Validate XSL File** option. In addition to the manual invocation of the validation process, the validation is also automatically run when you save the document. Any validation errors will be reported in the Task view with a little red marker for the corresponding line in the source code of the XSL file.

5.3 Namespace support

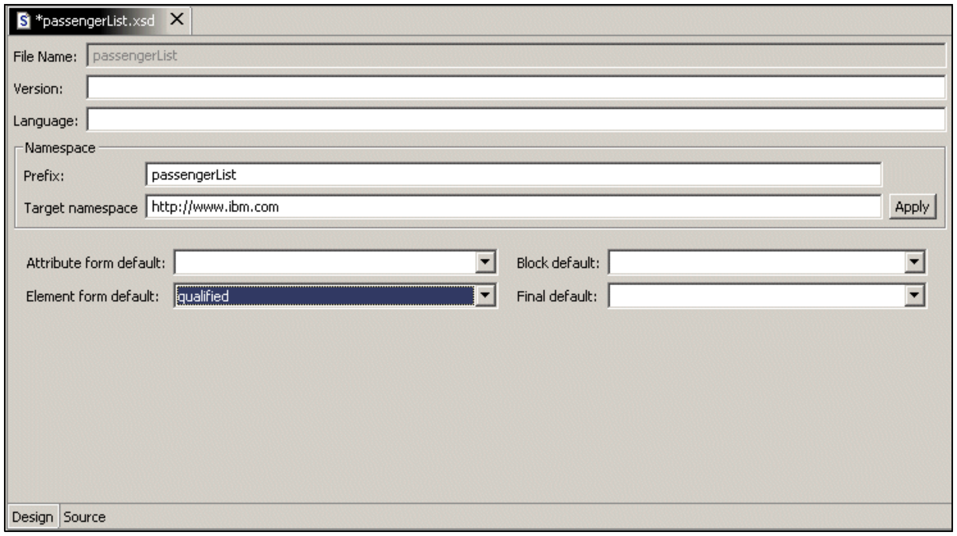
Namespaces are useful when there is a need for elements and attributes of the same name to take on different meaning depending on the context in which they are used. So, the XML namespace is the means to distinguish between elements that have the same name but come from different problem domain. An XML Schema supports namespaces.

Let us say for example we are taking about an attribute called Name. It will can have different meanings depending on whether it is applied to a customer, or flight carrier. If both entities (a customer and a flight carrier) need to define the same document such as in a flight entry, which associates a flight with its corresponding information, and the list passengers on that flight. We need some mechanism to distinguish between the two, and apply the correct semantic description for the attribute name, whenever it is used in the document. Namespaces provide the mechanism that allows us to write XML documents that contain information relevant to the problem domain.

To include your namespace information into your XML file when creating your XML document, select the option **Create XML file** from an XML Schema file, which specifies the namespace information. When you do that, the target namespace and prefix from the XML Schema file is automatically picked up when creating the document.

If you review the generated XML file, you will notice that the root element is qualified to belong to the target namespace specified in the schema file. It is also important to note that the local elements that belong to the root element are unqualified. That is, they do not have a prefix. This is because the schema file by default specifies that local elements should not be qualified.

In order to qualify all the local elements in an XML document, the XML Schema must set the `elementFormDefault` attribute to *qualified* in the schema element. This can be accomplished by selecting **Qualified** in the Element form default field of the Design view for the schema, as shown in Figure 5-8. All XML files created from this schema will have all the elements qualified with the namespace prefix.


*passengerList.xsd

File Name: passengerList
Version:
Language:
Namespace
Prefix: passengerList
Target namespace http://www.ibm.com
Attribute form default:
Block default:
Element form default: qualified
Final default:

Figure 5-8 Indicate all local elements to be qualified.

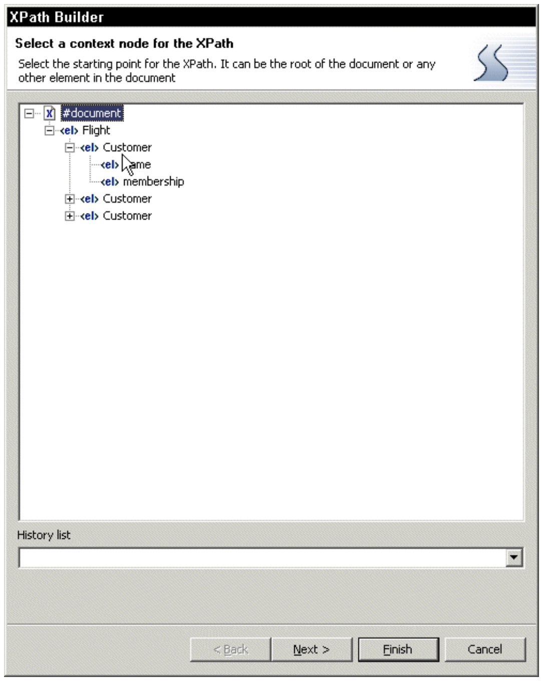
When you use the XML Schema editor to create your schema file by default, the target namespace for this schema is `http://www.ibm.com` as indicated by the target namespace attribute, as shown in Figure 5-8. If you do not specify a prefix in the prefix field for the schema object, then you are making the default namespace for this schema to be the same as the target namespace. When you click the **Apply** button, you will notice that the xmlns attribute is added to the schema tag to indicate that the default namespace of this schema is `http://www.ibm.com`. You will also notice that the XML Schema constructs will automatically be qualified with the prefix `xsd` to distinguish them from the types that are in the default namespace. By making the target namespace of this schema the default namespace, you do not have to qualify types from this schema when you reference them.

5.4 XPath support

XPath is an XSL sub-language that is designed to be used with XSLT. It is used for identifying or addressing parts of a source XML document. Every node within an XML document can be uniquely identified and located using the syntax defined for XPath.

WebSphere Application Developer provides support for XPath. To launch the XPath Builder

- Move to the XML perspective.
- Choose an XML file in your project.
- Right-click, and select **Generate —>XPath**.


XPath Builder

Select a context node for the XPath

Select the starting point for the XPath. It can be the root of the document or any other element in the document

```
#document
  <el> Flight
    <el> Customer
      <el> name
      <el> membership
    <el> Customer
    <el> Customer
```

Figure 5-9 XPath Builder

In order to define your XPath, the Application Developer provides you with means for defining all the necessary constructs. Look at Figure 5-10. After you choose the desired node in your XML tree, you will need to provide the steps for the location paths. Although location paths are not the most general grammatical construct in the language, they are the most important construct. A location path selects a set of nodes relative to the context node. The result of evaluating an expression that is a location path, is the node-set containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes. A location path has three steps:

1. An axis, which specifies the tree relationship between the nodes selected by the location step and the context node.
2. A node test, which specifies the node type and expanded-name of the nodes selected by the location step.
3. Zero or more predicates, which use arbitrary expressions to further refine the set of nodes selected by the location step.

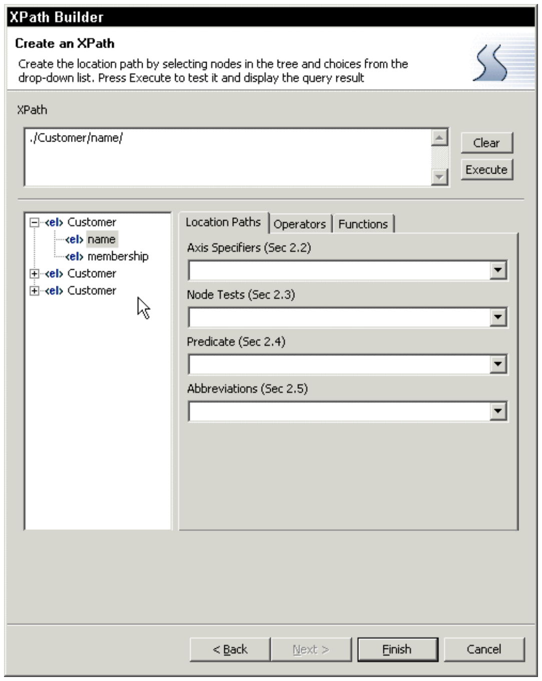


Figure 5-10 XPath definition

After you finish with the location paths, the next tab allows the usage of a operators, as shown in Figure 5-11. This wizard provides you with a list of node-set operators, boolean operators, and numeric operators. Expanding any on the list will show the available operators, from which you can select your choice.

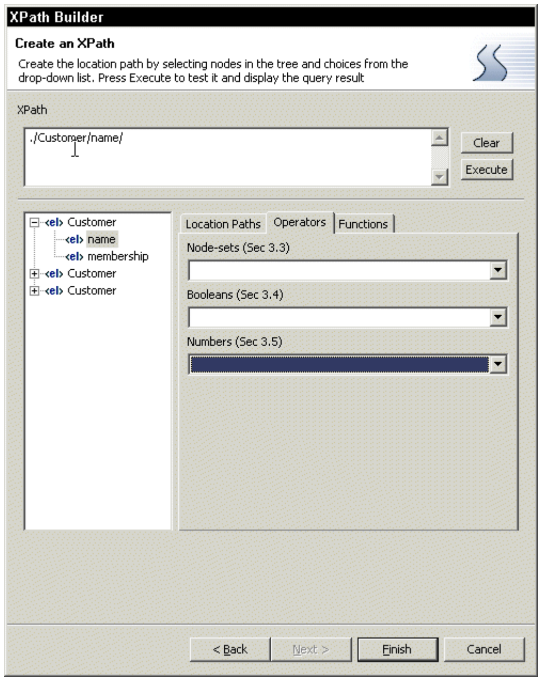
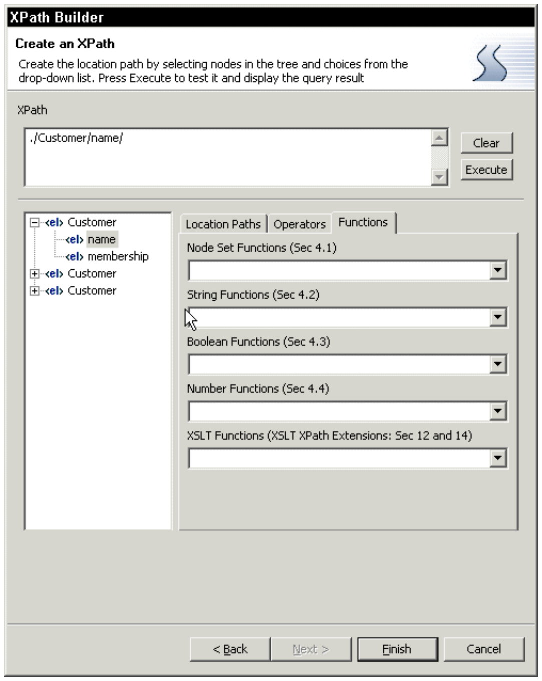


Figure 5-11 XPath Operators

The last tab shows a list of functions to use in your XPath definition, as shown in Figure 5-12. There is a list of node set functions, boolean functions, string functions, number functions, XSLT functions.

After you finish defining your XPath, you can click the **Execute** button to test your XPath. The wizard displays your XPath results, as shown in Figure 5-13.


XPath Builder

Create an XPath

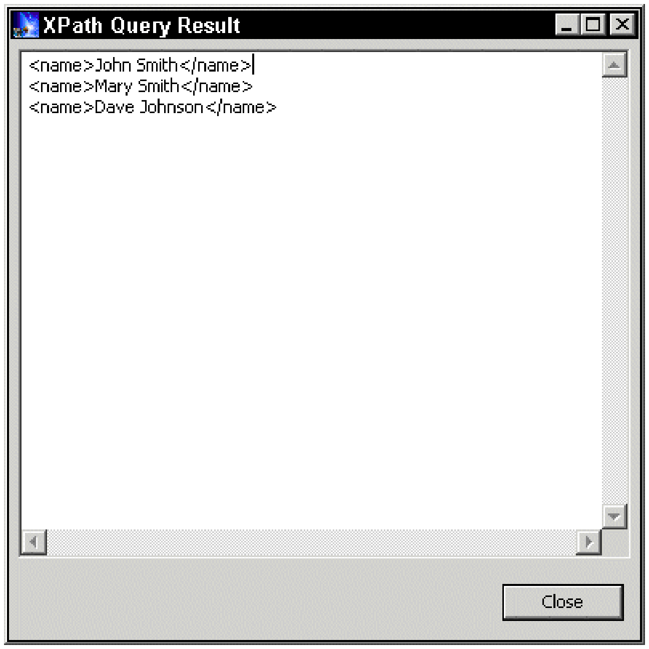
Create the location path by selecting nodes in the tree and choices from the drop-down list. Press Execute to test it and display the query result

XPath

./Customer/name/

Clear
Execute
<el> Customer
<el> name
<el> membership
<el> Customer
<el> Customer
Location Paths
Operators
Functions
Node Set Functions (Sec 4.1)
String Functions (Sec 4.2)
Boolean Functions (Sec 4.3)
Number Functions (Sec 4.4)
XSLT Functions (XSLT XPath Extensions: Sec 12 and 14)
< Back
Next >
Finish
Cancel

Figure 5-12 XPath functions



```
<name>John Smith</name>
<name>Mary Smith</name>
<name>Dave Johnson</name>
```

Figure 5-13 XPath Query Result screen

5.5 XSL debugger

This tool enables you to visually step through an XSL transformation script, highlighting the transformation rules as they are fired. It is used to test the generated XSL style sheet. Simply use the editor to apply the XSL style sheet to a source XML file, and create a new HTML or XML file. You can then trace through the new XML or HTML file to verify if the results are correct.

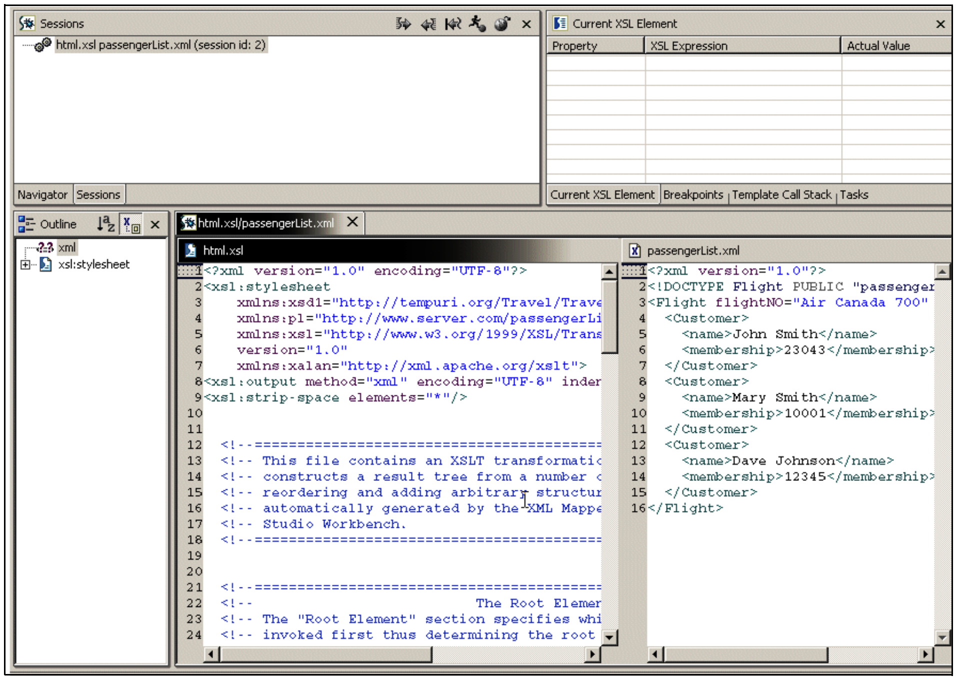
The XSL trace editor only works on a one-to-one basis. You can only apply an XSL file to one XML file at a time if you want to use the XSL trace editor to trace through the results. As well, you cannot apply as XSL file that references another XSL file to an XML file, and trace the results.

So, to invoke the XSL debugging process, select your XML file to be transformed, and XSL file in the Navigator view, then invoke the trace from the context menu by selecting **Apply XSL—>As HTML**. The trace editor presents the input XML, and the input XSL, as shown in Figure 5-14. You can replay the transform using stepping controls in the toolbar. As the transform is applied, the trace editor highlights each XSL statement, and the XML input element it matches, allowing you to understand and resolve any problems.

Moreover, the XSL debugger allows to add breakpoints to your XSL file, which facilitate the debugging process. To add a breakpoint, just double click on the line number where you want to place the breakpoint, and green ball will appear indicating that there is a breakpoint in this line. To be able to remove a breakpoint, choose the breakpoints tab in the window in the upper right hand corner shown in Figure 5-14. Just select the breakpoint you want to remove, and right-click, selecting the **Remove** option from the context menu.

The buttons in the Sessions window (in the top left hand corner shown in Figure 5-14) allow you to handle your debugging process. They provide features like step forward or step backward in the result document; restart the process from the beginning; run to breakpoint; and open the browser on the transformation result.

To close the XSL debugger, simply close the XSL debug perspective, or switch to any other perspective you desire.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsd1="http://tempuri.org/Travel/Trave
    xmlns:pl="http://www.server.com/passengerLi
    xmlns:xsl="http://www.w3.org/1999/XSL/Trans
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt">
<xsl:output method="xml" encoding="UTF-8" inder
<xsl:strip-space elements="*"/>


  <!--========================================
  <!-- This file contains an XSLT transformatic
  <!-- constructs a result tree from a number c
  <!-- reordering and adding arbitrary structur
  <!-- automatically generated by the XML Mappe
  <!-- Studio Workbench.
  <!--========================================


  <!--========================================
  <!--                              The Root Elemer
  <!-- The "Root Element" section specifies whi
  <!-- invoked first thus determining the root
```

```
<?xml version="1.0"?>
<!DOCTYPE Flight PUBLIC "passenger
<Flight flightNO="Air Canada 700"
  <Customer>
    <name>John Smith</name>
    <membership>23043</membership>
  </Customer>
  <Customer>
    <name>Mary Smith</name>
    <membership>10001</membership>
  </Customer>
  <Customer>
    <name>Dave Johnson</name>
    <membership>12345</membership>
  </Customer>
</Flight>
```

Figure 5-14   XSL Debugger

5.6 Web services support

XML and its associated family of standards play a central role in Web services by providing a data interchange format that is independent of both programming languages and operating systems. This section gives an overview of the support provided by the Application Developer for XML-based Web services. For detailed information about the development of XML Web services using Websphere Studio Application Developer, please refer to Chapter 9. Please note that we assume you already have a background about Web services.

To start developing your Web service, you will need to create a Web project in your work space to contain the Web service. To implement your XML Web service, the Application Developer provides you with the two very important wizards to use.

Web Service DADX group configuration wizard

DAD extension (DADX) is an extension of the DB2 XML Extender document access definition (DAD) file. A DADX document specifies how to create a Web service using a set of operations that are defined by SQL statements or DAD files. DAD files are files that describe the mapping between XML documents elements and DB2 database columns. They are used by the DB2 XML Extender, which supports XML-based operations that store or retrieve XML documents to and from DB2 database.

The Web services DADX group configuration wizard enables you to create a DADX group. The DADX group created with this wizard is used to store DADX documents. A DADX group contains connection (JDBC and JNDI) and other information that is shared between DADX files.

Once you have created a DADX group, you can import DADX files and then start the Web services wizard to create a Web service from a DADX file. A DADX file is created using the XML from SQL query wizard.

Web service wizard

This wizard is used for creating Web services. It is cable of generating several types of Web services. The focus of this book is only on DADX Web services. Using the wizard, you can specify to generate a Web service proxy class. A Web service proxy class makes it very easy to invoke a Web service. You just instantiate the class and invoke the desired method on it.

Moreover, the wizard gives you the ability to generate and invoke a sample application. The sample application is a small Web application, which enables you to invoke the different methods of the Web service, and to view the results.

The generated sample contains the code to instantiate the generated Web service proxy class and to invoke the desired methods on it.

You can specify to immediately launch the sample application after the wizard. A default Websphere test environment is started, and the test client starts in a browser, and you can test the Web service.

6

RDB and XML integration

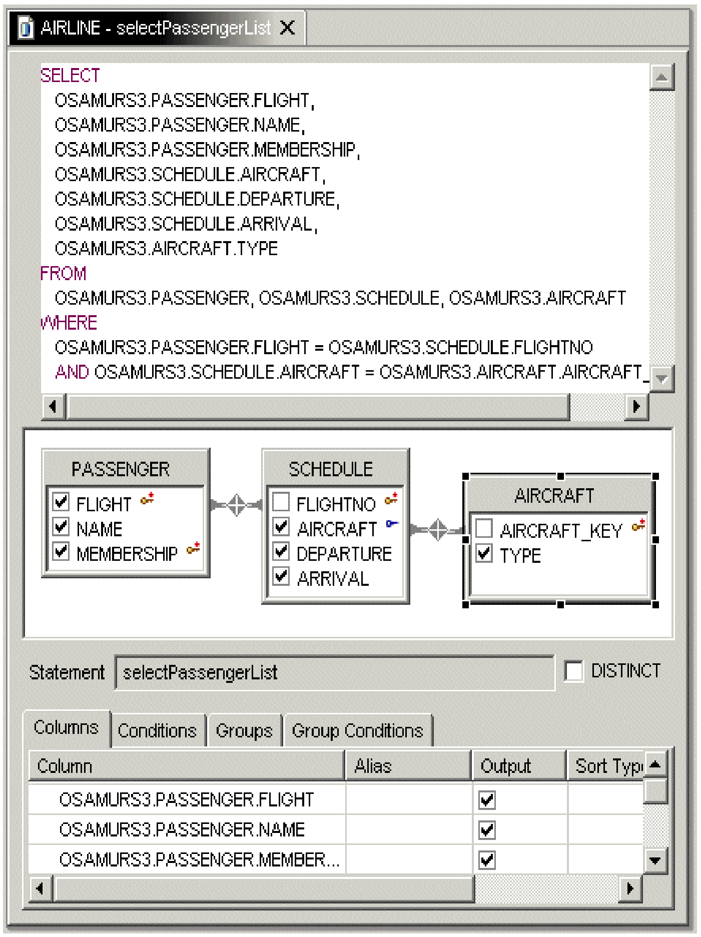
Chapter six discusses XML and SQL capabilities within Application Studio. In any application, simple or complex, a database usually acts as the data store. For any application development tool, whether developing the graphical user interface or the business logic components, an easy to use tool graphical tool that works well with a database is a must. Application Studio's tool and wizards for interacting with the database provide a multitude of capabilities.

The SQL to XML wizards gives the user the capability of generating eXtensible Style Sheet (XSL) and XML files from a SQL statement. In the chapter, a simple schema was created to demonstrate the various file generation capabilities. The files from the different options are hen compared.

The DB2 XML Extender is discussed in 6.4, "DB2 XML Extender" on page 126. A very broad overview is provided with the two methods used for storing XML data in the DB2 database. A comparison is carried between the two methods with recommendations for using each method in a set of conditions.

6.1 The SQL to XML wizards

The SQL to XML wizards are found in the data perpective. To demonstrate the wizards and the files produced, a schema and SELECT statement has to be created. The schema is shown below in Figure 6-1. It consists of three tables: PASSENGER, SCHEDULE, and AIRCRAFT. The PASSENGER and SCHEDULE tables are joined by the flight and flightNo attributes. The aircraft attribute on the SCHEDULE table is connected to the aircraft_key attribute on the AIRCRAFT table through a foreign key.



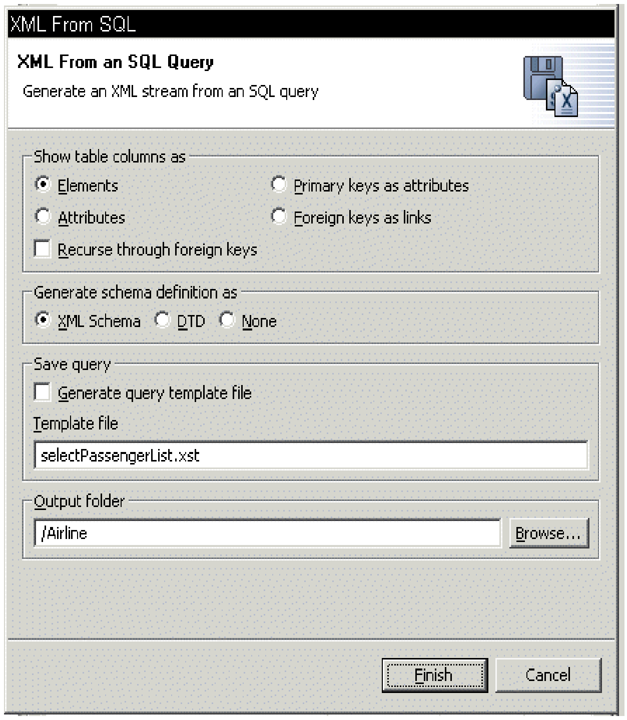
```
SELECT
  OSAMURS3.PASSENGER.FLIGHT,
  OSAMURS3.PASSENGER.NAME,
  OSAMURS3.PASSENGER.MEMBERSHIP,
  OSAMURS3.SCHEDULE.AIRCRAFT,
  OSAMURS3.SCHEDULE.DEPARTURE,
  OSAMURS3.SCHEDULE.ARRIVAL,
  OSAMURS3.AIRCRAFT.TYPE
FROM
  OSAMURS3.PASSENGER, OSAMURS3.SCHEDULE, OSAMURS3.AIRCRAFT
WHERE
  OSAMURS3.PASSENGER.FLIGHT = OSAMURS3.SCHEDULE.FLIGHTNO
  AND OSAMURS3.SCHEDULE.AIRCRAFT = OSAMURS3.AIRCRAFT.AIRCRAFT_
```

Statement: selectPassengerList

Column	Alias	Output	Sort Typ
OSAMURS3.PASSENGER.FLIGHT		✓	
OSAMURS3.PASSENGER.NAME		✓	
OSAMURS3.PASSENGER.MEMBER...		✓	

Figure 6-1 The Passenger List Select statement

To generate the XML relate files based on the Passenger List Select statement, right-clink on the **selectPassengerList** statement, and select the **Generate new XML** from the pop-up menu. The SQL to XML wizard opens as shown below:


XML From SQL

XML From an SQL Query

Generate an XML stream from an SQL query

Show table columns as: [✓ Elements] Attributes Primary keys as attributes Foreign keys as links

☐ Recurse through foreign keys

Generate schema definition as: [✓ XML Schema] DTD None

Save query

☐ Generate query template file

Template file: selectPassengerList.xst

Output folder: /Airline

Figure 6-2 The SQL to XML wizard panel

The wizard provides three options (Figure 6-2). The last Output folder is automatically filled in if there is only one project and if that project has been pre-selected. Other projects can be selected by clicking the **Browse** button.

The first option Show table columns as produces three files. To view the files, the user has to navigate through to the Navigator view. We will compare these files with the different options further below. Through the second option, `Generate schema definition as`, we can opt to produce a XML Schema file, or a DTD, or none of these.

The third option, `Save query` gives us the option to generate a query template file with the name of the template file already defined from the select statement. This name can be changed, if so desired.

Show table column as option

This option produces three files for each select statement. Accordingly, they are named after the select statement, and in our example their names are:

- The selectPassengerList.html
- The selectPassengerList.xml
- The selectPassengerList.xsl

The most important files here are the XML and XSL files, because the HTML file can also be generated from these two files. To generate the HTML file, select both of them, right-click, and from the pop-up menu, select **Apply XSL**, and then select **As HTML**.

If the Recurse through foreign keys check box is ticked, another three files are produced:

- The selectPassengerList_AIRCRAFT.html
- The selectPassengerList_AIRCRAFT.xml
- TheselectPassengerList_AIRCRAFT.xsl

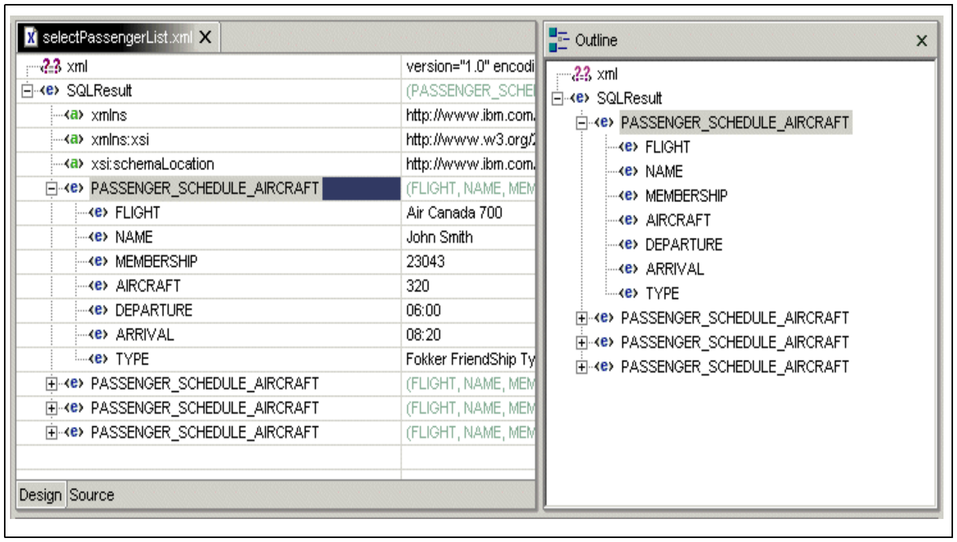
However, this only works if the `Foreign key as links` option is chosen. For the other three options, ticking this check box has no effect.

XML and XSL files

For the first option, `Show table columns as 'Elements'` implies, all table columns are generated as elements. Each element name follows the table column name. In our example, the elements are <FLIGHT>, <NAME>, <MEMBERSHIP>, <AIRCRAFT>, etc. The elements are part of the `<PASSENGER_SCHEDULE_AIRCRAFT>` element, which is a derivation from the names of the three tables. The `<SQLResult>` is the root element for all the options.

Example 6-1 XSL file for Show table columns as 'Elements'

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform exclude-result-prefixes="sqltoxml" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="html"/>
 <xsl:template match="/">
  <HTML>
   <HEAD>
    <META content="text/html; charset=iso-8859-1" http-equiv="Content-Type"/>
    <META content="0" http-equiv="Expires"/>
   </HEAD>
   <BODY>
    <DIV>
     <xsl:apply-templates/>
    </DIV>
   </BODY>
  </HTML>
  </xsl:template>
  <xsl:template match="SQLResult">
    <Table border="2">
      <TR>
       <TD>FLIGHT</TD>
        <TD>NAME</TD>
        <TD>MEMBERSHIP</TD>
        <TD>AIRCRAFT</TD>
        <TD>DEPARTURE</TD>
        <TD>ARRIVAL</TD>
        <TD>TYPE</TD>
      </TR>
      <xsl:for-each select="PASSENGER_SCHEDULE_AIRCRAFT">
        <TR>
         <TD>
           <xsl:value-of select="FLIGHT"/>
         </TD>
         <TD>
           <xsl:value-of select="NAME"/>
         </TD>
         <TD>
           <xsl:value-of select="MEMBERSHIP"/>
         </TD>
         <TD>
           <xsl:value-of select="AIRCRAFT"/>
         </TD>
         <TD>
           <xsl:value-of select="DEPARTURE"/>
         </TD>
         <TD>
           <xsl:value-of select="ARRIVAL"/>
         </TD>
         <TD>
          <xsl:value-of select="TYPE"/>
         </TD>
        </TR>
      </xsl:for-each>
    </Table>
  </xsl:template>
</xsl:transform>
```


selectPassengerList.xml

xml	version="1.0" encodi
SQLResult	(PASSENGER_SCHE
xmlns	http://www.ibm.com.
xmlns:xsi	http://www.w3.org/2
xsi:schemaLocation	http://www.ibm.com.
PASSENGER_SCHEDULE_AIRCRAFT	(FLIGHT, NAME, MEM
FLIGHT	Air Canada 700
NAME	John Smith
MEMBERSHIP	23043
AIRCRAFT	320
DEPARTURE	06:00
ARRIVAL	08:20
TYPE	Fokker FriendShip Ty
PASSENGER_SCHEDULE_AIRCRAFT	(FLIGHT, NAME, MEM
PASSENGER_SCHEDULE_AIRCRAFT	(FLIGHT, NAME, MEM
PASSENGER_SCHEDULE_AIRCRAFT	(FLIGHT, NAME, MEM

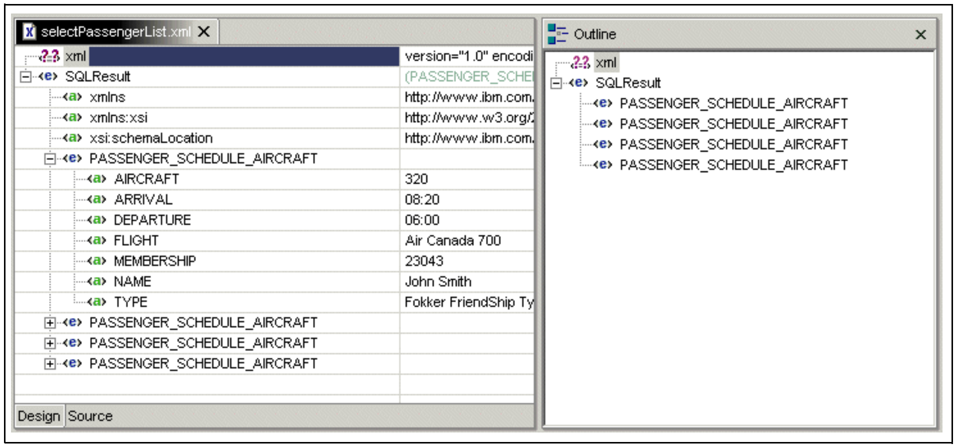
Design Source

Outline

```
xml
SQLResult
  PASSENGER_SCHEDULE_AIRCRAFT
    FLIGHT
    NAME
    MEMBERSHIP
    AIRCRAFT
    DEPARTURE
    ARRIVAL
    TYPE
  PASSENGER_SCHEDULE_AIRCRAFT
  PASSENGER_SCHEDULE_AIRCRAFT
  PASSENGER_SCHEDULE_AIRCRAFT
```

Figure 6-3 XML file generated with the Elements option

With the Attributes option all the column name are generated as attributes (Figure 6-3). The attribute name takes on the name of the column. Similarly, all the attributes are within the `<PASSENGER_SCHEDULE_AIRCRAFT>` element. (Figure 6-4).


selectPassengerList.xml

xml	version="1.0" encodi
SQLResult	(PASSENGER_SCHE
xmlns	http://www.ibm.com.
xmlns:xsi	http://www.w3.org/2
xsi:schemaLocation	http://www.ibm.com.
PASSENGER_SCHEDULE_AIRCRAFT	
AIRCRAFT	320
ARRIVAL	08:20
DEPARTURE	06:00
FLIGHT	Air Canada 700
MEMBERSHIP	23043
NAME	John Smith
TYPE	Fokker FriendShip Ty
PASSENGER_SCHEDULE_AIRCRAFT	
PASSENGER_SCHEDULE_AIRCRAFT	
PASSENGER_SCHEDULE_AIRCRAFT	

Design Source

Outline

```
xml
SQLResult
  PASSENGER_SCHEDULE_AIRCRAFT
  PASSENGER_SCHEDULE_AIRCRAFT
  PASSENGER_SCHEDULE_AIRCRAFT
  PASSENGER_SCHEDULE_AIRCRAFT
```

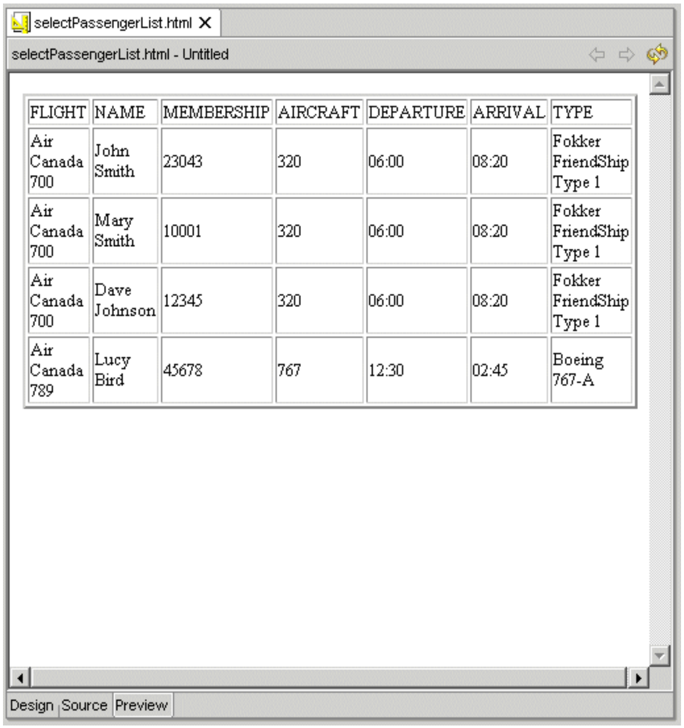
Figure 6-4 XML file generated with the Attributes option

When the‘`Primary key as attributes` option is chosen, the XML file produced has the flight and membership columns generated as attributes, and the remaining columns as elements. The attribute with the foreign key `aircraft` still comes out as an element. When the `Foreign key as links` option, this column tables still remains an element. However, with this same option, but with the Recurse through foreign keys check box ticked, this column is generated as an attribute. This option also generates a XSL and XML file for the AIRCRAFT table.

For the these files for the AIRCRAFT table, the primary key, aircraft_key, is generated as a attribute and the type column becomes an element. The root element is <SQLResult> and the <AIRCRAFT> element holding the attributes and elements is names after the table.

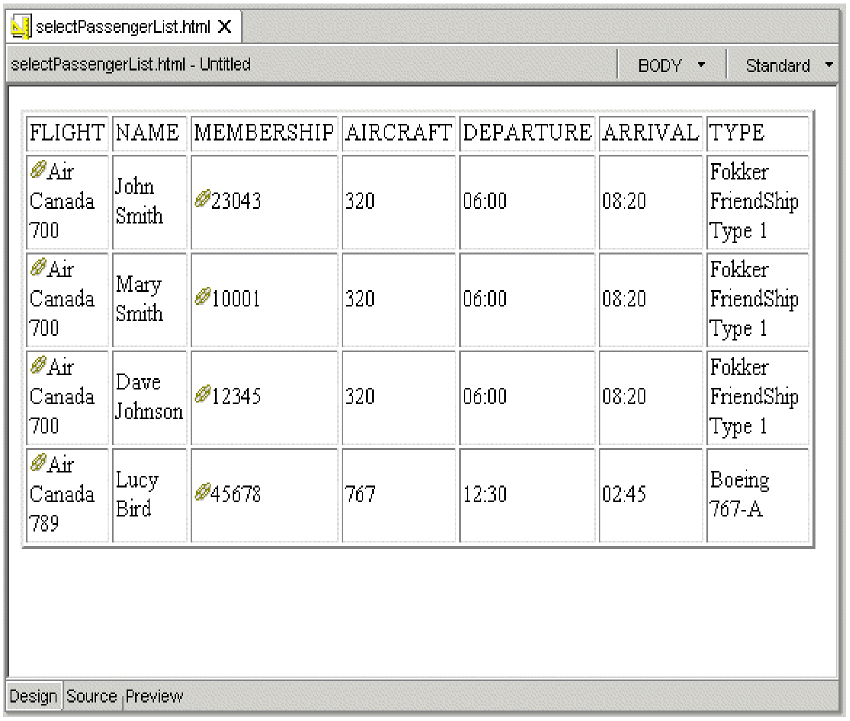
HTML file

The resulting HTML files (Figure 6-5 and Figure 6-6) generated by the first three options, (‘Elements’, ‘Attributes’ and ‘Primary keys as attributes’) produce similar results. A HTML form is produced for all the fields selected.



FLIGHT	NAME	MEMBERSHIP	AIRCRAFT	DEPARTURE	ARRIVAL	TYPE
Air Canada 700	John Smith	23043	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Mary Smith	10001	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Dave Johnson	12345	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 789	Lucy Bird	45678	767	12:30	02:45	Boeing 767-A

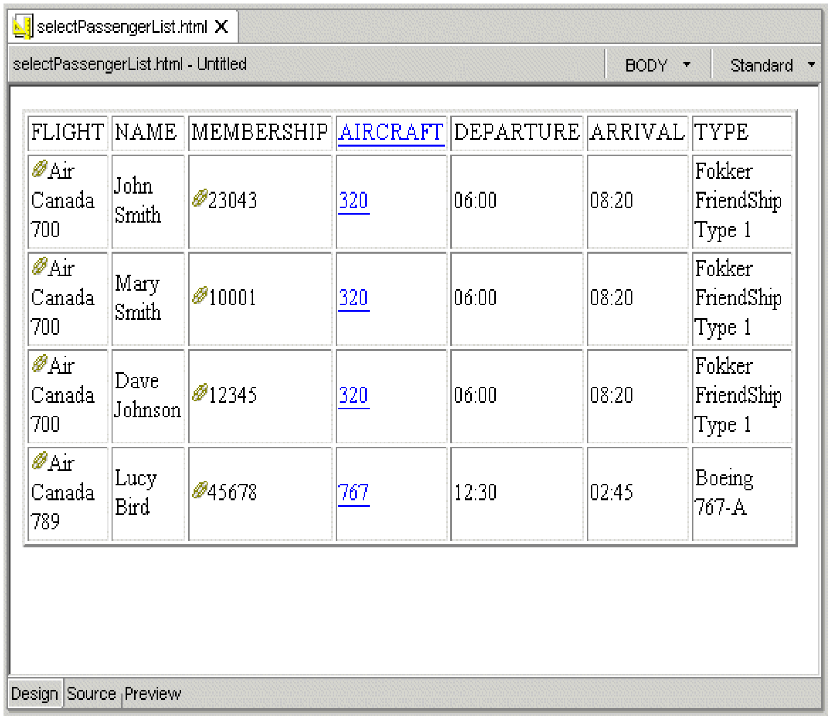
Figure 6-5 HTML for ‘Elements’, ‘Attributes’, and ‘Primary key as attributes’



selectPassengerList.html - Untitled

FLIGHT	NAME	MEMBERSHIP	AIRCRAFT	DEPARTURE	ARRIVAL	TYPE
Air Canada 700	John Smith	23043	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Mary Smith	10001	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Dave Johnson	12345	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 789	Lucy Bird	45678	767	12:30	02:45	Boeing 767-A

Figure 6-6 HTML with the 'Recurse through foreign keys' unchecked



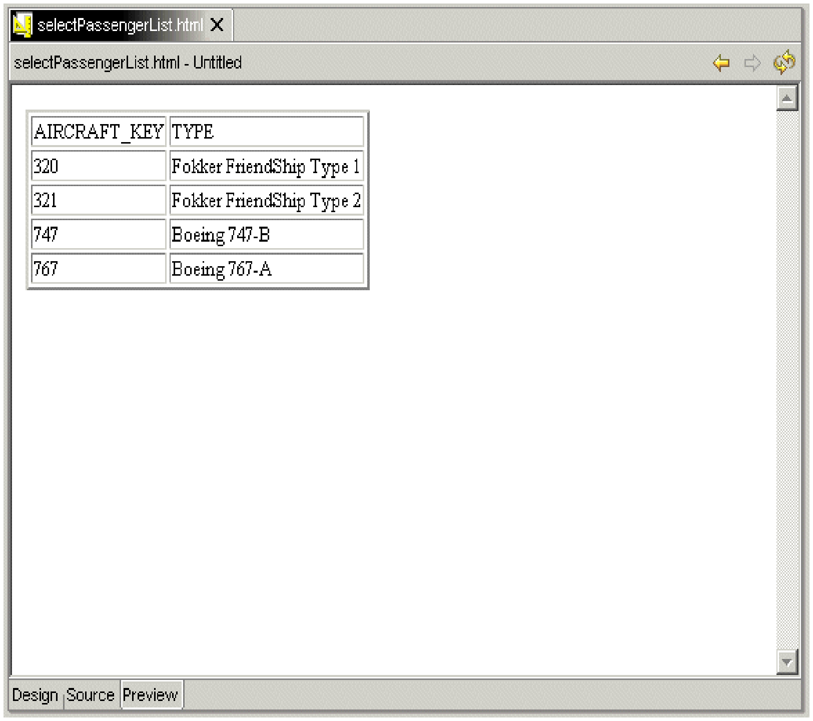
selectPassengerList.html

selectPassengerList.html - Untitled

FLIGHT	NAME	MEMBERSHIP	AIRCRAFT	DEPARTURE	ARRIVAL	TYPE
Air Canada 700	John Smith	23043	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Mary Smith	10001	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 700	Dave Johnson	12345	320	06:00	08:20	Fokker FriendShip Type 1
Air Canada 789	Lucy Bird	45678	767	12:30	02:45	Boeing 767-A

Figure 6-7 HTML generated for Foreign key as links option

With the ‘Foreign key as links’ option, the HTML form generated adds a link for every attribute which is a key (Figure 6-7). When the ‘Recurse though foreign key’ check box is ticked, the ‘aircraft’ fields also gets an HREF attribute added to the link. This link calls the AIRCRAFT HTML form as shown in Figure 6-8.



AIRCRAFT_KEY	TYPE
320	Fokker FriendShip Type 1
321	Fokker FriendShip Type 2
747	Boeing 747-B
767	Boeing 767-A

Figure 6-8 Aircraft HTML generated for 'Foreign key as links' option

XML Schema file

The XML Schema and DTD files vary depending on the option chosen on the `Show table columns as`.

The XML Schema file for the Elements option is shown below:

Example 6-2 XML Schema file for "Show table columns as 'Elements'"

```
[001] <?xml version="1.0" encoding="UTF-8"?>
[002] <schema targetNamespace="http://www.ibm.com/PASSENGER_SCHEDULE_AIRCRAFT"
    xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:PASSENGER_SCHEDULE_AIRCRAFT="http://www.ibm.com/PASSENGER_SCHEDULE_AIRCRA
FT">
[003]  <element name="SQLResult">
[004]    <complexType>
[005]      <sequence>
[006]        <element maxOccurs="unbounded" minOccurs="0"
ref="PASSENGER_SCHEDULE_AIRCRAFT:PASSENGER_SCHEDULE_AIRCRAFT"/>
[007]      </sequence>
[008]    </complexType>
```

```
    <key name="PASSENGER_SCHEDULE_AIRCRAFTPRIMKEY">
      <selector xpath="PASSENGER_SCHEDULE_AIRCRAFT:PASSENGER_SCHEDULE_AIRCRAFT"/>
      <field xpath="PASSENGER_SCHEDULE_AIRCRAFT:FLIGHT"/>
      <field xpath="PASSENGER_SCHEDULE_AIRCRAFT:MEMBERSHIP"/>
    </key>
  </element>
  <element name="PASSENGER_SCHEDULE_AIRCRAFT">
    <complexType>
      <sequence>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:FLIGHT"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:NAME"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:MEMBERSHIP"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:AIRCRAFT"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:DEPARTURE"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:ARRIVAL"/>
        <element ref="PASSENGER_SCHEDULE_AIRCRAFT:TYPE"/>
      </sequence>
    </complexType>
  </element>
  <element name="FLIGHT">
    <simpleType>
      <restriction base="string">
          <maxLength value="30"/>
      </restriction>
    </simpleType>
 </element>
  <element name="NAME" nillable="true">
    <simpleType>
      <restriction base="string">
        <maxLength value="20"/>
      </restriction>
    </simpleType>
  </element>
  <element name="MEMBERSHIP">
    <simpleType>
      <restriction base="string">
        <maxLength value="10"/>
      </restriction>
    </simpleType>
  </element>
  <element name="AIRCRAFT" nillable="true">
    <simpleType>
      <restriction base="string">
         <maxLength value="10"/>
      </restriction>
    </simpleType>
  </element>
  <element name="DEPARTURE" nillable="true">
```

```
[056]    <simpleType>
[057]      <restriction base="string">
[058]        <maxLength value="10"/>
[059]      </restriction>
[060]    </simpleType>
[061]  </element>
[062]  <element name="ARRIVAL" nillable="true">
[063]    <simpleType>
[064]      <restriction base="string">
[065]        <maxLength value="10"/>
[066]      </restriction>
[067]    </simpleType>
[068]  </element>
[069]  <element name="TYPE" nillable="true">
[070]    <simpleType>
[071]      <restriction base="string">
[072]        <maxLength value="30"/>
[073]      </restriction>
[074]    </simpleType>
[075]  </element>
[076]</schema>
```

The first element generated is `SQLResult` is the complex type. Appropriately, it can have a minimum of no records with the upper-limit being unbounded. In the 'key element the two attributes, 'flight' and 'membership', that make up the key are declared within the 'field' element (line 11 and 12).

Every column name has been declared as an <simpleType> element, each of type string and a max length (lines 27 to 75). This has been declared a <restriction> element. All columns, besides 'flight' and 'membership' columns, are declared as nillable.

The element <PASSENGER_SCHEDULE_AIRCRAFT> has been declared as a complexType which hold all the other elements and to be in the order specified (lines 16 to 25).

The XML Schema file generated through the 'Attributes' option is tabulated below:

Example 6-3 XML Schema file for Show table columns as 'Attributes'

```
[001]<?xml version="1.0" encoding="UTF-8"?>
[002]<schema targetNamespace="http://www.ibm.com/PASSENGER_SCHEDULE_AIRCRAFT"
    xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:PASSENGER_SCHEDULE_AIRCRAFT="http://www.ibm.com/PASSENGER_SCHEDULE_AIRCRA
FT">
[003]    <element name="SQLResult">
[004]        <complexType>
```

```
            <sequence>
                <element maxOccurs="unbounded" minOccurs="0"
ref="PASSENGER_SCHEDULE_AIRCRAFT:PASSENGER_SCHEDULE_AIRCRAFT"/>
            </sequence>
        </complexType>
        <key name="PASSENGER_SCHEDULE_AIRCRAFTPRIMKEY">
            <selector
xpath="PASSENGER_SCHEDULE_AIRCRAFT:PASSENGER_SCHEDULE_AIRCRAFT"/>
            <field xpath="@FLIGHT"/>
            <field xpath="@MEMBERSHIP"/>
        </key>
    </element>
    <element name="PASSENGER_SCHEDULE_AIRCRAFT">
        <complexType>
            <attribute name="FLIGHT">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="30"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="NAME">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="20"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="MEMBERSHIP">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="10"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="AIRCRAFT">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="10"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="DEPARTURE">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="10"/>
                    </restriction>
                </simpleType>
            </attribute>
```

```
[052]            <attribute name="ARRIVAL">
[053]                <simpleType>
[054]                    <restriction base="string">
[055]                        <maxLength value="10"/>
[056]                    </restriction>
[057]                </simpleType>
[058]            </attribute>
[059]            <attribute name="TYPE">
[060]                <simpleType>
[061]                    <restriction base="string">
[062]                        <maxLength value="30"/>
[063]                    </restriction>
[064]                </simpleType>
[065]            </attribute>
[066]        </complexType>
[067]    </element>
[068]</schema>
```

In contrast to the previous example, every column name is now declared as an attribute. The `<restriction>` element is still applied every column, but this time all the attributes do not have the ‘nullable’ attribute set to true. All the attributes are within a complexType element ‘PASSENGER_SCHEDULE_AIRCRAFT’.

Also, in the <key> element, this time, the field attributes are declared as ‘@FLIGHT and @MEMBERSHIP, (lines 11 and 12 in Example 5-3) as compared to "PASSENGER_SCHEDULE_AIRCRAFT:FLIGHT" and "PASSENGER_SCHEDULE_AIRCRAFT:MEMBERSHIP" (lines 11 and 12 in Example 5-2).

Choosing option ‘Primary key as attributes’ will produce an XML Schema file which is a combination of the previous two examples. The flight and membership columns are generated as attributes, with the remaining fields produced as elements. The element ‘PASSENGER_SCHEDULE_AIRCRAFT’ now holds a complexType containing the elements with two attributes.

With the ‘Foreign Key as links’ option and the ‘Recurse through Foreign keys’ check box ticked, the AIRCRAFT column now becomes an attribute instead of an element. With that it is removed as element from the complexType element ‘PASSENGER_SCHEDULE_AIRCRAFT’. Also an XML Schema file is also produced for the AIRCRAFT table, which is produced here:

Example 6-4   XML Schema for AIRCRAFT table

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ibm.com/AIRCRAFT"
    xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:AIRCRAFT="http://www.ibm.com/AIRCRAFT">
```

```
    <element name="SQLResult">
        <complexType>
            <sequence>
                <element maxOccurs="unbounded" minOccurs="0"
ref="AIRCRAFT:AIRCRAFT"/>
            </sequence>
        </complexType>
        <key name="AIRCRAFTPRIMKEY">
            <selector xpath="AIRCRAFT:AIRCRAFT"/>
            <field xpath="@AIRCRAFT_KEY"/>
        </key>
    </element>
    <element name="AIRCRAFT">
        <complexType>
            <sequence>
                <element ref="AIRCRAFT:TYPE"/>
            </sequence>
            <attribute name="AIRCRAFT_KEY">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="10"/>
                    </restriction>
                </simpleType>
            </attribute>
        </complexType>
    </element>
    <element name="TYPE" nillable="true">
        <simpleType>
            <restriction base="string">
                <maxLength value="30"/>
            </restriction>
        </simpleType>
    </element>
</schema>
```

DTD file

The Schema definition can also be generated as a DTD. Here we will compare the DTD generated by each of the options. With the Show table columns as 'Elements' option, the following DTD is generated:

Example 6-5 DTD generated by the Show table as 'Elements' option

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SQLResult (PASSENGER_SCHEDULE_AIRCRAFT)*>
<!ELEMENT PASSENGER_SCHEDULE_AIRCRAFT
(FLIGHT,NAME,MEMBERSHIP,AIRCRAFT,DEPARTURE,ARRIVAL,TYPE) >
```

```
<!ELEMENT FLIGHT (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT MEMBERSHIP (#PCDATA)>
<!ELEMENT AIRCRAFT (#PCDATA)>
<!ELEMENT DEPARTURE (#PCDATA)>
<!ELEMENT ARRIVAL (#PCDATA)>
<!ELEMENT TYPE (#PCDATA)>
```

As expected, all column names have been generated as elements. Comparing DTDs with XML Schemas, show how verbose and versatile XML Schemas can be.

When the DTD is produced through the “Show table as ‘Attributes’” the column names are produced as attributes.

Example 6-6 DTD generated by the Show table as 'Attributes' option

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SQLResult (PASSENGER_SCHEDULE_AIRCRAFT)*>
<!ELEMENT PASSENGER_SCHEDULE_AIRCRAFT EMPTY>
<!ATTLIST PASSENGER_SCHEDULE_AIRCRAFT
  FLIGHT CDATA #REQUIRED
  NAME CDATA #REQUIRED
  MEMBERSHIP CDATA #REQUIRED
  AIRCRAFT CDATA #REQUIRED
  DEPARTURE CDATA #REQUIRED
  ARRIVAL CDATA #REQUIRED
  TYPE CDATA #REQUIRED
>
```

When the 'Primary keys as attributes' option is selected, the 'flight' and 'membership' columns are generated as attributes, the remaining columns are produced as elements.

With the 'Foreign key as links' option, the aircraft column is also generated as an attribute. Additionally, an selectPassengerList_AIRCRAFT.dtd file is generated, which is shown below:

Example 6-7 DTD for AIRCRAFT table with the Show tables as Foreign keys as links

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT SQLResult (AIRCRAFT)*>
<!ELEMENT AIRCRAFT (TYPE) >
<!ATTLIST AIRCRAFT
  AIRCRAFT_KEY ID #REQUIRED
>
<!ELEMENT TYPE (#PCDATA)>
```

The Query template file

To generate the Query template file, the user needs to tick the 'Generate query template file in the Save query box as can be seen in Figure 5-2.

Example 6-8 The Query template file for Passenger List select

```
<?xml version="1.0" encoding="UTF-8"?>
<SQLGENERATEINFORMATION>
  <DATABASEINFORMATION>
    <LOGINID>osamurs3</LOGINID>
    <PASSWORD><![CDATA[osamurs3]]></PASSWORD>
    <JDBCDRIVER>COM.ibm.db2.jdbc.app.DB2Driver</JDBCDRIVER>
    <JDBCSERVER>jdbc:db2:AIRLINE</JDBCSERVER>
  </DATABASEINFORMATION>
  <STATEMENT>
    <![CDATA[ SELECT OSAMURS3.PASSENGER.FLIGHT,
          OSAMURS3.PASSENGER.NAME,
          OSAMURS3.PASSENGER.MEMBERSHIP,
          OSAMURS3.SCHEDULE.AIRCRAFT,
          OSAMURS3.SCHEDULE.DEPARTURE,
          OSAMURS3.SCHEDULE.ARRIVAL,
          OSAMURS3.AIRCRAFT.TYPE
          FROM OSAMURS3.PASSENGER,
             OSAMURS3.SCHEDULE,
             OSAMURS3.AIRCRAFT
          WHERE OSAMURS3.PASSENGER.FLIGHT =
OSAMURS3.SCHEDULE.FLIGHTNO
          AND OSAMURS3.SCHEDULE.AIRCRAFT = OSAMURS3.AIRCRAFT.AIRCRAFT
          ORDER BY FLIGHT, NAME ]]>
  </STATEMENT>
    <OPTIONS>
    <FORMATOPTION>GENERATE_AS_ELEMENTS</FORMATOPTION>
    <RECURSE>FALSE</RECURSE>
  </OPTIONS>
</SQLGENERATEINFORMATION>
```

This file provides database connection information within the <DATABASEINFORMATION> element and the SQL statement for the SQL Query within <STATEMENT> element.

With any of the ‘Show table columns as’ options, the contents of the file do not change except for the <FORMATOPTION> element. If the ‘Elements’ option is chosen, this element is produced as shown in the example above. For the ‘Attributes’ option, the element has contents: ‘`GENERATE_AS_ATTRIBUTES`. Similarly, it has ‘`GENERATE_PRIMARYKEYS_AS_ATTRIBUTES`’ for the ‘Primary Key as attributes’ option. When the ‘Foreign keys as links’ option is selected, this element has contents: ‘`GENERATE_ID_AND_IDREF`’ and depending if the ‘Recurse through foreign keys’ is checked on not, the `<RECURSE>` element has a value set to either ‘TRUE’ or ‘FALSE’.

The Query Template file can be used to execute SQL statements at runtime. Application Developer comes with a SQLtoXML Java class library, which can be used in an application or servlet to execute SQL statements and produce results as XML. A example servlet, XMLIntegratorServerlet, has been provided, which can be used as a sample for an application.

These references are available:

`http://www7b.software.ibm.com/wsdd/techjournal/0202_haggarty/haggarty.html`

http://www7b.software.ibm.com/wsdd/techjournal/0204_russell/russell.html
http://www.xml.com/pub/a/2000/11/29/schemas/structuresref.html

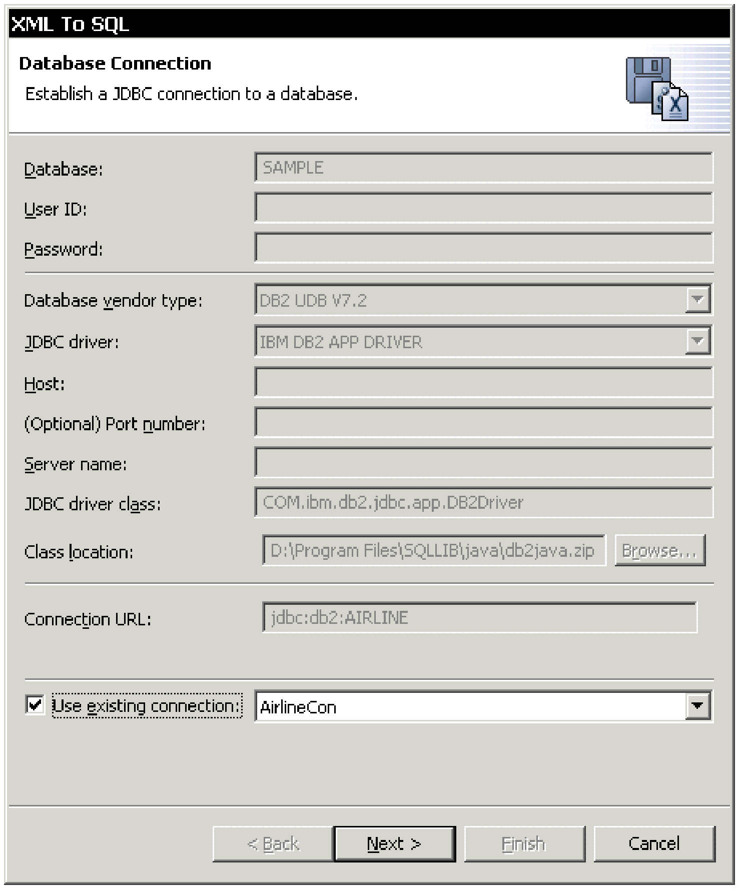
6.2 The XML to SQL wizard

The XML to SQL wizard enables to insert or update the database record using an XML. It works as the opposite way to SQL to XML wizard and this wizard is available on Application Developer V5 above. Example 6-9 is showing the result xml from SQL to XML wizard. This xml is representing a record of the customer database and we will insert new record into the database.

Example 6-9 Customer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SQLResult SYSTEM "Customer.dtd">
<SQLResult>
    <CUSTOMER>
        <FIRSTNAME>Osamu</FIRSTNAME>
        <LASTNAME>Takagiwa</LASTNAME>
        <SERIAL>6287</SERIAL>
        <EMAIL>osamu@itso.com</EMAIL>
    </CUSTOMER>
</SQLResult>
```

Go to the navigator view of the XML Perspective, then right-click on the customer.xml or the XML, which is representing the customer record on your workbench. Select **Generate -> Database data** and open the XML to SQL wizard window (Figure 6-9). We already defined the connection to the airline database, so we will reuse it by clicking the use existing connection box, then click **Next.**


XML To SQL

Database Connection

Establish a JDBC connection to a database.

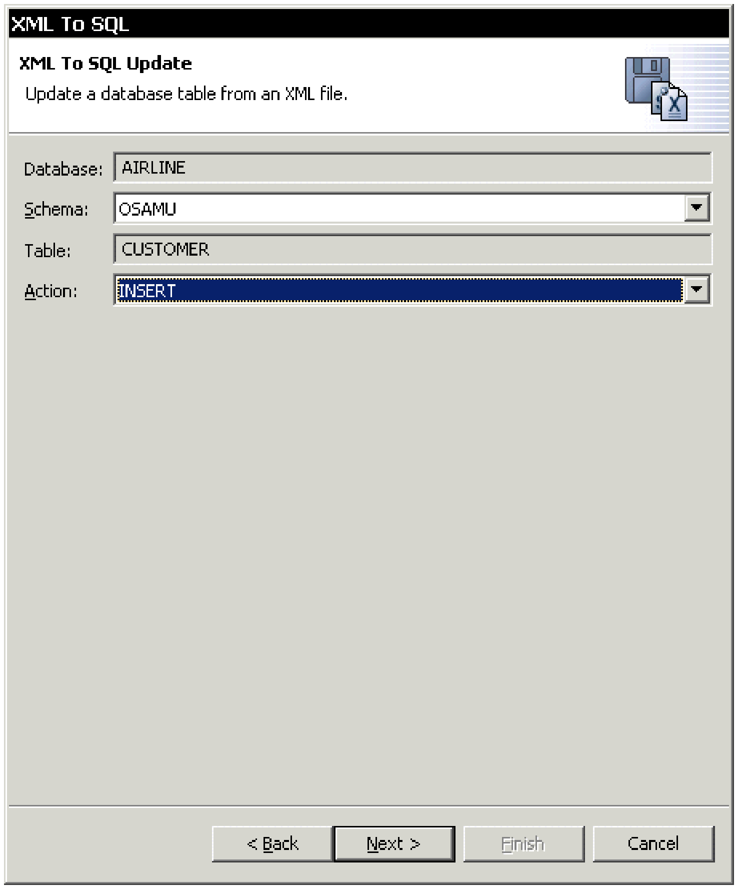
Database:	SAMPLE
User ID:	
Password:	
Database vendor type:	DB2 UDB V7.2
JDBC driver:	IBM DB2 APP DRIVER
Host:	
(Optional) Port number:	
Server name:	
JDBC driver class:	COM.ibm.db2.jdbc.app.DB2Driver
Class location:	D:\Program Files\SQLLIB\java\db2java.zip
Connection URL:	jdbc:db2:AIRLINE
✓ Use existing connection:	AirlineCon

Figure 6-9 XML to SQL wizard - Selecting the connection

The schema list is generated by the connection object and the table name is automatically recognized from the XML file.

Note: The XML to SQL always picks the second child element from the XML.

The action can be Insert or Update. We are going to insert the new record, so select the **Insert** (Figure 6-10) then click **Next.**



XML To SQL

XML To SQL Update

Update a database table from an XML file.

Database: AIRLINE

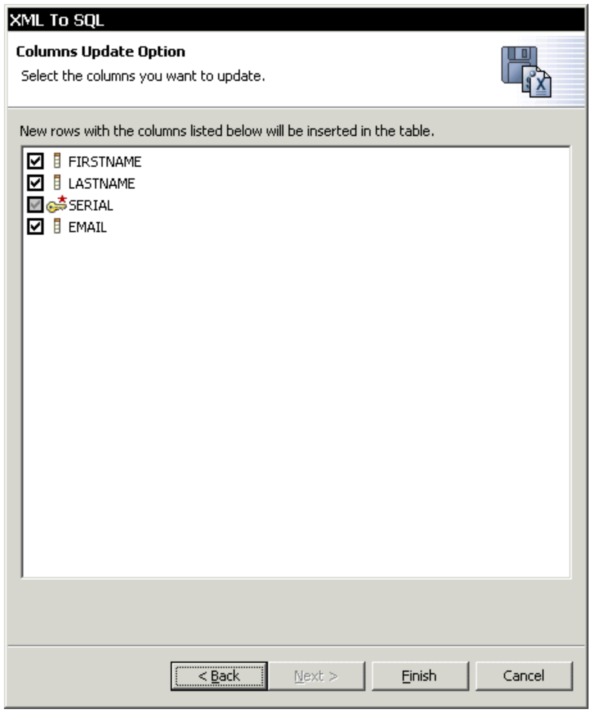
Schema: OSAMU

Table: CUSTOMER

Action: INSERT

Figure 6-10 XML to SQL wizard: Selecting Schema and Action

Since the serial is a key, you cannot de-select it. The other columns, you can de-select if you want (Figure 6-11). Then click **Finish** to insert. Using the update action, you can update the record.


XML To SQL

Columns Update Option

Select the columns you want to update.

New rows with the columns listed below will be inserted in the table.

- [✓] FIRSTNAME
- [✓] LASTNAME
- [✓] SERIAL
- [✓] EMAIL

Figure 6-11 XML to SQL wizard: Selecting the column

6.3 The DDL to XML Schema wizard

From a table definition, it is possible to generate an XML Schema. In the Data perpective, from the Data Definition view, navigate to the table that you want to the XML Schema from. Right-click on the table, and the pop-up menu should present an option to Generate XML Schema (Figure 6-12).

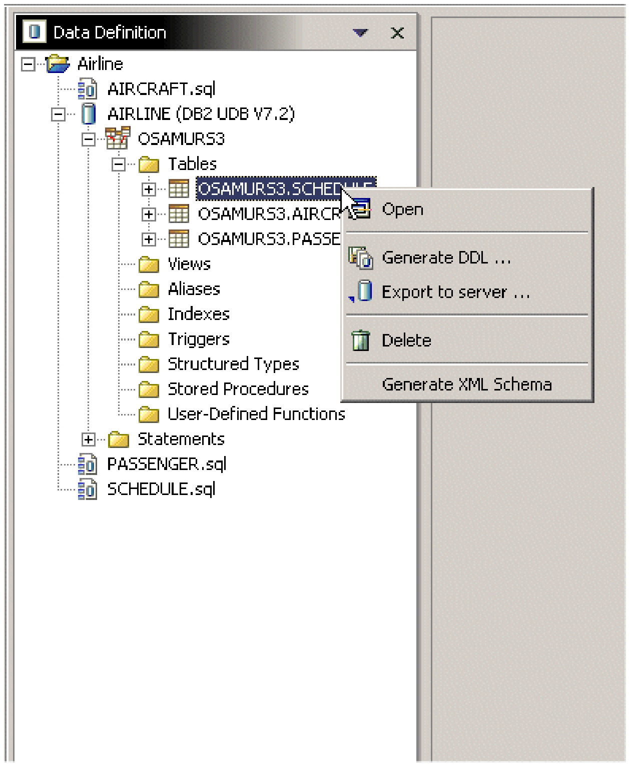


Figure 6-12 Generate XML Schema

The XML Schema generated for the SCHEDULE table is shown below:

Example 6-10 XML Generated through the DDL to XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.ibm.com/AIRLINE/OSAMURS3"
xmlns:AIRLINEOSAMURS3="http://www.ibm.com/AIRLINE/OSAMURS3">
<element name="SCHEDULE" type="AIRLINEOSAMURS3:SCHEDULE">
        <key name="SCHEDULEPRIMKEY">
            <selector xpath="AIRLINEOSAMURS3:SCHEDULE"/>
            <field xpath="FLIGHTNO"/>
        </key>
    </element>

    <complexType name="SCHEDULE">
        <sequence>
            <element name="FLIGHTNO">
                <simpleType>
```

```
                    <restriction base="string">
                        <length value="30"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="AIRCRAFT">
                <simpleType>
                    <restriction base="string">
                        <length value="10"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="DEPARTURE">
                <simpleType>
                    <restriction base="string">
                        <length value="10"/>
                    </restriction>
                </simpleType>
            </element>
            <element name="ARRIVAL">
                <simpleType>
                    <restriction base="string">
                        <length value="10"/>
                    </restriction>
                </simpleType>
            </element>
        </sequence>
    </complexType>
</schema>
```

For the tables that we have used in our example, the schemas generated are quite simple. Here we generated a schema for the SCHEDULE table. The main element is named as SCHEDULE after the table, and our key here is just the flightNo. All the attributes within the table are held within a complexType element, the attributes themselves have been generated as elements. Notice that a <sequence> element has not been enforced on the order of the columns in the XML Schema.

In this example, the foreign key from the Aircraft column to the AIRCRAFT table has not been given any extra attention. It appears just like another column in the table. When a XML Schema is generated for the AIRCRAFT table, the XML Schema does not show any relationship with the SCHEDULE XML Schema.

An alternative to this method of generating a schema from a table would be to use a select statement. Looking at Figure 5.1, showing the Passenger List statement, one could create a select statement just on the SCHEDULE table.

From there, using the XML from SQL wizard, the user has a few options on how to create the XML Schema. The example below was created with the Show tables as Primary keys as attributes:

Example 6-11 XML Schema for a single table using a Select

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.ibm.com/SCHEDULE"
xmlns:SCHEDULE="http://www.ibm.com/SCHEDULE">
    <element name="SQLResult">
        <complexType>
            <sequence>
                <element maxOccurs="unbounded" minOccurs="0"
ref="SCHEDULE:SCHEDULE"/>
            </sequence>
        </complexType>
        <key name="SCHEDULEPRIMKEY">
            <selector xpath="SCHEDULE:SCHEDULE"/>
            <field xpath="@FLIGHTNO"/>
        </key>
    </element>
    <element name="SCHEDULE">
        <complexType>
            <sequence>
                <element ref="SCHEDULE:AIRCRAFT"/>
                <element ref="SCHEDULE:DEPARTURE"/>
                <element ref="SCHEDULE:ARRIVAL"/>
            </sequence>
            <attribute name="FLIGHTNO">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="30"/>
                    </restriction>
                </simpleType>
            </attribute>
        </complexType>
    </element>
    <element name="AIRCRAFT" nillable="true">
        <simpleType>
            <restriction base="string">
                <maxLength value="10"/>
            </restriction>
        </simpleType>
    </element>
    <element name="DEPARTURE" nillable="true">
        <simpleType>
            <restriction base="string">
                <maxLength value="10"/>
            </restriction>
        </simpleType>
    </element>
    <element name="ARRIVAL" nillable="true">
        <simpleType>
            <restriction base="string">
                <maxLength value="10"/>
            </restriction>
        </simpleType>
    </element>
</schema>
```

The main difference here is that the primary key is now generated as an attribute and a sequence has been forced on the remaining columns and they can be set to null.

6.4 DB2 XML Extender

This section has been adapted from Chapter 9 of the redbook *DB2 e-business Guide* (SG2-6539).

There are different DB2 Extender products available to provide the functions to support e-Business requirements that include support for different media types, full text search capability, fast search capability to be used over the Internet, and support for XML data. These are:

- XML Extender
- Text Extender
- Net Search Extender
- Audio Image Video (AIV) Extender

The basic idea behind all the DB2 extenders is that they provide the means to support a new data type. That is, a table column can hold special type of data, such as a text document. In order to support the new data type, the extenders define user defined types, user defined functions, stored procedures, and a new set of tables. The Extenders also exploit DB2's support for large objects of up to 2 gigabytes, and use DB2 triggers to provide integrity checking across database tables ensuring the referential integrity of the data.

A User-Defined Type (UDT) is a way to create a new data type that has its own semantics based on existing built-in types. For example, the XML Extender creates the XMLCLOB UDT from the existing CLOB (character large object) data type to support XML data in DB2. The CLOB data type is itself derived from the LOB (large object) data type.

A User-Defined Function, *UDF***,** is a way to create functions that can be used within an SQL statement, and thus add to the set of built-in SQL functions supplied with DB2. For example, the XML Extender creates the *XMLCLOBFromFile* UDF to insert XML data from a file into an XMLCLOB column.

XML Extender

The XML Extender, extends the capability of DB2 to work with data in XML documents. This means that you can store, update, and retrieve XML documents from DB2 tables with the help of the XML Extender.

A summary of the facilities that the XML Extender provides:

- Stores and retrieves *as-is* XML documents in DB2 table columns
- Stores the elements and attributes of an XML document in separate tables, or separates columns within the same table
- Composes an XML document with data that is residing in existing DB2 tables.

The XML Extender performs a series of tasks in order to enable a DB2 database and table for XML data. These tasks include:

- Create a set of tables for its own use
- Create User Defined Types (UDT) to accommodate XML data in table columns
- Create Used Defined Functions (UDF) and stored procedures (SP) to manipulate XML data
- Create triggers to maintain the data integrity for the XML data spread across multiple tables

Figure 6-13 shows an overview of the XML Extender.

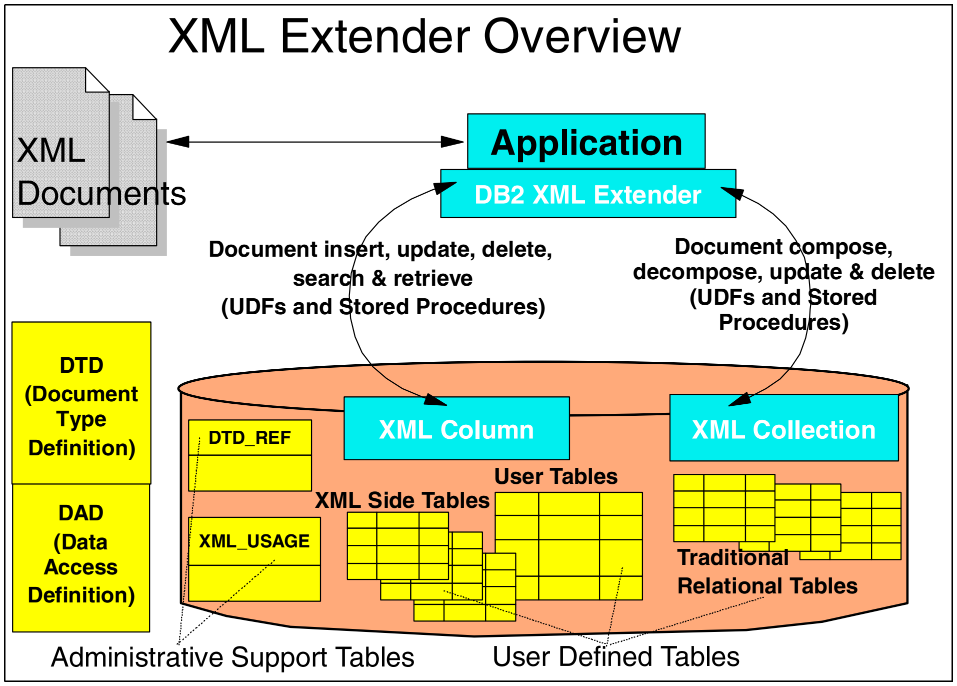

XML Extender Overview
XML Documents
Application
DB2 XML Extender
Document insert, update, delete, search & retrieve (UDFs and Stored Procedures)
Document compose, decompose, update & delete (UDFs and Stored Procedures)
DTD (Document Type Definition)
DAD (Data Access Definition)
DTD_REF
XML_USAGE
XML Column
XML Collection
User Tables
XML Side Tables
Traditional Relational Tables
Administrative Support Tables
User Defined Tables

Figure 6-13 XML Extender overview

Administrative support tables, UDTs, and UDFs

The XML Extender creates the following set of administrative support tables (Table 6-1), UDTs (Table 6-2), UDFs (Table 6-3) and stored procedures (Table 6-4) when a database is enabled for XML.

Table 6-1 .Administrative support tables created

Table Name	Description
XMLUsage	Stores the Document Access Definition (DAD) files
DTD_Ref	Stores the DTDs used to vaildate XML documents

The schema qualifier for all the tables, UDFs and stored procedures created is *DB2XML*.

Table 6-2 UDTs created

UDT	Description
XMLVarchar	Stores small XML documents with base type VARCHAR (3000 bytes)

UDT	Description
XMLCLOB	Stores large XML documents with the base type CLOB, up to 2 GIG
XMLFile	References an external XML document file using the base type VARCHAR(512)

This means that you will have to provide the qualifier for each reference to a UDT, UDF, table or stored procedure created by the extender.

Table 6-3 UDFs created

UDF	Description
`XMLVarcharFromFile`	`Import XML document from a file to a XMLVarchar column`
`XMLCLOBFromFile`	`Import XML document from a file to a XMLCLOB column`
`XMLFileFromVarchar()`	`Import XML document from a XMLVarchar column to XML file`
`XMLFileFromCLOB()`	`Import XML document from a CLOB column to XML file`
`Content()`	`Export XML document from XMLVarchar, XMLCLOB or XMLFile`
`extract functions`	`For example, find an element/attribute within an XML document that only has a single value`
`Update()`	`Replace XML document stored in XML columns by changing some element or attribute value`

For example, if you want to refer the data type XMLVarchar in a SQL statement, you will refer it as DB2XML.XMLVarchar.

Table 6-4 Stored Procedures created

Stored Procedure	Description
dxxEnableDB()	Enable database for XML
dxxDisableDB()	Disable database for XML
dxxEnableColumn()	Enable table column for XML data
dxxDisableColumn()	Disable table column for XML data
dxxEnableCollection()	Enables a collection of tables for XML
dxxDisableCollection()	Disables an enabled collection

Stored Procedure	Description
dxxGenXML()	uses a DAD file for an XML collection to compose XML documents
dxx.RetrieveXML	uses an XML enabled collection to compose XML documents
dxxShredXML()	uses a DAD file for an XML collection to decompose XML documents
dxxInsertXML()	uses an XMLenabled collection to decompose XML documents

DTD Repository

The *DTD Repository* is a table created by the XML Extender when a database is enabled for XML. The table name is *DTD_REF*. Each row of this table represents a DTD with additional metadata information. Users can access this table to insert their own DTDs. The DTDs in this table are used when the validation of an XML document is requested.

The DTD_REF table has the following columns:

- DTDID: This column specifies the location of the DTD file that you will insert into the DTD_REF table. For example, our DTD file '*redbook.DTD*' is located in the directory `x:\redbooks\SG246586\code\redbook.dtd`, where *x* is the local hard-disk drive letter.
- CONTENT: This column contains the actual DTD file content. To insert data into the XML enabled column of a table, (depending on the data type you chose for the table column to store the XML data) the insert parameter will be:

  ```
  db2xml.XMLClobFromFile('x:\redbooks\SG246586\code\redbook.dtd')
  ```

  Here *db2xml* is the schema qualifier for the function *XMLClobFromFile*. The full path to the dtd file, required by the function, is specified in parenthesis. We assumed that if you chose *XMLCLOB* data type for the table column in which you will store the XML data.
- USAGE_COUNT: this column indicates the number of XML documents for which this DTD is being used.
- The column names AUTHOR, CREATOR, UPDATOR are self-explanatory.

The DTD_REF table is an administrative support table created by XML Extender for its own use. The other administrative support table is the XML_USAGE table. This table maintains the *Document Access Definition* (DAD) files. We will now describe what a DAD file is and how it is used.

Document Access Definitions (DAD)

The Document Access Definitions file, itself an XML formatted document, is used to associate the XML document structure to a DB2 database. Basically, it provides the mapping between the elements or attributes of an XML document and the table columns, and the details of how a request for an XML document is to be handled.

The DAD file answers questions like:

- If the document is to stored in the database, then will it be stored *as is*, that is, the complete document in a table column or not? If so, then do you need any indexes built on the elements/attributes that are most frequently used as search parameters?
- Do you need to break up the elements/attributes of the xml document and keep them in separate tables? If so, then which tables will participate in the decomposition process?
- Do you want to compose an xml document from data in DB2 tables? If so, then the tables that will provide the elements/attributes and so on.

The structure of the DAD file depends on whether you are defining an XML Column or Collection. For now, we will mention that the XML Column stores the entire XML document *as is*, while the XML Collection takes the elements or attributes from the XML document and stores them in separate tables or columns.

The DAD file conforms to the DTD provided in the `DAD.dtd` file is located in the *dtd* subdirectory of the XML Extender install directory. In the case of Windows NT or Windows 2000, it is "`<drive-letter>:\<installation path>\dxx\dtd\DAD.dtd`".

The XML Extender manages the DAD files with the administrative support table XML_USAGE. This table is created when a database is enabled for XML. For example, the DAD file that maps the element 'title' from the DTD in Example 5-2 to a table column using the XML Column method is shown in Example 5-3:

Example 6-12 DTD representation of a book

```
<!ELEMENT redbook (title, subject, pubdate, leader, author*)>
<!ATTLIST redbook isbn ID CDATA #required>
<!ELEMENT title (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT pubdate (#PCDATA)>
<!ELEMENT leader (#PCDATA)>
<!ELEMENT author (authorname, authorloc)>
<!ELEMENT authorname (#PCDATA)>
<!ELEMENT authorloc (#PCDATA)>
```

Example 6-13 DAD file that maps the element 'title' from the DTD

```
<Xcolumn>
   <table name="redbook_title_sidetab">
      <column name="title"
            type="varchar(100)"
            path="/redbook/title"
            multi_occurence="NO"
      </column>
   </table>
</Xcolumn>
```

We can see the following tags in the DAD file shown in Example 5-3:

- *<Xcolumn>*, this tag indicates that we are using the XML Column method
- *<table>,* this tag identifies the side table that we want to create. A *side table* is a table that is created if you want to index an element of the XML document. In Example , we want to index the element 'title' so that we can later search for the XML document by title. The side table *redbook_title_sidetab* will be created with a column called title, and will have an index pointing to the exact location of the XML document in which the title exists
- *<column>*, this tag specifies the columns that the side table will contain. The number of column tags in the DAD file will depend on the elements or attributes of the XML document that you want to index.

  We also see that the column tag has four attributes:

  – name: This attribute specifies the name of the column in the side table.

  – path: This attribute indicates the location path in the XML document for each element or attribute.

  – type: This attribute indicates the data type of the table column that will be used to store the element or attribute.

  – multi-occurrence: This attribute indicates whether the element or attribute referred to by the path attribute can occur more than once in the XML document. In Example , the column tag 'title' has multi-occurrence set to "NO". This means that, the book will have only one title. All the column tags that have multi-occurrence set to "NO" can be mapped to the table columns within a single DB2 table.

    If multi-occurrence is "YES" for a specific column tag, then there can be only one column tag within that table tag. For each element or attribute that has a multi-occurrence value of "YES", we need to declare a separate a table with only one column that corresponds with this "multi-occurrence" element/attribute (see "XML Column method" next).

XML Column method

The *XML Column method* provides the facility to store the entire XML document *as is* in a table column. The XML documents are inserted into table columns that are enabled for XML and can be updated, retrieved, and searched. The XML document can also be kept on the local file system and only a pointer to the document be kept in the table column.

If you want to search the XML document based on certain element or attribute values, you can map the element and attribute data from an XML document to DB2 tables called *side tables*. We can build indexes on these side tables to support fast structural search. Figure 5-6 gives an overview of the column method.

When to use XML Column method

We recommend using the XML Column method under the following conditions:

- The XML documents already exist.
- There is a requirement to archive documents. For example, a news publishing company that serves articles over the Web may want to maintain an archive of published articles, which the users can search for.
- Generally read-only documents
- Stores documents externally, but use DB2 for management and search.
- Requirement for range-based searches on document elements.
- Documents with large text blocks (no larger than 2 GB, in which case the table column cannot accommodate the document and it has to be decomposed into more columns) which require structured searches using Text Extender.

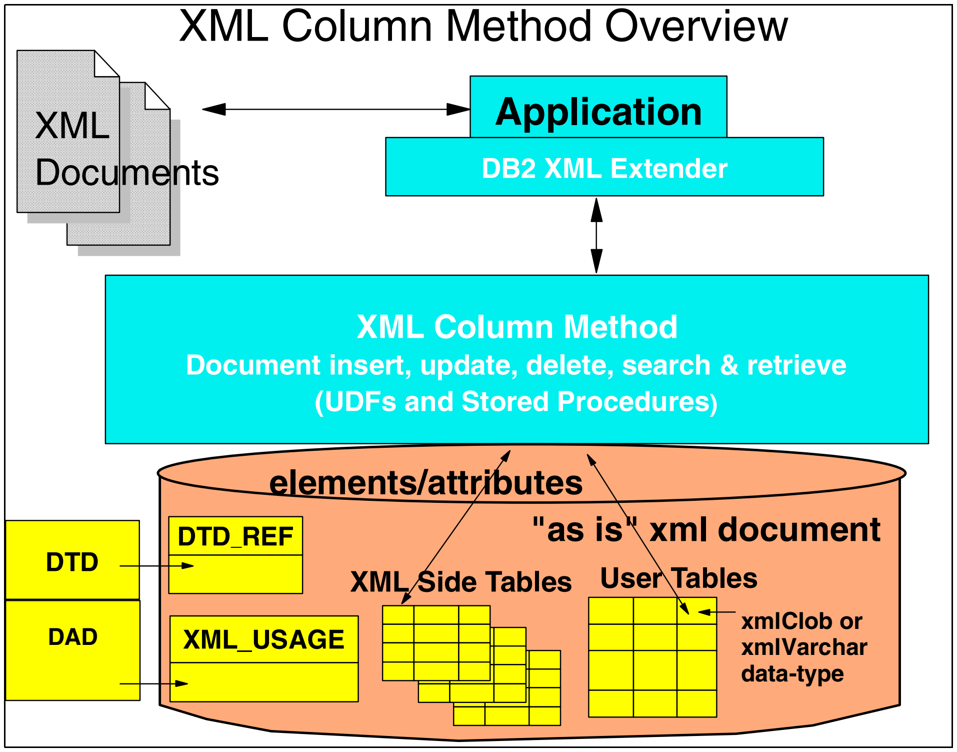


Figure 6-14 XML Column method overview

6.4.1 XML Collection

The *XML Collection* refers to a set of tables that is mapped to XML documents. This method is used to decompose the incoming XML document into database table columns or compose XML document from the data in database tables.

The XML Collection is defined in a DAD file, which specifies how the elements or attributes of an XML document are mapped to one or more tables. The collection is given a name so that it is easily run with stored procedures when composing or decomposing the XML documents.

The tables can be a new set of tables that the XML Extender creates based on the DAD file, when decomposing the XML document, or existing set of tables that were used to compose the XML document. Columns of these tables are mapped to the elements or attributes of the XML data. The data in these table columns does not contain XML tags; it contains the content of the element and the values of the attributes in the XML document. Figure 5-7 gives an overview of XML collection.

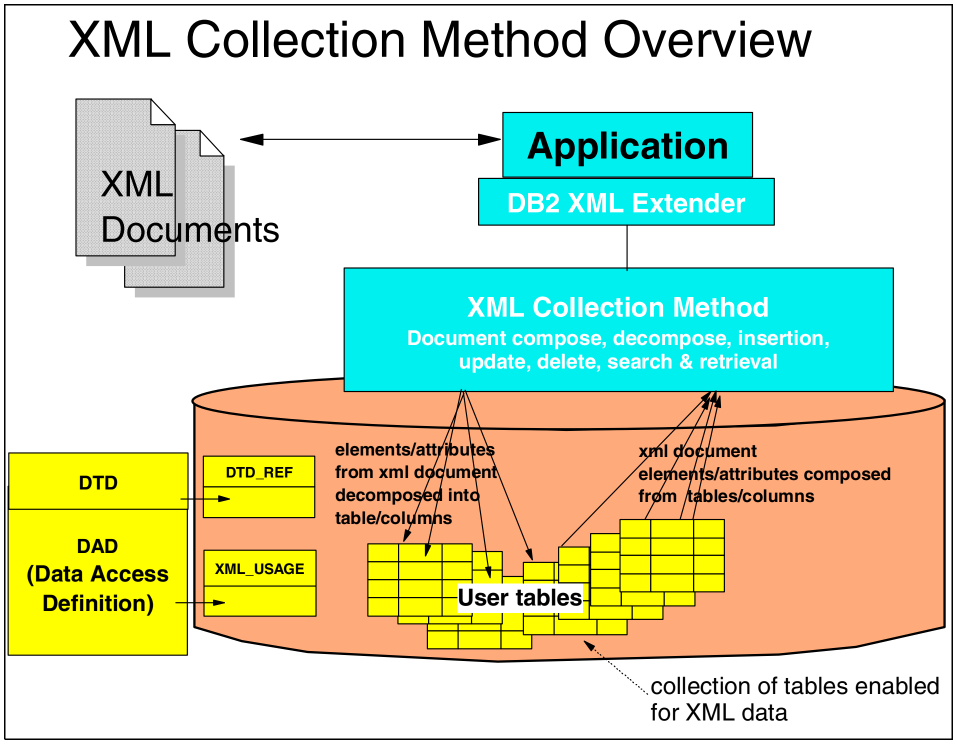

XML Collection Method Overview
XML Documents
Application
DB2 XML Extender
XML Collection Method
Document compose, decompose, insertion, update, delete, search & retrieval
DTD
DAD (Data Access Definition)
DTD_REF
XML_USAGE
elements/attributes from xml document decomposed into table/columns
xml document elements/attributes composed from tables/columns
User tables
collection of tables enabled for XML data

Figure 6-15 XML Collection Method overview

When to use XML Collection method

We recommend using the XML Collection method under the following conditions:

- You have data in your existing DB2 database and you want to compose XML documents based on a specified DTD. This is very helpful in exchanging data between applications within an organization or across organizations so long as the XML data conforms to the 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. Basically, you can use the data in your database to compose different XML documents based on your DTDs.
- You have XML documents that 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 store the XML data in existing tables or in new tables.

Mapping schemes for XML collections

For using the XML collection method, you must select a mapping scheme that defines how XML data is represented in a relational database.

For example, let us consider the DTD shown in Example 6-14.

Example 6-14 Sample DTD

```
<!ELEMENT employee (name, dept, proj*)>
<!ATTLIST employee empno CDATA #REQUIRED>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT dept (#PCDATA)>
<!ELEMENT proj (projno, startdate)>
<!ELEMENT projno (#PCDATA)>
<!ELEMENT startdate (#PCDATA)>
```

For the sample DTD shown in Example 6-14, we can determine how we want to create the tables (for decomposition) to map to the incoming XML document to table columns. Figure 6-16 shows an example of the mapping scheme.

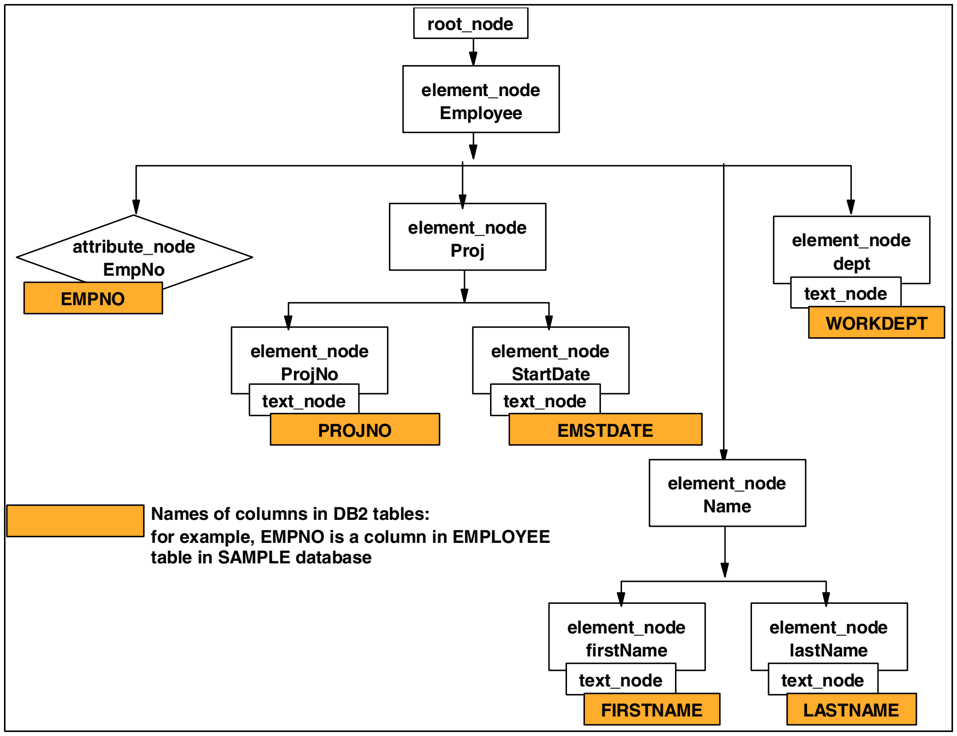


Figure 6-16 XML Collection mapping scheme

By analyzing the mapping scheme we will be able to determine whether we can compose the entire XML document with one SQL statement or not. You can use one of the two types of mapping schemes when defining a collection in the DAD file:

- *SQL mapping*. SQL mapping uses SQL SELECT statement to define the DB2 tables and conditions used for composing a document from the collection.
- *RDB_node mapping*. RDB_node mapping uses an XPath-based relational database node (RDB_node). The RDB_node has child elements that define the tables, columns, and conditions used to associate XML data with DB2 tables.

SQL mapping

Defining a DAD file with SQL mapping scheme allows direct mapping from relational data to XML documents through a single SQL statement and the *XPath data model*. When working with SQL mapping it is important to understand the following points:

- SQL mapping scheme can be used for composition, not for decomposition.

- The tag <SQL_stmt> identifies the SQL_stmt element in the DAD file.
- The content of the element <SQL_stmt> is a valid SQL SELECT statement.
- The SQL_stmt maps the columns in the SELECT statement to XML document elements or attributes.
- The column names in the SELECT define the value of an *attribute_node* or the content of a *text_node.*

  An *attribute_node* maps the value of an attribute in the XML document to a table column.

  A *text_node* maps the content of the element in the XML document to a table column. A text_node is specified for the lowest level element nodes, that is, these element nodes do not have any child elements.
- The FROM clause of the SELECT identifies the tables containing the data.
- The WHERE clause specifies *join* (the columns on which the collection tables will be joined) and search condition.

Table 6-5 EMPLOYEE table in SAMPLE database

Column name	**Data type**
EMPNO	CHAR(6) NOT NULL
FIRSTNME	VARCHAR(12) NOT NULL
MIDINIT	MIDINIT(1) NOT NULL
LASTNAME	VARCHAR(15) NOT NULL
WORKDEPT	CHAR(3)
PHONENO	CHAR(4)
HIREDATE	DATE
JOB	CHAR(8)
EDLEVEL	SMALLINT NOT NULL
SEX	CHAR(1)
BIRTHDATE	DATE
SALARY	DECIMAL(9,2)
BONUS	DECIMAL(9,2)
COMM	DECIMAL(9,2)

Table 6-6 EMP_ACT TABLE in SAMPLE database

Column name	Data type
EMPNO	CHAR(6) NOT NULL
PROJNO	CHAR(6) NOT NULL
EMSTDATE	DATE

Example 6-15 Example DAD file specification with SQL_mapping scheme

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "X:\dxx\dtd\dad.dtd">
<DAD>
   <validation>NO</validation>
   <SQL_stmt>
         SELECT a.empno, firstname, lastname, workdept, b.projno, b.emstdate
         from EMPLOYEE a, EMP_ACT b
         where a.empno = b.empno order by b.projno
   </SQL_stmt>
   <prolog>?xml version="1.0"?</prolog>
   <doctype>!DOCTYPE employee SYSTEM
                "<drive-letter>:\redbooks\SG246586\code\employee.dtd">
   </doctype>
   <root_node>
      <element_node name ="employee">
         <attribute_node name="id">
           <column name="empno"/>
         </attribute_node>
         <element_node name ="firstname">
            <text_node>
               <column name="firstname"/>
            </text_node>
         </element_node>
         <element_node name ="lastname">
            <text_node>
               <column name="lastname"/>
            </text_node>
         </element_node>
         <element_node name ="workdept">
            <text_node>
               <column name="workdept"/>
            </text_node>
         </element_node>
         <element_node name ="project" multi_occurrence="YES">
            <element_node name="projno">
               <text_node>
                  <column name="projno>/>
               </text_node>
            </element_node>
            <element_node name ="startdate">
               <text_node>
                  <column name="emstdate"/>
               </text_node>
            </element_node>
         </element_node>
      </element_node>
   </root_node>
</Xcollection>
</DAD>
```

RDB_node mapping

This method defines the location of the content of an XML element or the value of an attribute so that the XML Extender can determine where to store or retrieve the XML data. It uses the XML Extender-provided RDB_node, which contains one or more definitions for tables, optional columns, and conditions.

When working with RDB_node mapping it is important to understand the following points (Example 5.6 and 5.7 refer to Tabes 5.5 and 5.6):

- RDB_node mapping scheme can be used for composition and decomposition.
- Use <RDB_node> element in each of the top nodes for *element_node* and for each *attribute_node* and *text_node*. An *element_node* corresponds to an element in the XML document. An element_node can have child element nodes. Refer SQL mapping for definition of attribute_node and text_node.
  - **RDB_node for the top element_node:**

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

    - Specify all tables that are associated with the XML document. For the mapping scheme shown in Figure 5-8, the RDB_node for the element employee is shown in Example 6-16.

Example 6-16 The RDB_node

```
<element_node name="employee">
<RDB_node>
  <table name="employee"/>
  <table name="emp_act"/>
  <condition>
    employee.empno = emp_act.empno
  /condition>
</RDB_node>
```

- For decomposing or enabling an XML collection specified in the DAD file, you must specify a primary key for each table.
- Use the orderBy attribute to recompose XML documents containing elements or attributes with multiple occurrence.

– **RDB_node for each attribute and text_node**

  Specify an RDB_node for each attribute_node and text_node, telling the stored procedure from which table/column and under which condition to get data:

  i. Specify the name of the table containing the column data. The table name must be included in the RDB_node of the top element_node. In Figure 5-8, the text_node of element <startdate> is associated with the table emp_act. The RDB_node mapping is shown in Example 6-17.

Example 6-17 RDB_node mapping

```
<element_node name="startdate">
<text_node>
  <RDB_node>
    <table name="emp_act"/>
    <column name="emstdate"/>
    <condition>
      emstdate > 2001-01-01
    /condition>
  </RDB_node>
</text_node>
</element_node>
```

  ii. Specify the name of the table column that contains the data for the element. In Example i, we specified the column EMSTDATE.

  iii. Specify the condition under which the XML data is generated. In Example i, we specified the condition for generating the XML data for projects with project start date greater than ’2001-01-01’.

  iv. For decomposing a document or enabling the XML collection specified in the DAD file, you must specify the column type for each attribute_node and text_node. For example, we can specify that the column type for EMPNO is CHAR as follows:

```
<column name=”empno” type=char”>
```

XML Extender administration tools

The XML Extender administration functions help you to enable your database and table columns for XML, and map XML data to DB2 relational structures. XML Extender provides the following tools to complete administration tasks:

1. *A command line tool*. The administration command is **dxxadm** and all the administration functions can be executed with this command from the DB2 command window. For example, you can enter the following command in the DB2 command window to enable a database for XML:

   ```
   dxxadm enable_db databasename
   ```

2. *An administration wizard*. The wizard is a GUI tool that helps you accomplish the administration such as, enabling a database, table, or column, enabling a collection, or editing the DAD file. All the administration functions that you can perform with the wizard are also available with the command line tool.

   The functions to edit the DAD file are now available in the WebSphere Studio Application Developer tool.

3. *Programming interfaces*. The XML Extender provides a number of UDFs that can be used within the SQL statements, and stored procedures that can be invoked from application programs. A list of these UDFs and stored procedures is provided in “Administrative support tables, UDTs, and UDFs” on page 128.

XML MQSeries enablement

In XML Column method, and XML Collection, we discussed the two methods of storing and accessing XML data: the XML column method for storing/retrieving *as-is* documents in the database tables, and the XML collection method for decomposing incoming XML document into tables and composing XML document from the data in DB2 tables.

With the MQ XML UDFs you can complete the following tasks:

- Query XML documents with the XML column method, and then publish the results to a message queue.
- Retrieve an XML document from a message queue, decompose it into untagged data and store the data in tables.
- Compose an XML document from DB2 data and send the document to a message queue.

MQSeries supports three messaging models:

- *datagrams*: Messages are sent to a single destination with no reply expected.
- *publish/subscribe*: Publishers and subscribers register with a publication service. One or more publishers send a message to the publication service which distributes the message to subscribers who want to receive the message from the publisher.
- *request/reply*: Messages are sent to a single destination and the sender expects a reply.

You can use these three messaging models to distribute XML data and documents between disparate applications with the help of MQSeries and MQXML functions and stored procedures.

7

Generators

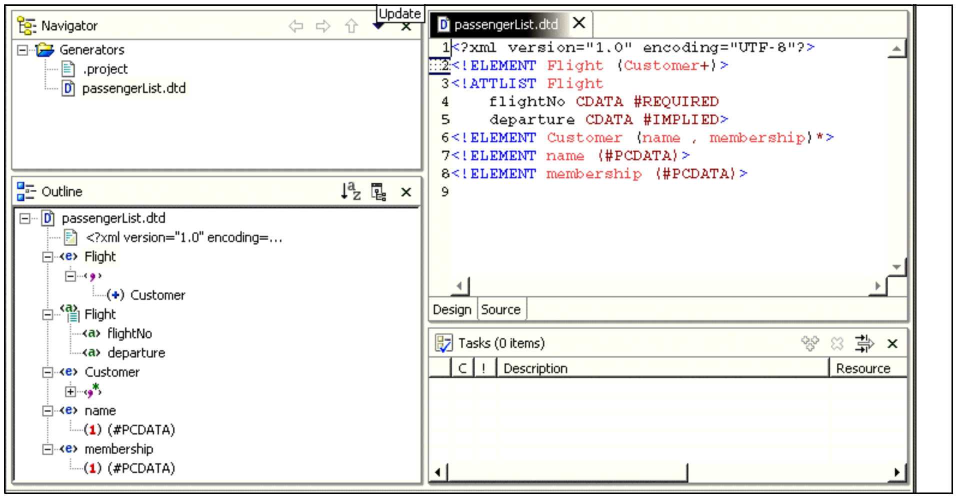
In this chapter we discuss WebSphere Studio XML generators. With these tools you can generate XML Schemas, XML from other components, and also generate components from XML files and XML Schemas.

In this chapter describes the following generators:

- Schemas: DTD <—> XSD
- XML: XML <—> DTD/XSD
- XSD —> HTML doc
- JavaBeans from DTD/XSD
- JavaBeans —>XML/XSL
- HTML —>XML/XSL

7.1 DTD <—> XSD

WebSphere Studio Application Developer offers the developer the ability to transform your XSD file to a DTD file or a DTD file to a XSD file, so whether you start is a matter of in which schema you feel more comfortable. We are going to use the DTD file shown in Figure 7-1.



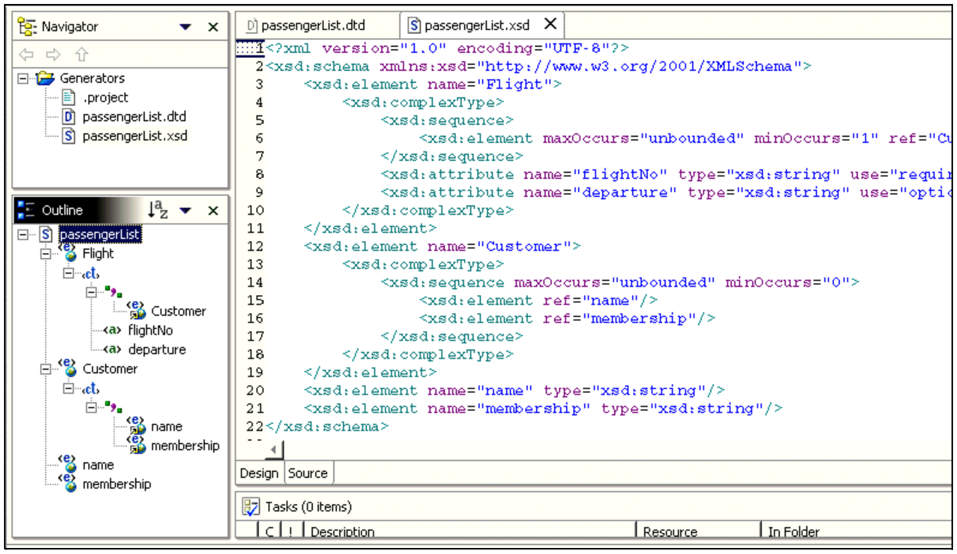
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Flight (Customer+)>
<!ATTLIST Flight
    flightNo CDATA #REQUIRED
    departure CDATA #IMPLIED>
<!ELEMENT Customer (name , membership)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT membership (#PCDATA)>
```

Figure 7-1 passengerList.dtd

Having the passengerList.dtd file, we can generate the corresponding XSD file using the generator following the next steps:

1. Select **passengerList.dtd**
2. Right-click **Generate—>XML Schema**
3. Select the folder to store the file. In this example it is *Generators*.
4. Click **Finish.**

The resulting XSD file is shown in Figure 7-2 on page 147.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Flight">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="Cu
            </xsd:sequence>
            <xsd:attribute name="flightNo" type="xsd:string" use="requi
            <xsd:attribute name="departure" type="xsd:string" use="opti
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Customer">
        <xsd:complexType>
            <xsd:sequence maxOccurs="unbounded" minOccurs="0">
                <xsd:element ref="name"/>
                <xsd:element ref="membership"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="membership" type="xsd:string"/>
</xsd:schema>
```

Figure 7-2 passengerList.xsd

7.2 XML <—> DTD/XSD

There are several ways to create XML files using Application Developer. Now we are going to discuss how to:

- Create an XML file from DTD file
- Create an XML file form XSD file
- Create a DTD file from an XML file
- Create XSD from an XML file

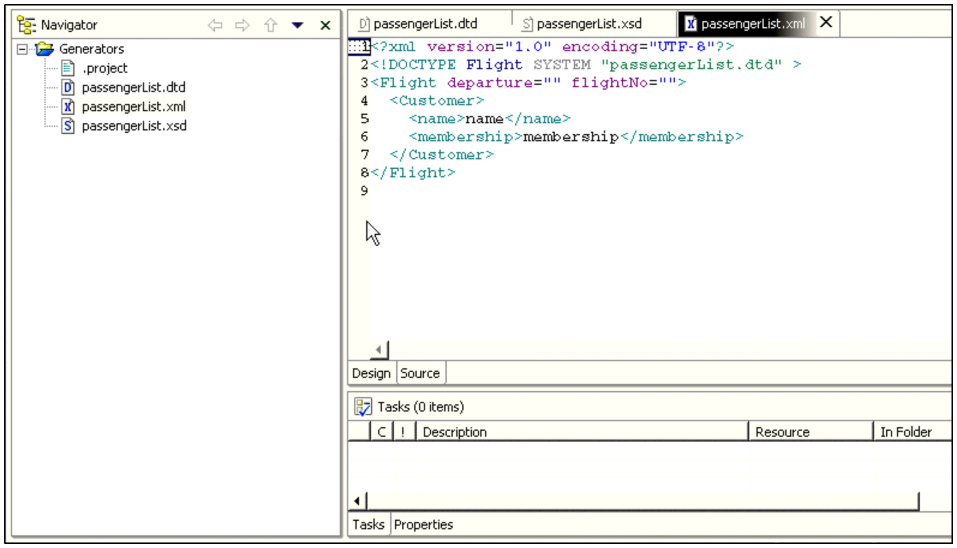
This options are very convenient if you have an existing rules file that details how the new XML file should look. By letting the XML file be created based on these rules, you automatically get a coded skeleton for the file. The option to create XML files from these file types is also available from the context menu of DTD and XML Schema files. XML files are placed in an existing folder under any project type. If you plan to create JavaBeans from your DTD files then this should be a Java project.

7.2.1 Create an XML file from a DTD file

There are two ways to create an XML file from a DTD file. The first one is using the Create XML File wizard:

1. Open XML perspective if necessary.

2. Click **File—>New—XML**.
3. Select **Create XML file from a DTD file**.
4. Enter the folder to store the file. In this example is Generators.
5. Enter the XML file name. i.e. **`passengerList.xml`.** Click **Next.**
6. Click **Select file from Workspace** option.
7. From Workbench Files box, select **passengerList.dtd.** Click **Next**.
8. Select **flight** as root name.
9. Select **Create required and optional content.** Click **Finish.**



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Flight SYSTEM "passengerList.dtd" >
<Flight departure="" flightNo="">
  <Customer>
    <name>name</name>
    <membership>membership</membership>
  </Customer>
</Flight>
```

Figure 7-3 passengerList.xml

The second way to do it is using the context menu of the passengerList.dtd file:

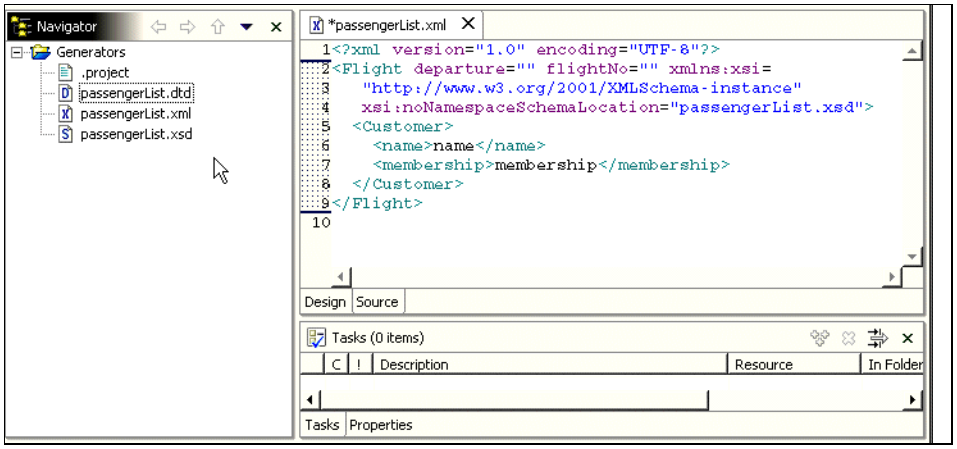
1. Select **passengerList.dtd.**
2. Right-click **Generate—>XML File**
3. Enter the folder to store the file.
4. Enter **`passengerList.xml`** as file name. Click **Next**.
5. Select **Flight** as root element.
6. Select **Create required and optional content**. Click **Finish.**

7.2.2 Create an XML file from an XSD file

There are two ways to create an XML file from a XSD file. The first one is using the Create XML File wizard: (See Figure 7-4.)

1. Open XML perspective if necessary.
2. Click **File—>New—XML**.
3. Select **Create XML file from a XSD file**.
4. Enter the folder to store the file. In this example it is Generators.

5. Enter the XML file name. i.e. **`passengerList.xml.`** Click **Next**.
6. Select **Select file from Workspace** option.
7. From Workbench Files box, select **`passengerList.xsd.`** Click **Next**.
8. Select **Flight** as the root name.
9. Select **Create required and optional content.** Click **Finish**.



```
<?xml version="1.0" encoding="UTF-8"?>
<Flight departure="" flightNo="" xmlns:xsi=
  "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="passengerList.xsd">
  <Customer>
    <name>name</name>
    <membership>membership</membership>
  </Customer>
</Flight>
```

Figure 7-4 passengerList.xml

The second way to do it is using the context menu of the passengerList.xsd file:

1. Select **passengerList.xsd**.
2. Right-click **Generate—>XML File**
3. Enter the folder to store the file.
4. Enter **`passengerList.xml`** as file name. Click **Next**.
5. Select **Flight** as root element.
6. Select **Create required and optional content**. Click **Finish.**

7.2.3 Create DTD/XSD files from XML

Now, let us work in the case that we have an XML file, and we need to create an schema for it. Application Developer provides two wizards to generate the schema from the XML file. To show you how to do this, we are going to use the file Customer.xml which contains the personal data of a customer.

Example 7-1 Customer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Customer>
    <Firstname>John</Firstname>
    <Lastname>Doe</Lastname>
    <Email>jdoe@dummy.com</Email>
    <Membership>123456</Membership>
  </Customer>
</Root>
```

To generate the DTD file follow the steps below:

1. Select **customer.xml**
2. Right-click **Generate—>DTD**
3. Enter the folder name to store the file.
4. Enter the file name. Click **Finish**.

See resulting DTD file below.

Example 7-2 Customer DTD

```
<?xml version='1.0' encoding="UTF-8"?>
<!ELEMENT Customer
   (Firstname,Lastname,Email,Membership)
>
<!ELEMENT Email
   (#PCDATA)
>
<!ELEMENT Firstname
   (#PCDATA)
>
<!ELEMENT Lastname
   (#PCDATA)
>
<!ELEMENT Membership
   (#PCDATA)
>
<!ELEMENT Root
   (Customer+)
>
```

To generate the XSD file follow the steps below:

1. Select **Customer.xml.**
2. Right-click **Generate—>XML Schema.**
3. Enter the folder to store the file.
4. Enter the file name. Click **Finish**.

See resulting XSD file in Example 7-3.

Example 7-3 Customer.xsd

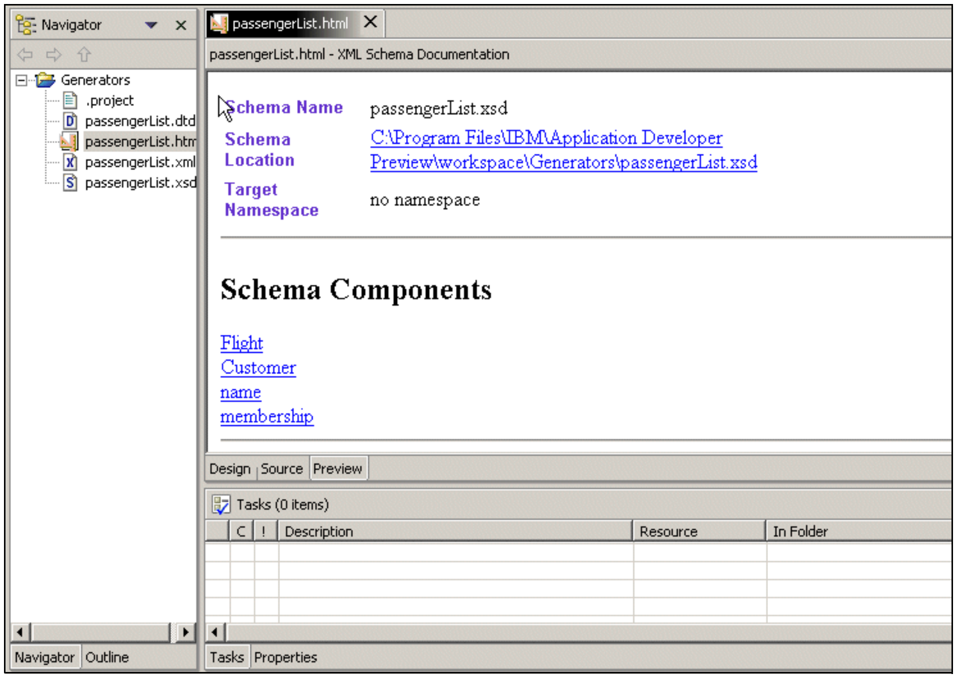
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Customer">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Firstname"/>
                <xsd:element ref="Lastname"/>
                <xsd:element ref="Email"/>
                <xsd:element ref="Membership"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Email" type="xsd:string"/>
    <xsd:element name="Firstname" type="xsd:string"/>
    <xsd:element name="Lastname" type="xsd:string"/>
    <xsd:element name="Membership" type="xsd:string"/>
    <xsd:element name="Root">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1"
                   ref="Customer"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

7.3 Generate a HTML from an XSD

We can generate a HTML document to describe how the XSD file is designed. This document is similar to make a Java doc when creating documentation of classes. To generate the documentation of an XSD file follow the next steps:

1. Select **passengerList.xsd** file.
2. Right-click **Generate—>HTML Doc.**
3. Enter the folder to store the documentation.
4. Enter the HTML file name. Click **Finish**.

Figure 7-5 is showing the result of this process.



Schema Name	passengerList.xsd
Schema Location	C:\Program Files\IBM\Application Developer Preview\workspace\Generators\passengerList.xsd
Target Namespace	no namespace

Schema Components

Flight
Customer
name
membership

Figure 7-5 HTML Documentation for passengerList.xsd

7.4 JavaBeans from DTD/XSD

WebSphere Studio Application Developer gives us several tools to quickly build applications. One of the tools is the generation of JavaBeans from an XML Schema or a DTD file, which allows you to code directly to instance rather than DOM APIs. This wizard:

- Creates a bean for each element in DTD, XSD.
- Creates a Factory bean for creation of a new XML document.
- Creates a sample program for using the beans created.

To create the JavaBeans you need to have a container within Application Developer. This must be a project which could contain Java classes such as Web Project. To create a Web Project follow the next steps:

1. Click **File—>New—>Project**.
2. Select **Web Project** from Web category. Click **Next**.
3. Enter a project name, for example `Travel`.
4. Select **J2EE Web Application Project**. Click **Next**.
5. Create or select an Enterprise Application Project. Click **Finish**.

To generate the JavaBeans from the schema follow the next steps:

1. Select **passengerList.xsd.**
2. Right-click **Generate—>JavaBeans.**
3. Enter the container created before.
4. Enter a name for the package.
5. Select **Flight** as the root element.
6. Select **Generate sample test program**. Click **Finish**.

Figure 7-6 is showing the generated Java codes.

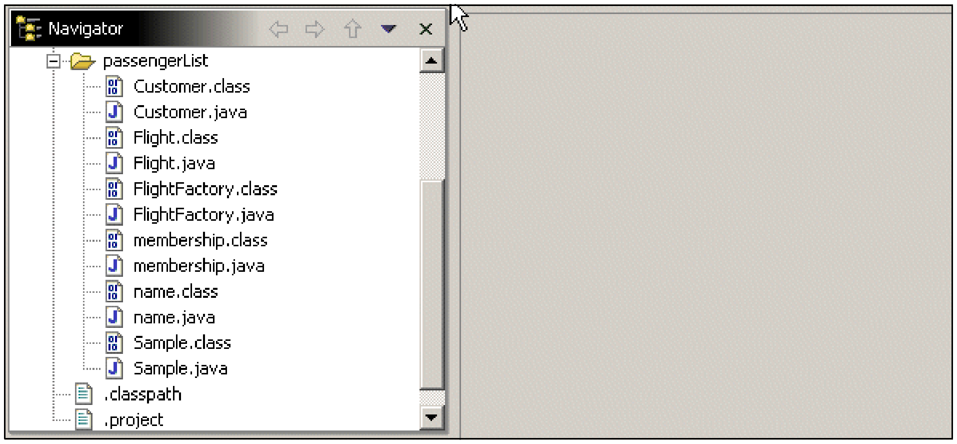


Figure 7-6 JavaBeans from schema

Take a few minutes to review the generated code, you will notice that some classes extend *ComplexType* corresponding to main elements, and the other classes extend SimpleType corresponding to sub elements. Try to run the sample application to follow how it builds the XML file.

7.5 Generate XML/XSL from JavaBeans

You can generate Web pages that can access JavaBeans using XSL. For this section we are going to use the JavaBean showed in Example 7-4. We create this Java file in our Travel Web Project and the package is *airline*.

Example 7-4 passenger.java

```
package airline;

/**
 * @author osamurs2
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 */
public class passenger {
   private String name;
   private String membership;

   public passenger()
   {
   }

   public String getName()
   {
      return name;
   }

   public void setName(String aName)
   {
      name=aName;
   }

   public String getMembership()
   {
      return membership;
   }

   public void setMembership(String aMem)
   {
       membership=aMem;
   }


}
```

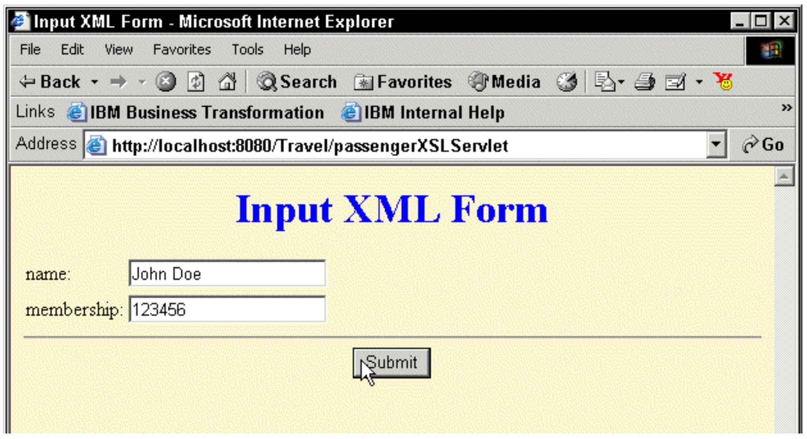
Several files are created in this generator:

- Input XML Form: An XSL Stylesheet to render the JavaBean.
- Result XSL: ResultXSL Stylesheet that displays results from the underlying JavaBean.
- Splash screen: A splash screen to invoke the XSLServlet.
- XSLServlet: This servlet applies the generated XSL to the DOM.
- XML DOM: Produces a DOM from the JavaBean.
- XML Schema: Produces an XML Schema from the JavaBean.

To generate all the files follow the steps below:

1. Open XML perspective if necessary.
2. Click **File—>New—>Java Bean XML Client.**
3. Enter your Web project created before as the destination folder.
4. Enter the package name for the Java files. Click **Next**.
5. Browse for the airline.passenger bean.
6. Select **name** and **membership** as the beans you would like to invoke. Click **Next**.
7. Design Input form as desired. Click **Next**.
8. Design Result page as desired. Click **Next**.
9. Enter a prefix to the file names. Click **Finish**.

The generated servlet implements *doGet* method. This method returns the input form that we designed before. To test the application, you need to call PassengerXSLServlet directly from the URL (Figure 7-7).


Input XML Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Business Transformation IBM Internal Help
Address http://localhost:8080/Travel/passengerXSLServlet
Go

Input XML Form

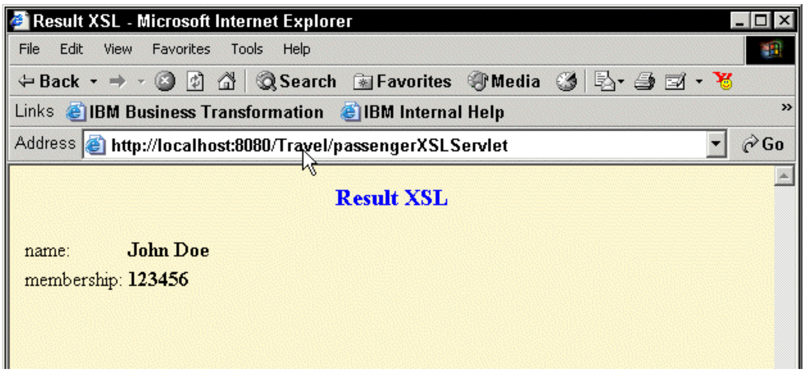
name: John Doe

membership: 123456

Submit

Figure 7-7 Input XML Form screen

Click **Submit** to call the servlet's doPost method (Figure 7-8).


Result XSL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Business Transformation IBM Internal Help
Address http://localhost:8080/Travel/passengerXSLServlet
Go

Result XSL

name: John Doe

membership: 123456

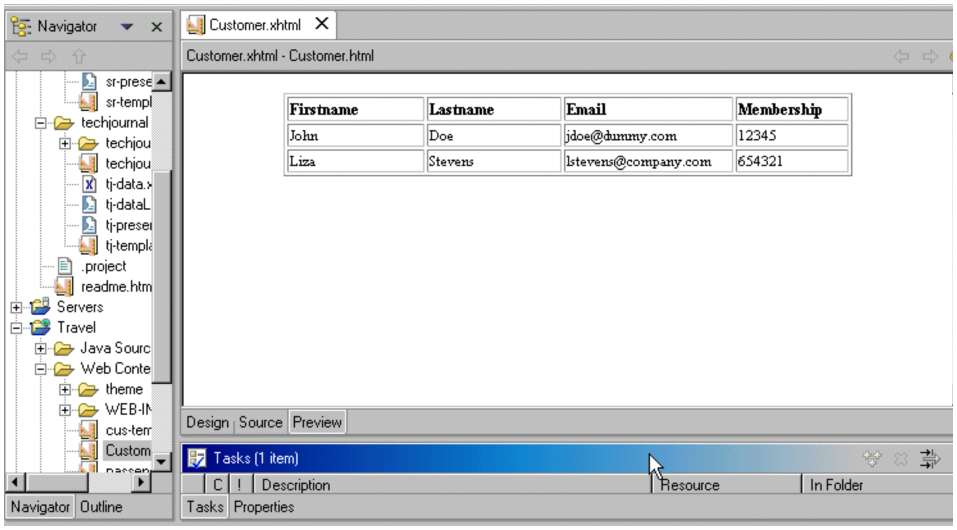
Figure 7-8 Result XSL panel

7.6 Generate XML/XSL from HTML

With this generator we are trying to create XSL and XML files from a specific HTML, also will enable to separate out the presentation logic from the dynamic data in an existing HTML document. It extracts the data into an XML file and the presentation data into two XSL files. Once the separation is completed, you can use XSLT technology to combine new data that is defined in XML format with the generated XSL files to create new HTML Web pages.

So the first step will be to create a HTML file with data. We are going to create an HTML for a customer information design as shown in Figure 7-9:

1. Click **File—>New—>Other.**
2. Select **HTML** from Web category.
3. Enter folder name. Enter `Customer.html` as file name.
4. Click **Finish**.



Firstname	Lastname	Email	Membership
John	Doe	jdoe@dummy.com	12345
Liza	Stevens	lstevens@company.com	654321

Figure 7-9 Customer.html

7.6.1 Preparing the HTML file for generation

There are several steps to get the HTML file ready to generate XSL and XML file. The first step is to rename *customer.html* to *customero.xhtml,* since the generator is only available to files with this extension.

Open the renamed file and follow the next steps:

1. Delete any DOCTYPE declaration.
2. Remove any entity references. For example, substitute `&nbso`l with a blank.
3. Fix up any missing tags. For example, change `<br>` to `</br>`*.*

Adding annotation tags

The second step is to create a template file that contains the annotation tags, to mark the section that contains data that needs to be extracted into the XML document. To do this follow the next steps:

1. Copy customer.xhtml into a new file, for example, `cus-template.xhtml`.
2. Open cus_template.xhtml file and add the tag `<?HTMLTemplate version=”0.1”?>` at the top to indicate that it is a template file.
3. Add the tag `<TemplateRegion name=”nodeName”`**>** around the data that you want to convert.The nodeName name will be the node name in the resulting XML file.

4. Delete the repeating data in each TemplateRegion. In our example, we have to delete the repeating records of customers to keep only one. (See Example 7-5.)
5. Parameterized each data item using this format: {tagName}. The tagName will be used in the resulting XML and XSL files.

Example 7-5 cus-template.xhtml

```
<?HTMLTemplate version="0.1"?>
<TemplateRegion name="Root">
<HTML>
<HEAD>
<META name="Generator" content="IBM WebSphere Studio"/>
<META http-equiv="Content-Style-Type" content="text/css"/>
<LINK href="/Travel/theme/Master.css" rel="stylesheet" type="text/css"/>
<TITLE>Customer.html</TITLE>
</HEAD>
<BODY>
<DIV align="center">
<TABLE border="1" width="524">
   <TBODY>
      <TR>
         <TD width="133">
            <FONT color="#000000" size="3" face="Times New Roman">
            <B>Firstname</B></FONT></TD>
         <TD width="123">
            <FONT color="#000000" size="3" face="Times New Roman">
            <B>Lastname</B></FONT></TD>
         <TD width="123">
            <FONT color="#000000" size="3" face="Times New Roman">
            <B>Email</B></FONT></TD>
         <TD width="133">
            <FONT color="#000000" size="3" face="Times New Roman">
            <B>Membership</B></FONT></TD>
      </TR>
      <TemplateRegion name="Customer">
      <TR>
         <TD width="133">
            <FONT color="#000000" size="2" face="Times New Roman">
               {Firstname}
            </FONT>
         </TD>
         <TD width="123">
            <FONT color="#000000" size="2" face="Times New Roman">
               {Lastname}
            </FONT>
         </TD>
         <TD width="123">
```

```
            <FONT color="#000000" size="2" face="Times New Roman">
               {Email}
            </FONT>
         </TD>
         <TD width="133">
            <FONT color="#000000" size="2" face="Times New Roman">
               {Membership}
            </FONT>
         </TD>
      </TR>
      </TemplateRegion>
   </TBODY>
</TABLE>
</DIV>
</BODY>
</HTML>
</TemplateRegion>
```

The final step is to invoke the Generator. This wizard is going to generate the XSL stylesheets, and optionally the XML data that can be extracted from your HTML document:

1. Select **cus-template.xhtml.**
2. Right-click **Generate—>XSL File.**
3. Review the information and change as needed. Click **Next.**
4. Check Extract XML Data from HTML file.
5. Select **Customer.xhtml.** Click **Next**.
6. Enter the resulting XML filename, `customer.xml`, and select the folder to store it. Click **Finish**.

Example 7-6 shows the generated xml.

Example 7-6 Customer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Customer>
    <Firstname>Firstname</Firstname>
    <Lastname>Lastname</Lastname>
    <Email>Email</Email>
    <Membership>Membership</Membership>
  </Customer>
  <Customer>
    <Firstname>John</Firstname>
    <Lastname>Doe</Lastname>
    <Email>jdoe@dummy.com</Email>
    <Membership>123456</Membership>
  </Customer>
  <Customer>
    <Firstname>Liza</Firstname>
    <Lastname>Stevens</Lastname>
    <Email>lstevens@company.com</Email>
    <Membership>654321</Membership>
  </Customer>
</Root>
```

Part 3

XML application development

Part three provides an overview of Web Services and Enterprise JavaBeans capabilities of Application Developer. Then introduces how to create three type of applications step-by-step.

8

WebSphere and XML approaches

Chapter eight provides an overview of Web services and Enterprise JavaBeans capabilities of Application Developer. The overviews are a precursor to two applications that have been detailed in Chapter 9, “Developing XML Web services” on page 177 and Chapter 10, “Development of XML-based Enterprise applications” on page 215.

The first, the Passenger List application demonstrates Application Developer’s capabilities in creating and generating the basics in Data Access and XML files. For the user interface, it uses HTML, XSLT stylesheets, servlets and XSL transformers.The Customer Registration application is oriented towards the usage of Enterprise JavaBeans.

8.1 XML in Application development

Web services are becoming part of today's Internet. They play a key role for maximum interoperability across the different technologies on the Internet. XML's role is that of a data interchange format between the different components, not only between applications, but also within the applications. In Client/Server and distributed computing technologies, integration between proprietary systems was always expensive and time-consuming. To derive efficiencies, systems had to be tightly coupled and were not necessary kept to open-standards as we have now. As compared to the binary formats used in those systems, XML is very verbose. However, with the latest improvements in network technologies, especially in network bandwidth, compression techniques and improvements in processing, this disadvantage has been absolved.

In Web services, the data types are specified using XML Schema, which are an improvements on DTDs, and the data that is exchanged is in XML format. In other technologies, the proprietary systems had to make use of data structures that were closely dependent on the underlying systems. An example, would be an applications with its client running in Windows and the data being on a mainframe systems. Data from the mainframe, being in EMBDIC format had to be translated to formats suitable for the PC environment. If only, all data were in a common format, that was shared by every system, data interchange issues would have taken less of a priority.

As the usage of XML increases, tools and utilities will be required for authoring XML, Schemas and the Web Services Description Language (WSDL). with these new tools, application designers and developers will use XSD to gain flexibility in their product and application implementations. Web services can now be implemented using languages such as Java and Javascript. With XSLT, XML Query language and the support for XML by major database vendors like IBM's DB2 UDB, XML documents are can be processed as they are, instead of having to be converted between XML and other languages or formats. DB2 UDB includes the XML Extender, which allows XML documents to be stored in columns and also to be store the elements into tables as fields. The XML Document Access Definition (DAD) files maps XML to relational data. Application Studio has incorporated the RDB to XML mapper tool to facilitates the generation of the DAD. From the DAD, the DB2 XML Extender (DADX) can be produced. This file is in XML and contains operations defined by normal SQL statements and calls to the stored procedures in the DB2 XML Extender. IBM SOAP includes provides the handling of DADX files. This file, since it is ASCII text, can be easily edited using any text editor or the XML editor tools.

Any distributed computing technology, they must be support for converting between from one system to another. In the case of SOAP, the data interchange format is XML and the run-time components are implemented in Java. Therefore, there a mapping between Java and XML data types must be provided for. The Apache SOAP run-time environment allows the developer to map between Java and XML data types for an encoding style. The designer can specify a serializer to marshal the Java type to XML, a deserializer to unmarshall the XML type to Java, or both for two-way mapping.

The rules for mapping between Java and XML data types are stored in a SOAP mapping registry object that is used by either the Java client proxy or the Web service. The SOAP mapping registry has predefined rules for mapping between simple Java and XML types. For complex XML types, it is up to the developer to specify the mapping. The best way to map is to use the org.w3c.dom.Element. This represents a generic XML element in the Document Object Model (DOM).

8.2 Web services

A Web service is a collection of functions that are packaged as a single entity and published to the network for use by other programs. Web services are building blocks for creating open distributed systems, and allow companies and individuals to quickly and cheaply make their digital assets available worldwide. Some examples of Web services are:

- A credit checking service that returns credit information when given a person's identification number.
- A stock quote service that returns the sock price associated with a specified ticker symbol.
- A purchasing service that allows computer systems to buy office supplies when given an item code and a quantity.

A Web service can aggregate other Web services to provide a higher-level set of features. For example, a Web service could provide a set of high-level features by orchestrating lower-level Web services for car rental, air travel, and hotels. Applications of the future will be built from Web services that are dynamically selected at runtime based on their cost, quality, and availability.

Web services are pretty much guaranteed to be at the heart of the next generation of distributed systems. The reasons are:

- *Interoperability:* Any Web service can interact with any other Web Service.The Simple Object Access Protocol (SOAP), the new standard protocol by all of the major vendors (and most of the minor ones), the agonies of converting between CORBA, DCOM and other protocols should be over. And because Web services can be written un any language, developers do not need to change their developed environments in order to produce or consume Web services.

- *Ubiquity*: Web Servies communicate using HTTP and XML. Therefore, any device, which supports these technologies can both host and access Web services. Pretty soon, they will be present in phones, cars, and even soda machines. Soda supplies getting low? No problem, the wireless-networked soda machine can contact their local supplier's Web service and order more of your favorite beverage.
- *Low barrier to Entry*: The concept behind Web services are easy to understand and free toolkits from vendors like IBM and Microsoft allow developers to quickly create and deploy Web services. In addition, some of the toolkits allow pre-existing COM components and JavaBeans to be easily exposed as Web services.
- *Industry Support*: All of the major vendors are supporting SOAP and the surrounding Web services technology. For example, the Microsoft.NET platform is based on Web services, thereby making it very easy for components written in Visual Basic to be deployed as Web services, and consumed by Web services written using WSAD and, and vice-versa.

Web Services Description Language (WSDL) is a new specification to describe networked XML-based services. It provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying protocol, such as SOAP and XML, or encoding, such as Multipurpose Internet Messaging Extensions (MIME). WSDL is a key part of the effort of the Universal Description, Discovery and Integration (UDDI) initiative to provide directories and descriptions of such on-line services for electronic business.

Detail information is available:

The Web services (r)evolution: Applying Web services to applications by Graham Glass at:

`http://www-106.ibm.com/developerworks/webservices/library/ws-peer1.html`

Using WSDL in SOAP applications: An introduction to WSDL for SOAP programmers by Uche Oqbuji at:

`http://www-106.ibm.com/developerworks/webservices/library/ws-soap/`

8.3 Passenger List application

The Passenger List application was developed to demonstrate application developer's capabilities in a development environment. It consists of only a single operation, that is, for a given flight number, the application retrieves a list of passengers for that flight. All details of the passengers and schedules are stored in the database. As in any such system, we hold flight details such flight numbers, aircraft types and arrival and departure times and also passenger details names, membership number and the flight to be taken.

The database has been kept simple. We could have used other more complex mechanism to query the database, but chose to use the wizards within Application Studio.

8.3.1 Solution Outline

As in all applications, we are a clear distinction between the Graphical User Interface, the Web Tier and the Data Source(Figure 8-1). The advantages of these delineation are obvious. If the tier were loosely coupled, they could be reused to interface with other components of different applications. The Client interface could easily have been implemented using JSP, ASP or even Javascript. Although we have generated HTML, we could easily have generated a display for a PDA or a mobile phone, just but changing the XSL stylesheet.

Similarly, the data server could be any database that was available where proprietary or open source. The invocations for data access, SQL queries, stored procedures could be in the Web Tier with the queries and procedures in the data server. In our Web Tier, also we have used XSLT to transform the XML data.

The Web Tier is the major component, and with time, this too could be further delineated within itself. As transformers, validators and parsers incorporated newer versions of XML and Schemas there would be further modularization in this area.

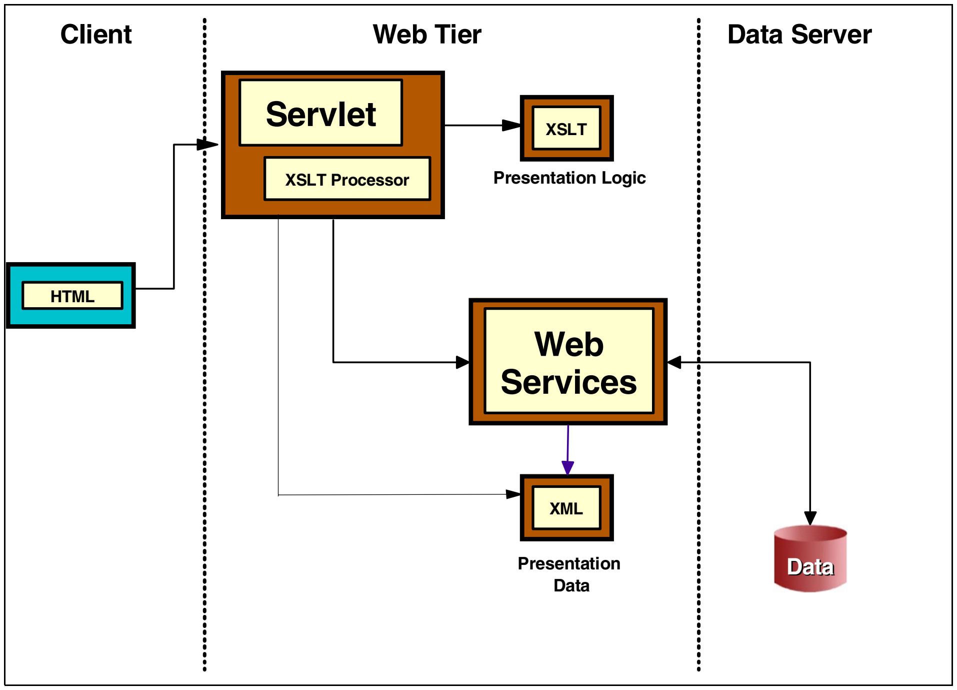


Figure 8-1 Solution overview of the Passenger List application

8.3.2 XML in this application

XML plays a key advantage in this application. It provides the full infrastructure for the data exchange and description between the database and the user interface, and also within Application Studio as well. WSDL bindings are in XML format.

Application Developer includes an integrated suite of tools for XML:

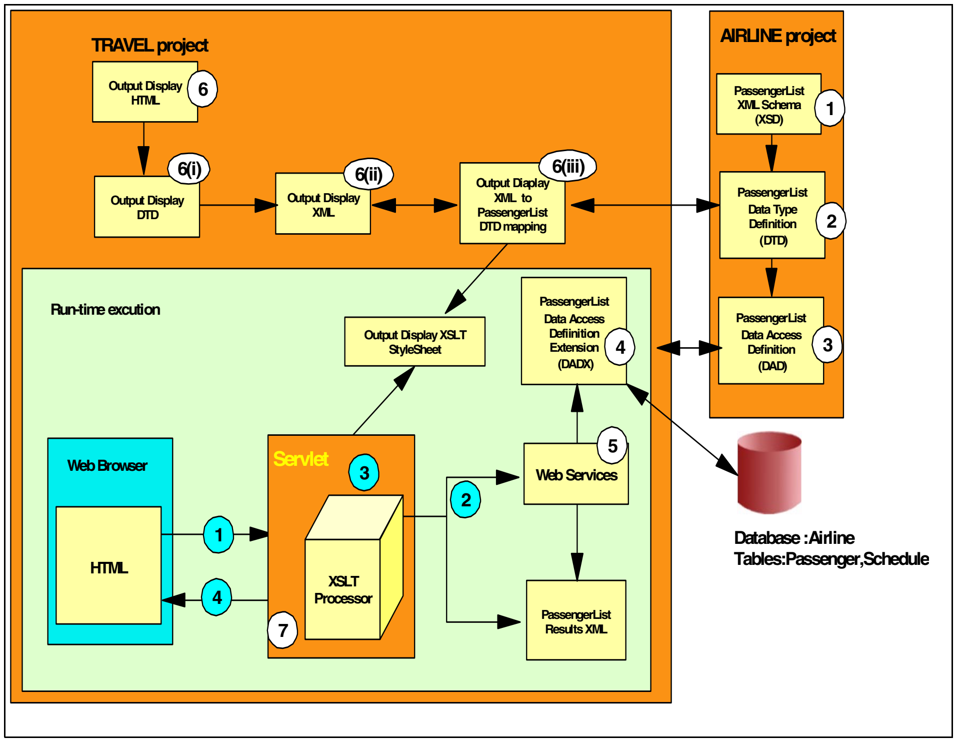
- Importing of Relational Database information through the wizards and specifying it in XML Metadata Interchange (XMI) format. This is done when the database connection is establish.
- XML editing and validation
- XML Schema and DTD editing, validation and generation between the two
- XSLT generation and transformation
- Creation of Input and Output forms from JavaBeans, as well as DTD and XML Schema
- Generation of XML from DTD and XML Schema and vice versa.

- Tools to generate XML data from Relational Data and vice versa.
- Generation of the DAD and DADX for use in the DB2 UDB XML Extender.
- Generation of WSDL bindings.

These tools are used in the development of this application and will be obvious as the construction progresses.

8.3.3 Technical implementation overview

The technical implementation of the Passenger List application consists of two projects. The first is the AIRLINE project is where the data access XML files are generated. The second TRAVEL project builds upon the first to create the graphical user display and the Web services (Figure 8-2). The approach is bottom-up approach where the data access is created first. The graphical user interface is created after, where the input fields are mapped to the tables and attributes in the database.



TRAVEL project
Output Display HTML
6
Output Display DTD
6(i)
Output Display XML
6(ii)
Output Diaplay XML to PassengerList DTD mapping
6(iii)
AIRLINE project
PassengerList XML Schema (XSD)
1
PassengerList Data Type Definition (DTD)
2
PassengerList Data Access Definition (DAD)
3
Run-time excution
Output Display XSLT StyleSheet
PassengerList Data Access Defiinition Extension (DADX)
4
Web Browser
HTML
1
4
Servlet
3
XSLT Processor
7
2
Web Services
5
PassengerList Results XML
Database :Airline
Tables:Passenger,Schedule

Figure 8-2 Development and running of the Passenger List application

The development of the two projects follows these steps:

1. The AIRLINE project starts with the creation of a XML Schema file. For our application, the schema is simple: It holds only flight and customer information. Correspondingly, we also create a database to hold this data. The schema consists of two tables: ’PASSENGER’ and ‘SCHEDULE’. Each table consists of four fields. The creation of these tables and the connection is carried out through the data perspective of Application Developer.

   Application Developer is so rich in functionality that to put all its icons on a single menu would be impossible. The perspectives help to organize all the functions in an organized way. To access different parts of the project, the user will have to switch between the perpectives.

2. From the XML Schema, the DTD is generated. This then serves as a mapping to the relational database tables. The join between the tables are implemented in the RDB to XML Mapping wizard. Note that the join between the two tables is carried out in Application Developer, and not in the database. This serves as a discussion point.

Discussion:

The advantages and disadvantages of establishing relational database relationships in Application Developer:

Advantages of using the database:

- For application development, relationships between tables are best established in the database. The tables are shared by many applications, and therefore is their common point. When a new application is being developed, the relationship have already being established for the application.
- The database is a tried and tested product where joins, triggers, indexes, etc. are commonplace. These are the forte of any database. Using a database offers advantages in terms of scalability and performance. This is important especially for large databases.
- XML documents need to be parsed every time they are accessed and the parsed file must be memory resident during query processing.

Advantages of using the DTD approach in Application Developer

- This could be the best approach if the relationships between the tables were specific to this application.
- In e-Business information exchange, DTDs are a common point between different companies intending to share data. Database schemas and database relationships are proprietary to an organization and are used for commercial advantage.This DTD can now be shared between organizations.
- The shared DTD can be agreed to by the organizations, and then using the XML tools, map parts or all of DTD to the tables in the databases of each organization.

3. The Data Access Definition (DAD) file is then generated. While generating this file, a test harness can also be generated. We will use this to unit test the application until this stage.
4. A new project, the TRAVEL project is created to hold the Web service's functionality. The project is a J2EE project. As a first step, a Web services configuration group is created. After we have copied the DTD and the DAD into the group, we generate a Data Access Definition eXtention (DADX) file. The resulting file will have to be customized with the parameters for the flight.
5. The DADX file then serves as a base for the construction of our Web services through the Web services wizard. This Web service is based on a Java proxy. The wizard also has the option to generate a Web Services Object Runtime Framework (WORF) run-time component to dynamically generate a test and

documentation page. This is made use of at this point to test if the retrieval of data works properly.

6. A graphical user interface is developed using XSL stylesheets. To start with, the target HTML file needs to be located. This file would represent the output page to be displayed. The Application Developer wizards are used to create a mapping between the output from the database to the output required for display through a few steps:
   a. Generating of the DTD
   b. Generation of the XML file for the output data
   c. The mapping between the input data and the output display data. The stylesheet, however, has to be customized.

7. A servlet is developed to transform the data from a hierarchical structure to HTML using XSLT processors. The servlet is in turn called from a HTML page.

During the execution of the system, only the components in the run-time components are executed. The following sequence of events follows:

1. The user activates the servlet through the HTML form.
2. The servlet calls the generated Web service classes. It calls the proxy classes, supplying it the primary key and stores them in a DOM structure.
3. The servlet then passes the DOM structure to the XSL Transformer. The XSLT stylesheet of the display output is also an input to the transfer. The transformer produces an HTML file.
4. The newly created HTML file is sent to the Web browser.

Application Developer XML Web Services provide a powerful new tool for integrating heterogeneous applications over the Internet. Application Developer provides a fully supported production-ready deployment environment for Web services based on the Apache SOAP run-time environment. XML plays a central part by providing a data interchange format that is independent of programming languages, operating systems, and hardware.

8.4 Enterprise JavaBeans

The Enterprise JavaBeans (EJB) specification provides a framework for creating reusable business logic components without regard to system infrastructure or location.

Many business problems today require large, complex systems to solve them. The trend is to use a multi-tier architecture in which the client communicates with an application server (Figure 8-3). The server uses specific business logic and communicates with some form of persistent data storage, typically, a database.

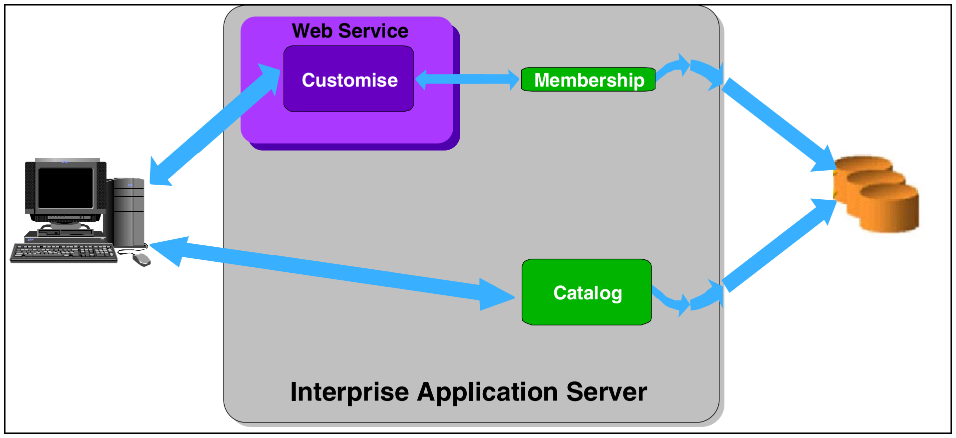


Figure 8-3 Enterprise Application Server

Depending on what type of client is used, an additional, middle tier may exist between the client and the application server (for example, a Web server to provide dynamic HTML).

Developing the *middle* tier has always been the costly part of system development. In a perfect world, the developer would only be concerned with the business logic required to implement the system. But without technologies like Enterprise JavaBeans, developers also need to be expert in several different areas:

- **Business logic:** Developers must understand the business problem and the logic required to solve it.
- **Transactions:** Developers must correctly group different areas of business logic into specific transaction contexts.
- **Database access:** Developers must understand how to gain access to a database for retrieving and updating information.
- **State management:** Developers must understand how to use multiprocessing capabilities so different sections of the business logic can be run concurrently to optimize performance.

And while developing the code for the server application, the developer must also take into account:

- The client and communication protocol to be used to access the server.
- The application server that will be used and the APIs it supports.

- The database that will be used and how to interact with it.

EJBs have been designed to provide a framework in which reusable business logic components can be created without an understanding of the infrastructure or a concern for where the components will be deployed. And with EJBs, it's simple to reuse code in other applications once it's been developed to implement part of a process. (Most business processes can be broken down into a number of distinct parts, and some of these parts will be common to multiple applications.)

The simple answer is that EJBs are a server-side component architecture. They enable a developer to create a component that represents a discrete, well-defined piece of functionality. Since the interfaces to the component can be published to other developers, the component can easily be combined with other components to create a complete application.

In both applications wizards have been extensively used and custom code has been kept to a minimum. The wizards, by generating the required files, pages and code, help to jump start the development of the applications while reducing the learning curve for the developer. They complexity of creating some of the files is hidden behind from the developer, allowing him more time on customizing the application.

8.5 The Customer Registration application

The Customer Registration application was developed to demonstrate Application Developer's capabilities in the JavaBeans and Enterprise JavaBeans area. As in any such similar system, there are always two components: The first called Customer Registration, and the second Customer Retrieval for retrieving information about the customer that has been registered. The data is prompted through a Web page and stored in the database. For this application, the database has a single table and four fields: First Name, Last Name, E-mail, and Membership number.

8.5.1 XML in this application

XML plays a key advantage in this application. In this application, XML is used as follows:

- Generating of XML from JavaBeans through a wizard. This creates the XML and XSL accessing JavaBeans. This produces the input and output XSL forms.
- Uses Sun's Java API for XML Processing (JAXP) to produce the XML data from the JavaBean.

- Through Java classes maps the XML data to the org.w3c.dom.Element and also invoking the XSLT processor on the XML document and the XSL document.

The provision of these capabilities, reduces the development time and the learning curve of the developer involved in the development of the application.

8.5.2 Technical overview

The Customer Registration system consists of a two main parts. Its has a registration form where customer data is entered, two Java classes which the customer information is stored and retrieved from the database. Wizards are used extensively.

The starting point of the customer registration systems is a JavaBean, which specifies the getter and setter methods for every attribute of the customer. From the JavaBean, the JavaBean to XML wizard is used to generate the Web tier. The wizard leads through a few steps:

- It creates the XML and XSL files for all the chosen attributes. Two XSL files are produced one for the output and another for the input.
- It creates the Input and Output forms, which can be customized.
- It creates a XML Schema file which describes the customer entity. In this application there are only four attributes: FirstName, LastName, EMail and the membership number.
- An Java class that converts the customer attributes to a DOM representation.
- A Servlet that stores and retrieves the attributes to and from the DOM representation using the Java class.

As a final part of the system, we carry out a unit test to check if the system as been properly built until this stage.

For the second part of the project, we build an Enterprise JavaBean framework. This part of the application also creates the database, and the components required to access the data. The starting point here is the Enterprise bean wizard. The wizard guides the developer through the following steps:

- Creating a Entity bean with Container Managed Persistence (CMP) fields.
- Creating the attributes, and the various classes required for the EJB.
- Creating a mapping between the EJB and the relational database fields. In our application, the database schema is created from the Enterprise bean.

- The wizard only creates a schema file that can be run against the database to create the table and its attributes. However, this may need to be fine-tuned for the application.

An Access bean can also be used. This is created through the Access Bean wizard. An access bean adapts an Enterprise bean to the Java programming model, by hiding the home and remote interfaces from the developer. They provide fast access to Enterprise beans by letting the developer maintain a local cache of Enterprise bean attributes. This wizard also incorporates in JNDI bindings.

The two components are then integrated by customizing the generated Java class and the servlet.

The Java class is modified to reference the EJB. A ‘create’ method is developed to use the EJB’s factory’s ‘create’ method. The servlet is modified to use Java Server Pages (JSP), the HTML forms developed previously being abandoned. The JSPs are generated through the JSP wizard. The servlet’s doGet and doPost methods are modified to interact with the EJB. The doPost is changed to store information while the doGet method is altered to retrieve data from the database given the membership number.

The EJB deployed code is generated through an option, and finally the two projects are tested.

9

Developing XML Web services

In chapter nine we discuss how to develop a simple XML Web service using the Application Developer. To illustrate this we are going to create the Passenger List application divided in two parts.

- Design and develop the Passenger List application using static XML.
- Modify the Passenger List application to use an XML Web service.

9.1 Passenger List application

We separate the development into two phases (Figure 9-1). In the phase one, we develop the Web tier that returns a static XML using a servlet and XSLT Processor. In the phase two, we implement a Web service, which returns the passenger list as a query result.

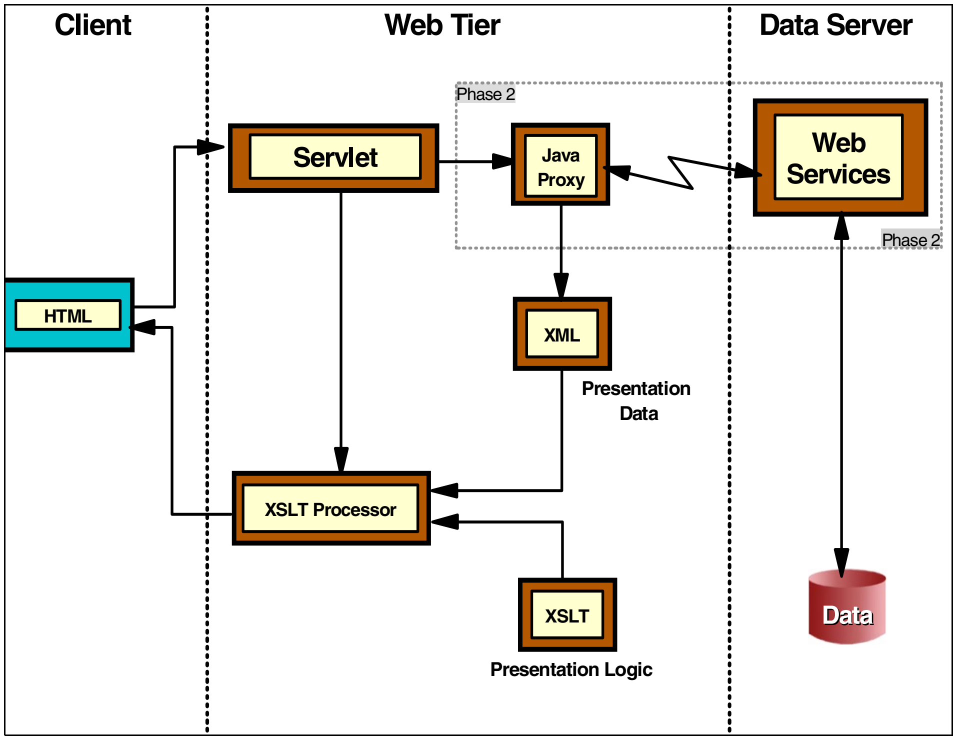


Figure 9-1 Solution overview of the Passenger List application

In the first phase, we start to design the output message (data) format as an XML Schema and output presentation as an XSLT. Then create a servlet which works with them. In the second phase, we generate the Web Service using the XML Schema.

9.1.1 Creating the Web tier

The first part consists in getting the passenger list data from a static XML file and show it as an HTML output page. To do this we need to follow some steps: (See Figure 9-2.)

1. Design the XML Schema

2. Generate a DTD from the XML Schema.
3. Generate a sample XML output.
4. Generate the XML file from the schema and add some data to it.
5. Design an HTML output page.
6. Map the XML and the output presentation
7. Generate an XSL file from the map to transform the XML to HTML, in order to show it as an output page.
8. Develop a simple servlet that reads the XML file and transform it to the HTML output page using the XSL file and XSLT processor.

- Test Passenger List application.

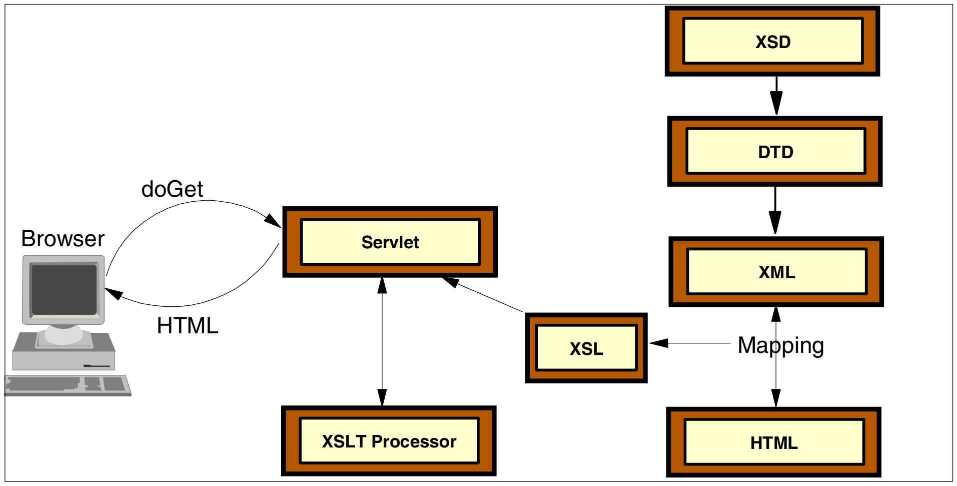


Figure 9-2 Passenger List application

We are going to create two projects to organize XML and Web application and lately Web services files separately. The first project is a simple project called Airline, which will contain the XML files.

Name	**Type**
Airline	Simple project
Travel	Web project

9.1.2 Create the Airline simple project

To create the simple project from the workbench take this steps:

1. Select **File—>New—>Project** and create a Simple Project from the Simple category. Click **Next.**

2. Set the project name as Airline.
3. Click **Finish** to complete creating the project.

9.1.3 Create the Travel Web project

To create a Web project from the workbench take this steps:

1. Select **File—>New—>Project** and create a Web Project from the Web category.
2. Enter the project name and choose J2EE Web Application Project, click **Next**.
3. In J2EE Settings Page create a new Enterprise Application Project and enter `Traveler` as the project name.
4. Click **Next** and then click **Finish** to accept defaults. (See Figure 9-3.)

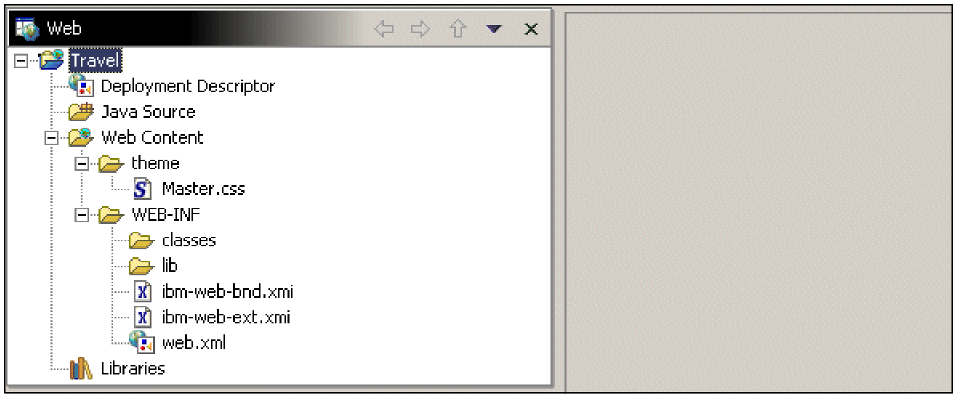


Figure 9-3 Travel Project

9.1.4 Design the XML Schema

We designed the output message format using XML Schema editor. Basically we need some description of the contents of the passenger list of a flight. To keep it simple we are going to design this content as shown in Figure 9-4. Customer element contains name and frequent flyer membership to describe it. Flight element contains flight number and departure time attributes and customer element as a child. A flight can contain multiple customers.

To do this take this steps:

1. Select **Airline** project.
2. Select **File—>New—>XML Schema.**
3. Set schema name as **passengerList.xsd.**

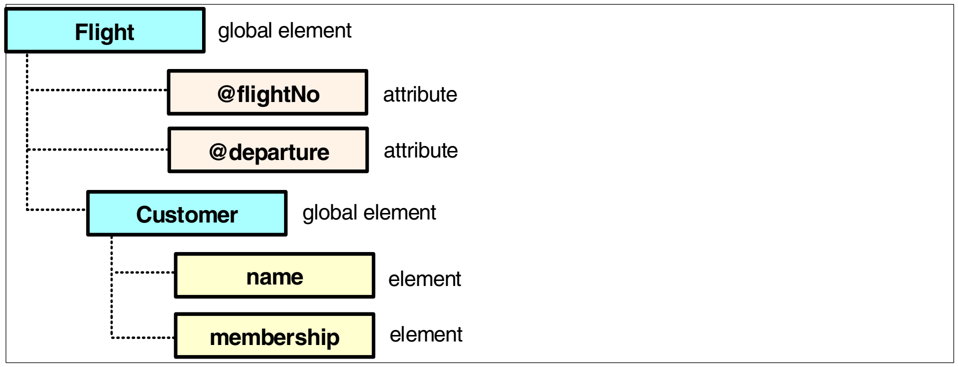
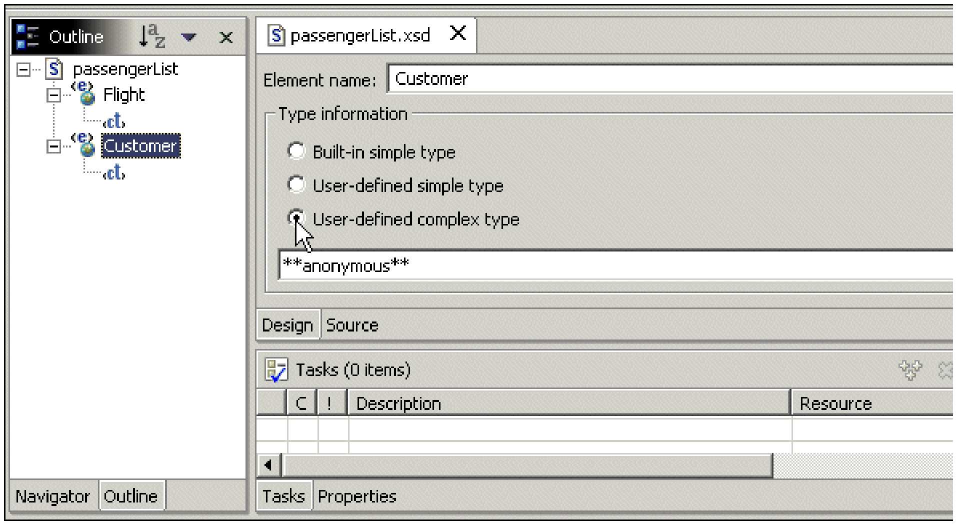


Figure 9-4 PassengerList DOM Tree

The XML Schema editor has a Design view and a Source view. The Outline view and eliminate the need for a knowledge of XSD syntax, but developers who prefer to directly edit the file can work with the source view. The Design view presents a form-based user interface that is synchronized with the outline and Source view. When a part of the schema is selected in the Outline view, it can be changed in the Design view.

Next, we have to add contents to the schema. Take these steps to accomplish this:

1. Add two global elements, `Flight` and `Customer`, and set type information to user-defined complex type.
2. Select **root node**.
3. Right-click **Add Global Element** (Figure 9-5).



Element name: Customer

Type information

( ) Built-in simple type

( ) User-defined simple type

(●) User-defined complex type

anonymous

Figure 9-5   Creating XML Schema

Add Content Model to the complex type.

- Select the **ct.** Right-click **Add Content Model**.

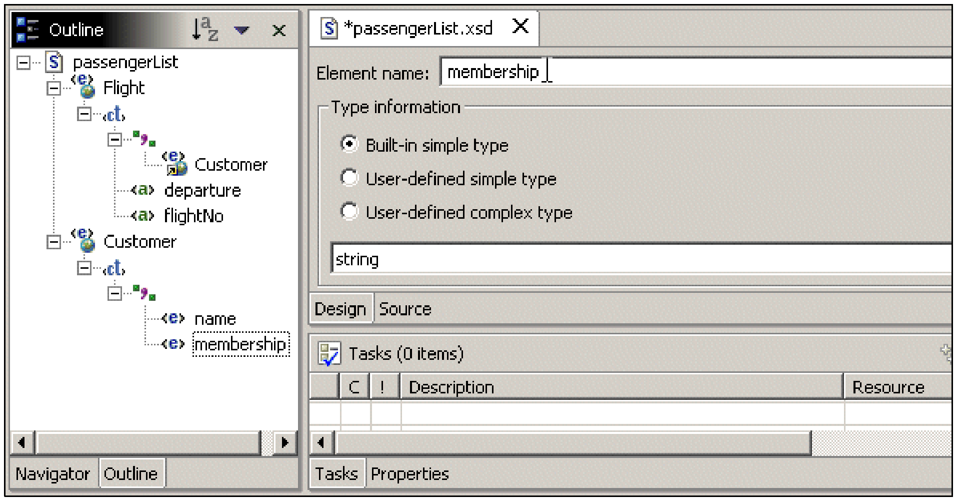
Add an Element Ref. under the model by:

- Select the model right-click **Add Element Ref**.
- Set the reference as customer. Since the flight has one or more customers set the minimum to one and the maximum to unbounded.

Add two attributes to the flight node, flightNo and departure.

- Select **ct** under Flight.
- Right-click **Add Attribute.** Attribute flightNo must be set as required since it is a primary key.
- Next add two child elements to the customer node. Your XML Schema should look like Figure 9-6.

Example 9-1 is showing the actual XML Schema.



Element name: membership

Type information

(●) Built-in simple type
( ) User-defined simple type
( ) User-defined complex type

string

Figure 9-6 XML Schema

Example 9-1 PassengerList.xsd

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:passenger="http://www.airline.com"
             targetNamespace="http://www.airline.com">
    <element name="Flight">
        <complexType>
            <sequence>
                <element ref="passenger:Customer" minOccurs="1"
                             maxOccurs="unbounded"/>
            </sequence>
            <attribute name="flightNo" type="string" use="required"/>
            <attribute name="departure" type="string"></attribute>
        </complexType>
    </element>

    <element name="Customer">
        <complexType>
            <sequence minOccurs="1" maxOccurs="1">
                <element name="name" type="string"/>
                <element name="membership" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

9.1.5 Generate XML file

Now we are going to generate the XML File and add some sample data to it. This data it’s going to be transformed and presented in a HTML output page.

1. Select **passengerList.xsd.**
2. Right-click **Generate—>XML File**.
3. Select **Airline** as the folder name.
4. Enter `passengerList.xml` as the filename, click **Next**.
5. Select **Flight** as root element, select **Create required and optional content,** click **Finish.**

We need to add some sample data to the XML file. (See Example 9-2.)

Example 9-2 Static XML file

```
<?xml version="1.0" encoding="US-ASCII"?>
<passenger:Flight departure="8:00pm" flightNo="Air Canada 700"
                xmlns:passenger="http://www.airline.com"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.airline.com passengerList.xsd ">
  <passenger:Customer>
    <name>John Doe</name>
    <membership>123456</membership>
  </passenger:Customer>
</passenger:Flight>
```

9.1.6 Design an output page

Since XML is not a markup language specialized in presentation, we are going to design an HTML output page to show the data. XSL Transformations (XSLT) will be used to transform the XML to HTML data. To create the XSL, there are several approaches as follows:

1. Design an HTML file, then map to the XML using mapping tool. We can use XSL generator to generate the XSL using the map file.
2. Develop the XSL from scratch. We can use the different wizards in the XSL editor.
3. Design an XHTML file then generate the XSL file.

HTML and mapping approach

Using this approach, we created a DTD file first. Then generate the XML which contains HTML tags. Given the XML definition, we can think of HTML as a sub set of XML. We need to map the static XML file data to HTML tags, in order to be able to use the XML to XML mapping tool, we are going to create the file html.dtd, and generate html.xml. (See Example 9-3.)

Example 9-3 html.dtd

```
<!ELEMENT html (body)>
<!ELEMENT body (h2, hr, table)>
<!ATTLIST body
          bgcolor CDATA #IMPLIED>

<!ELEMENT h2 (CENTER,CENTER)>
<!ELEMENT CENTER (#PCDATA)>

<!ELEMENT table (tr+)>
<!ATTLIST td
          width  CDATA #REQUIRED
          valign CDATA #REQUIRED>

<!ELEMENT tr (td,td)>
<!ELEMENT td EMPTY>
<!ELEMENT hr EMPTY>
```

After designing our html.dtd we generate html.xml file. To generate html.xml file take the following steps:

1. Select **html.dtd** file.
2. Right-click **Generate—>XML file**
3. Enter `Airline` as the folder.
4. Enter `html.xml` as file name, click **Next.**
5. Set html as root element.
6. Select **required and optional content,** click **Finish**.

Example 9-4 shows the generated XML file.

Example 9-4 html.xml source

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html SYSTEM "html.dtd" >
<html>
  <body>
    <h2>
      <CENTER>CENTER</CENTER>
      <CENTER>CENTER</CENTER>
    </h2>
    <hr/>
    <table>
      <tr>
        <td valign="" width=""/>
        <td valign="" width=""/>
      </tr>
    </table>
  </body>
```

```
</html>
```

Generate an XSL file

As we mentioned, we need to have an XSL file to transform. To do this we need design the way that the XML data is going to be mapped to the HTML tags. For this example, we are going to do it,as shown in figure Figure 9-7.

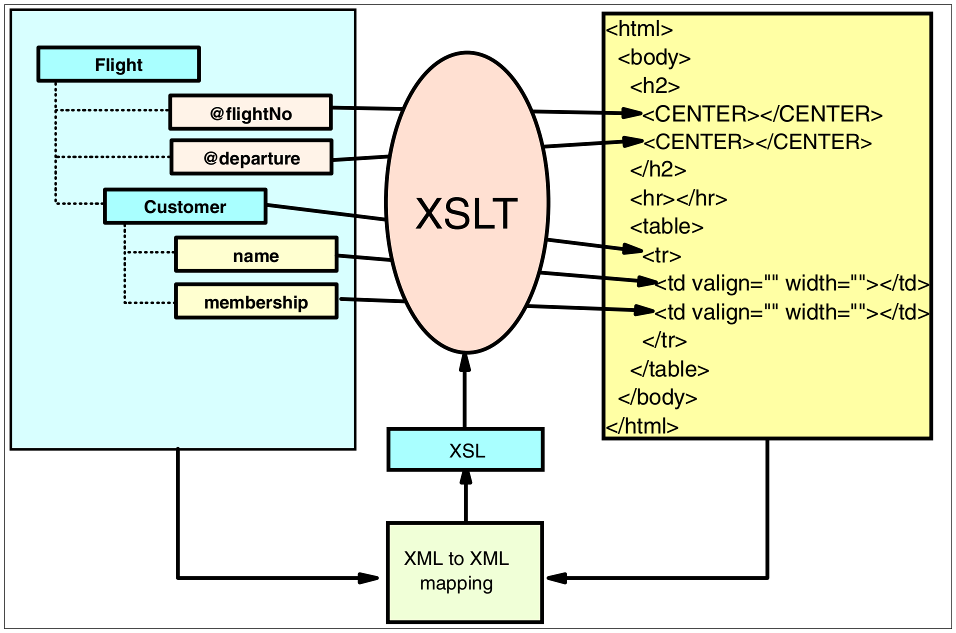
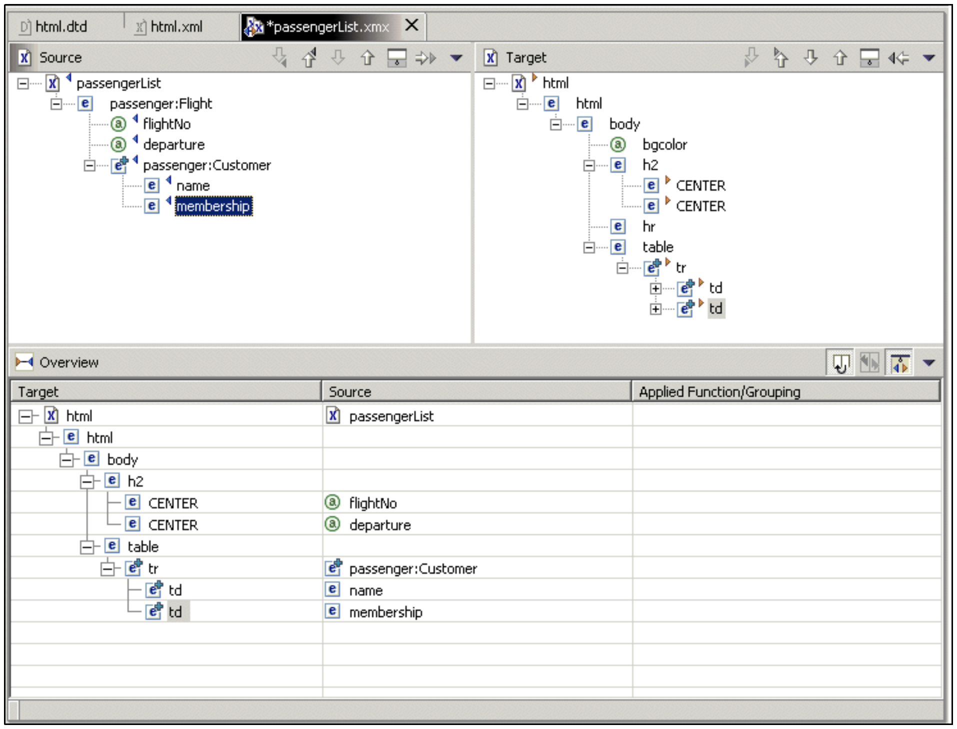


Figure 9-7 XML to HTML mapping

Steps to create the XSL file:

1. Switch to XML Perspective if necessary.
2. Start XML to XML Mapping wizard.
3. Select the folder where it is going to be located.
4. Enter a name for the mapping file passengerList.xmx, click **Next.**
5. Add source XML Schema file passengerList.xsd, click **Next**.
6. Select target **html.xml***,* click **Next.**
7. Select root element, must be HTML. Click **Finish**.

Figure 9-8 shows the XML to XML mapping editor.



```
Source                               Target
passengerList                        html
  passenger:Flight                     html
    flightNo                             body
    departure                              bgcolor
    passenger:Customer                     h2
      name                                   CENTER
      membership                             CENTER
                                           hr
                                           table
                                             tr
                                               td
                                               td
```

Target	Source	Applied Function/Grouping
html	passengerList	
html		
body		
h2		
CENTER	flightNo	
CENTER	departure	
table		
tr	passenger:Customer	
td	name	
td	membership	

Figure 9-8 XML to XML mapping

Mapping XML to HTML

Now we need to map each item from source to the target. To map, click on the source element then drag on to the target.

Map the elements as follows.

Source	**Target**
passengerList	HTML
flightNo	CENTER
departure	CENTER
Customer	tr
name	td
membership	td

To match the root, passengerList must be mapped to HTML. The flight number and departure time will be mapped in the center tag. Since the customers can be appeared as several lines, Customer element should be mapped to tr.

Create an XSL from scratch

Creating an XSL from scratch is also useful. The XSL Editor provides a number of wizards to help you create the content in your stylesheet. This section is describing how you can create an XSL stylesheet from scratch to format the xml data into an HTML table.

To create an HTML document, we will add a template to generate HTML header information as well as to define the output method for the document:

1. Position the cursor to an empty line after the `<xsl:stylesheet>` element in the html.xsl file.
2. In the menu bar, select XSL->HTML Template. This will create an `<xsl:output>` element that will output the result in HTML, and a template that will emit an HTML header with an `<xsl:apply-templates>` rule to will process all the immediate children in the passengerList.xml file.

Next, we will create a template that will produce an HTML table:

1. Position the cursor to an empty line after the `</xsl:template>`.
2. In the menu bar, select XSL->HTML Table. This will bring up the XSL Table Wizard.
3. Select the **Flight element** as the context node for building the HTML table. This implies that all the children of the Flight element will be added as columns in the table.
4. Check the Wrap table in a template check box to indicate that we want to wrap this table in a new template.
5. Check the Include Header check box to indicate that we want to include a header in this table.
6. Optionally, click **Next** to go to the next page to add a table border and background color for the table.
7. Click **Finish.** This will create a new Flight template in your html.xsl file that will produce an HTML table.

Generating XSL from XHTML

This will enable you to separate out the presentation logic from the dynamic data in an existing HTML document. It extracts the data into an XML file and the presentation data into two XSL files. Once the separation is completed, you can use XSLT technology to combine new data that is defined in XML format with the generated XSL files to create new HTML Web pages. To use this technology requires the following steps:

1. Prepare the HTML file for generation by converting it into a well-formed XML file with a `.xhtml` extension.

2. Create a template file that contains the annotation tags.
3. Invoke the Generation wizard.

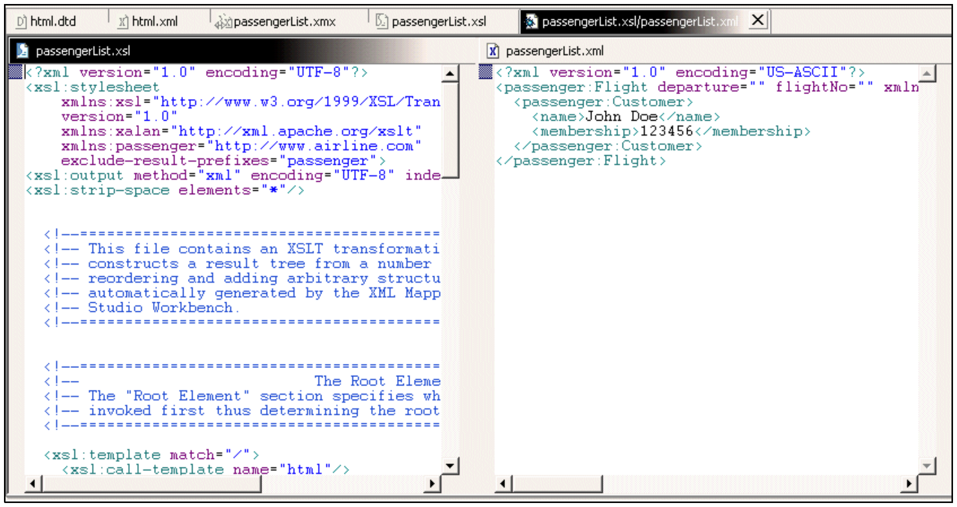
One of Application Developer example is showing how to generate XSL from HTML using XHTML technology step by step. To see in detail, install HTMLToXSL Project using New->Example wizard.

9.1.7 Testing the XSL

To test the XSL, select passengerList.xml then, use Apply XSL as HTML from the context menu:

1. Select **PassengerList.xml.**
2. Right-click **Apply XSL -> as HTML.**
3. Select **workbench projects.**
4. Select **passengerList.xsl.**
5. Click **Finish.**

Then the XSL Debugger is opened. Use Open the browser on the transformation result toolbar button to view the result (Figure 9-9).



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Tran
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt"
    xmlns:passenger="http://www.airline.com"
    exclude-result-prefixes="passenger">
<xsl:output method="xml" encoding="UTF-8" inde
<xsl:strip-space elements="*"/>


  <!--========================================
  <!-- This file contains an XSLT transformati
  <!-- constructs a result tree from a number
  <!-- reordering and adding arbitrary structu
  <!-- automatically generated by the XML Mapp
  <!-- Studio Workbench.
  <!--========================================


  <!--========================================
  <!--                          The Root Eleme
  <!-- The "Root Element" section specifies wh
  <!-- invoked first thus determining the root
  <!--========================================

  <xsl:template match="/">
    <xsl:call-template name="html"/>
```

```
<?xml version="1.0" encoding="US-ASCII"?>
<passenger:Flight departure="" flightNo="" xmln
  <passenger:Customer>
    <name>John Doe</name>
    <membership>123456</membership>
  </passenger:Customer>
</passenger:Flight>
```

Figure 9-9 Testing the XSL

9.1.8 Developing the servlet

The servlet that we are going to develop is simple. It implements the *doGet* method, reads the XML file generated in 9.1.5 “Generate XML file” on page 184, and transforms it to an HTML file using the XSL file generated in “Generate an XSL file” on page 186. See Example 9-5. Create the GetPassengerListServlet in

Travel project. We used com.ibm.itso package. WAS_V4_XALAN variable is required in the Java Build Path to create this servlet.

Example 9-5 GetPassengerList Server doGet method

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, java.io.IOException {

         String xslFile = "/WEB-INF/xsl/passengerList.xsl";
         String xmlFile = "/WEB-INF/passengerList.xml";
         try
         {
            resp.setContentType("text/html");
            PrintWriter out = resp.getWriter();
            TransformerFactory tFactory = TransformerFactory.newInstance();
            Transformer transformer = tFactory.newTransformer(
                     new StreamSource(
               getServletContext().getResource(xslFile).toExternalForm()));
             StreamSource source = new StreamSource(
                  getServletContext().getResource(xmlFile).toExternalForm());
            transformer.transform(source,new StreamResult(out));
         }
         catch(Exception ex)
         {
            System.out.println("Error in doGet "+ex.toString());
         }
   }
```

The ServletContext provides own getResource method. It returns the resource located on the location of the current Web application. To load files from the current WAR file, you need to specify the file name, start with **/WEB-INF/**. In this case you need to copy the following files into the /WEB-INF/ directory:

- passengerList.xml (sample data)
- PassengerList.xsd (required by passengerList.xml)
- passengerList.xsl (create /WEB-INF/xsl directory and put it inside)

Using JAXP

As we mentioned about JAXP, it provides a standard Java interface to many XSLT processors. We used three JAXP classes in our servlet.

- **javax.xml.transform.Source**

  This interface is used to read XML and XSLT file. This can read from a stream type object such as an InputStream, or a Reader.

- **javax.xml.transform.TranfrormerFactory**

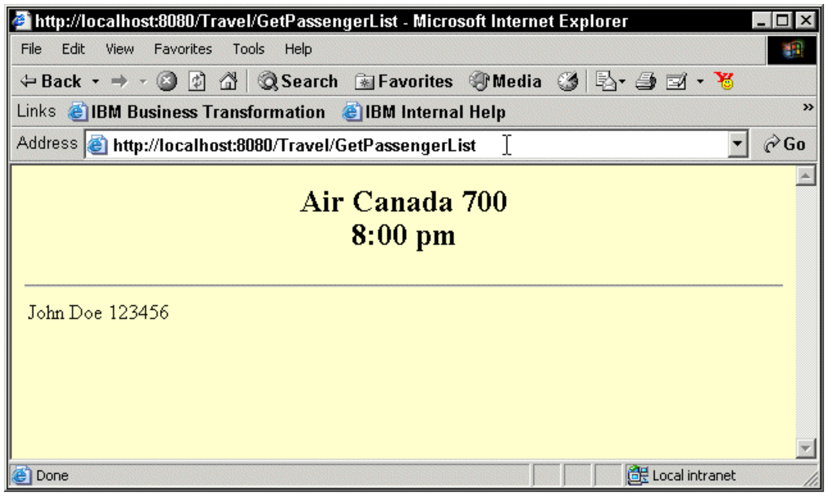
  This is responsible for crating transformer object. TrasformerFactory is abstract, and its new instance method is used to instantiate an instance of Transformer class.

- **javax.xml.transform.Transformer**

  This instance is used to perform the actual transformation. It is not tread-safe, so in a threaded servlet environment; this should be an instance variable. By this restriction, you need to load XSLT on every request.

9.1.9 Test the passenger list application

In this section we test the passenger list application. You need to copy the passengerList.xsl to the /WEB-INF/xsl folder, and copy the passengerList.xml to the /WEB-INF folder in the Travel project. To test the application we have to start a WebSphere test environment, and call the servlet from the browser. Since the servlet does not need an input parameter, and implements *doGet* method, it can be called using the URL directly.


http://localhost:8080/Travel/GetPassengerList - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Business Transformation IBM Internal Help
Address http://localhost:8080/Travel/GetPassengerList
Go
Done
Local intranet

Air Canada 700
8:00 pm

John Doe 123456

Figure 9-10 passengerList result page

9.1.10 Compiling XSL

Since the Transformer is not a thread-safe, we need to instantiate the Transformer on every request. Actually, in the doGet method, it is loading XSL file from the stream every request. The XSLT Complier enables that it load in to the Java vm as an object and reuse in the servlet instances. JAXP supplies the template's interface to provide consistency. This interface has newTransformer method. The instance of the Templates can be instantiate using newTemplates method of TransformerFactory class:

```
TransformerFactory transFact = TransformerFactory.newInstance();
String xslName = "/WEB-INF/xsl/Passenger.xsl";
URL stylesheetURL = getServletContext().getResource(xslName);
String xsltSystemID = stylesheetURL.toExternalForm();
template = transFact.newTemplates(new StreamSource(xsltSystemID));
```

Using the templates, we can load and compile the XSL into the memory while we are initializing the servlet. Doing this in the init method, the servlet can hold the Templates as a global variable. To get the instance of Transformer, use newTransformer method of the template:

```
Transformer transformer = template.newTransformer();
```

9.2 Creating a Web service

In the previous application we were extracting the information from an static XML file. Now, we are going to extract the information from the database using an XML Web service. We are going to generate a Web service using the previous generated XML files, so the flow would be as shown in Figure 9-11.

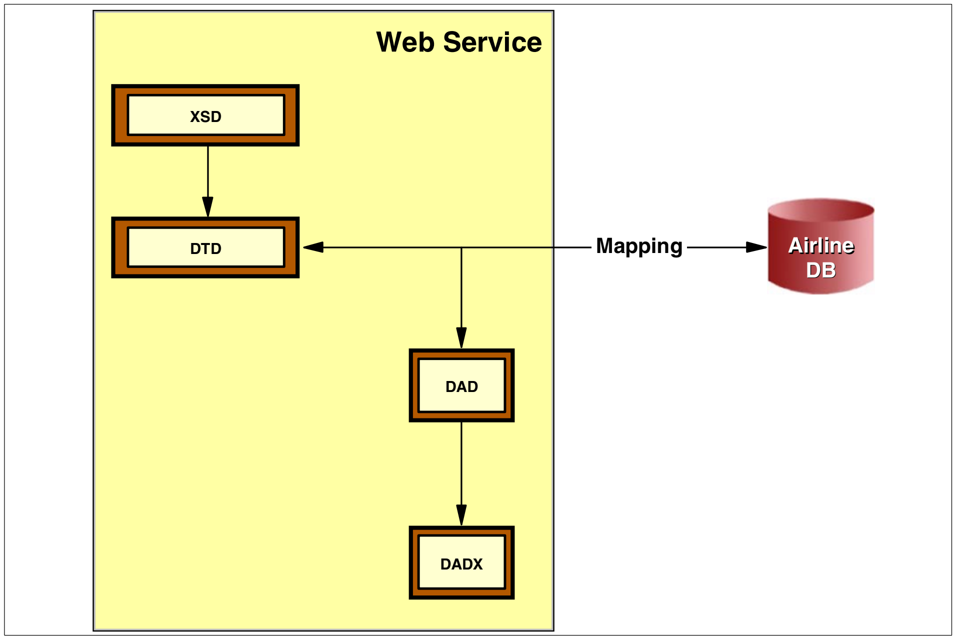


Figure 9-11 XML Web service

We are going to discuss:

- Creating the database tier
- Generating DTD from XML Schema
- Creating DAD file using RDB to XML mapping
- Creating Web service from DADX file
- Testing a Web service
- Modifying the Passenger List application to use Web service

9.2.1 Create the database tier

See database specification in figures below.

Column name	Data type	Maximum length
Flight	VarChar	30
Name	VarChar	20
Membership	VarChar	10

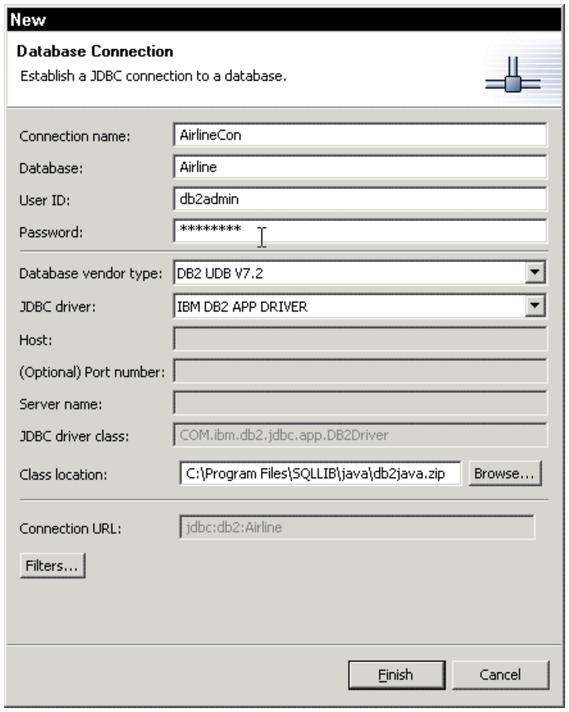
Column name	Data type	Maximum length
FlightNo	VarChar	30
AirCraft	VarChar	10
Departure	VarChar	10
Arrival	VarChar	10

In order to create the necessary files to access the database, we need to create a database connection. To add a database connection to your project:

1. Open the database perspective.
2. Click **Window—>Open Perspective—>Data**.
3. Select DBServers tab.
4. Right-click **New Connection.**

The connection must be configured as shown in Figure 9-12 on page 194. After creating the connection import it to your project:

1. Select the connection.
2. Right-click **Import to folder**.
3. Select your project's folder and click **OK.**
4. Click **Finish.**


New

Database Connection

Establish a JDBC connection to a database.

Connection name:	AirlineCon
Database:	Airline
User ID:	db2admin
Password:	********
Database vendor type:	DB2 UDB V7.2
JDBC driver:	IBM DB2 APP DRIVER
Host:	
(Optional) Port number:	
Server name:	
JDBC driver class:	COM.ibm.db2.jdbc.app.DB2Driver
Class location:	C:\Program Files\SQLLIB\java\db2java.zip
Connection URL:	jdbc:db2:Airline

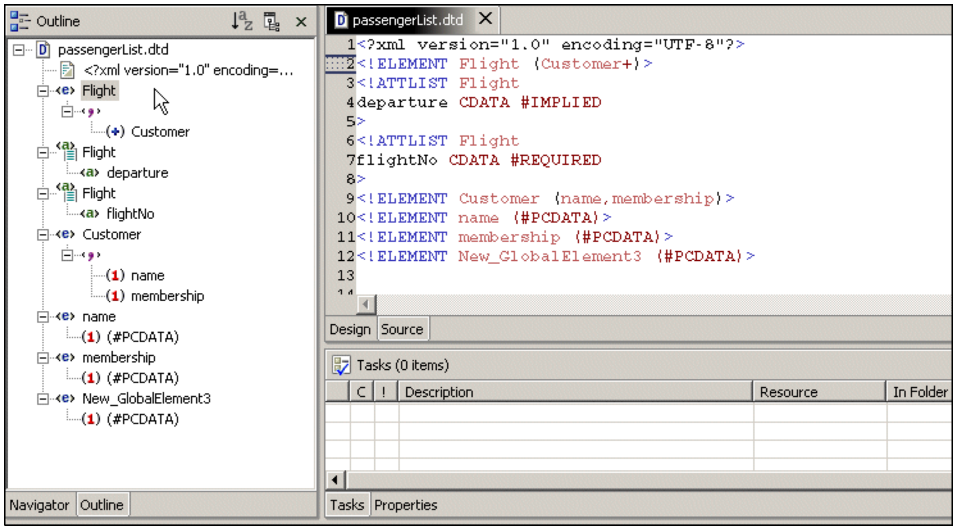
Figure 9-12 Database Connection panel

9.2.2 Generate DTD file

In addition to the XSD Editor, WebSphere Studio Application Developer also has the DTD Editor. We refer to both XSD and DTD as schemas. Application Developer has extensive support for schemas, including tools for validating schemas, converting between XSD and DTD, generating schemas from sample XML instance documents, generating sample XML documents from schemas, and generating Java classes and relational database schemas from XML Schema.

Now that we have our XML Schema we have to generate the DTD file since DB2 XML Extender only supports DTD. We need the DTD file because it is going to be useful when mapping our output message format elements to the database fields in “Generate an XSL file” on page 186.

Select the Navigator view and right-click **passengerList.xsd—>Generate—>DTD** (Figure 9-13).



```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Flight (Customer+)>
<!ATTLIST Flight
departure CDATA #IMPLIED
>
<!ATTLIST Flight
flightNo CDATA #REQUIRED
>
<!ELEMENT Customer (name,membership)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT membership (#PCDATA)>
<!ELEMENT New_GlobalElement3 (#PCDATA)>
```

Figure 9-13 Generated DTD File

9.2.3 Loading DTD into XML Extender

Before you start, you need to load the DTD into DB2 XML Extender to use. If you did not initialize the Extender, you can with the following commands:

```
db2 "bind '%DB2EXTENDER%\bnd\@dxxbind.lst'"
db2 "bind '%SQLLIB%\bnd\@db2cli.lst'"
```

Note: Default directory of DB2 XML Extender is c:\dxx and SQLLIB is c:\Program Files\SQLLIB.

You need to enable the database to use with DB2 XML Extender. Run following command to enable the airline database:

```
dxxadm enable_db AirLine
```

Now you need to install the passengerList.DTD into DB2 to run the PassengerList as a stored procedure under DB2 XML Extender:

```
db2 "insert into db2xml.dtd_ref values('passengerList.dtd',
db2xml.XMLClobFromFile('passengerList.dtd'), 0, 'user1', 'user1', 'user1')"
```

Note: PassengerList.dtd must be located in local folder. DB2 does not load it from the networked drive.

9.2.4 Creating DAD file using RDB to XML mapping

After defining the schema for the output message and setting our database, we can define the listPassenger operation that retrieves the information from the database. Here we plan to use the DB2 XML Extender run-time component to execute the operation and generate the XML result. We must define a DAD file for the retrieval operation.

The *Document Access Definitions* (DAD) file, is an XML formatted document, it is used to associate the XML document structure to a DB2 database. Basically, it provides the mapping between the elements or attributes of an XML document and the table columns, and the details of how a request for an XML document is to be handled.

To see how to map the database to the DTD file see Figure 9-14.

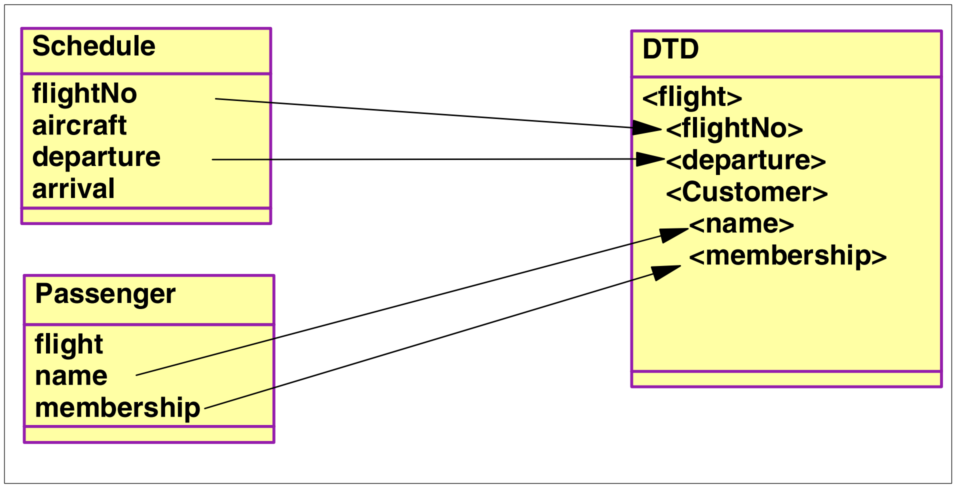


Figure 9-14 RDB to XML mapping

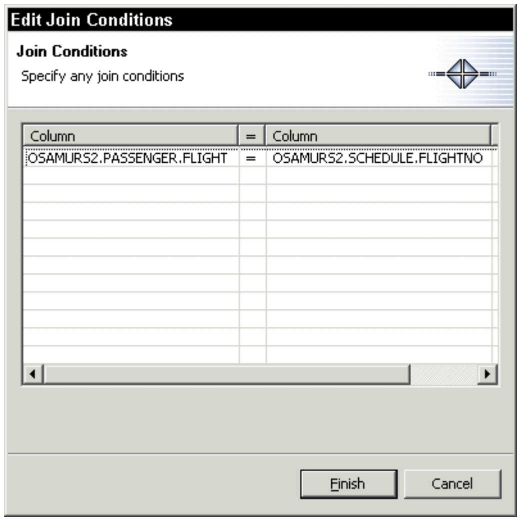
The RDB to XML Mapper tool, shown in Figure 9-16, helps us define the mapping from the database to the XML result:

1. Turn to XML perspective if necessary.
2. Select the project and right-click **New—>RDB to XML Mapping**.
3. Enter `passengerList.rmx` as the name of the file, click **Next**.
4. Select **RDB Table to XML Mapping,** click **Next**.
5. Select the source tables **Passenger** and **Schedule**, click **Next.**
6. Then select the target DTD file **passengerList.dtd***,* select **Flight** as the root element, and click **Finish**.

To define the mappings we select the column of the database in the tables view and its corresponding XML element or attribute in the XML view and then add the mapping to our definition. Since we are using two tables we have to specify the join conditions.

- Click **Mapping—>Edit Join Conditions**.

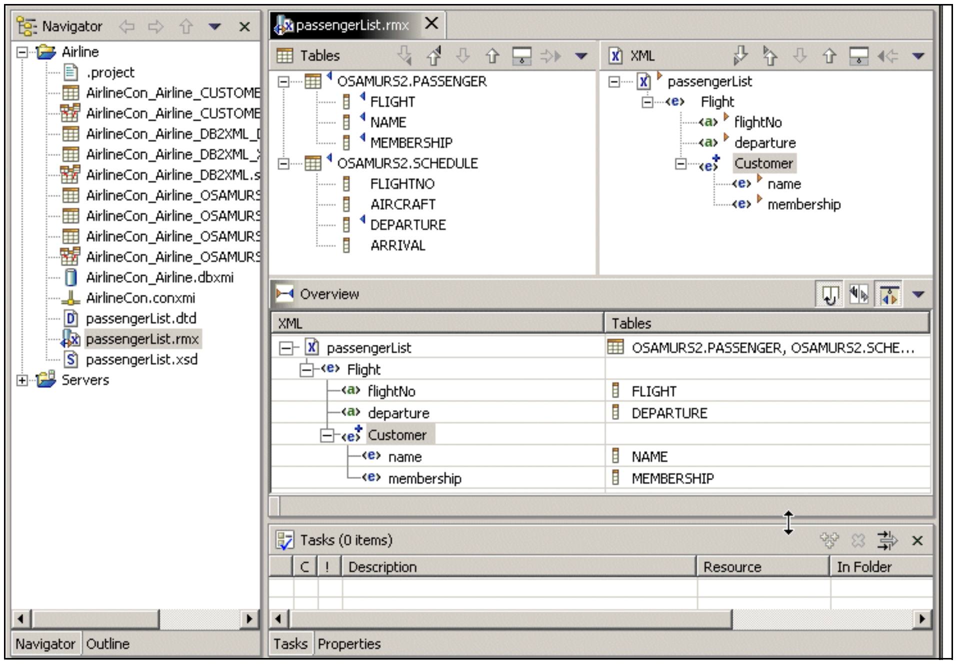
Remember that we are using *flightNo* as a primary key, so they must match in both tables. See Figure 9-15.



Column	=	Column
OSAMURS2.PASSENGER.FLIGHT	=	OSAMURS2.SCHEDULE.FLIGHTNO

Figure 9-15 Edit Join Conditions

The complete definition is stored in *passengerList.rmx,* which is an abstract representation of the mapping.



XML	Tables
passengerList	OSAMURS2.PASSENGER, OSAMURS2.SCHE...
Flight	
flightNo	FLIGHT
departure	DEPARTURE
Customer	
name	NAME
membership	MEMBERSHIP

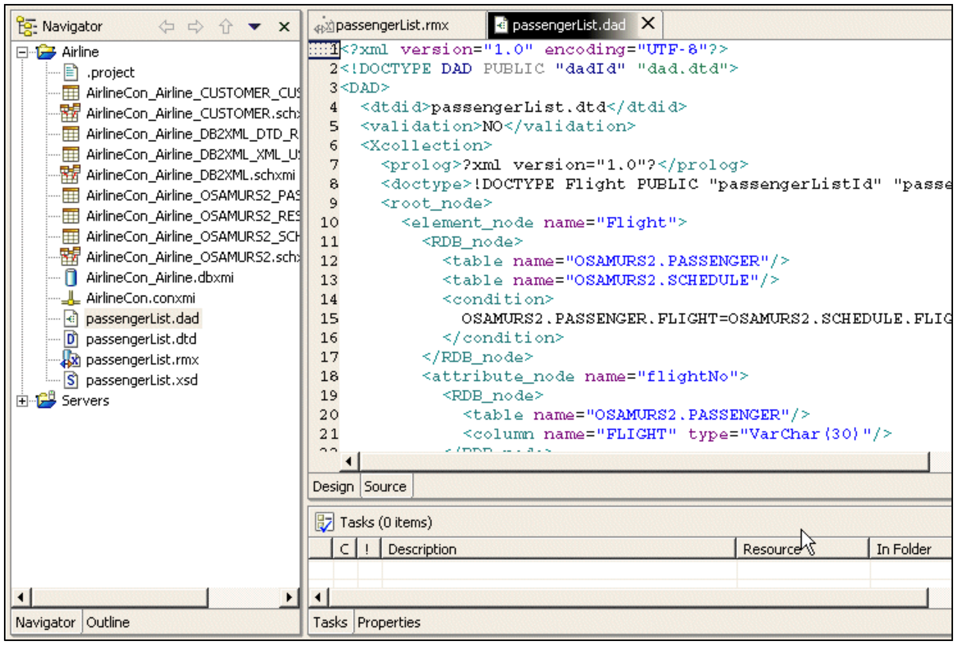
Figure 9-16 RDB to XML mapping

The mapper generates a concrete mapping in a DAD file called passengerList.dad, for the DB2 XML Extender.

In the Navigator view:

1. Select **passengerList.rmx.**
2. Right-click Generate DAD.
3. Set `passengerList.dad` as the name of the file.
4. Choose your project folder, then click **Next.**
5. In *Dad Generation Advanced Options* and *Generate Test Harness,* click **Next** to accept defaults.
6. Click **Finish**.

The wizard will show you the generated DAD file. See Figure 9-17.


Navigator
Airline
.project
AirlineCon_Airline_CUSTOMER_CUS
AirlineCon_Airline_CUSTOMER.sch
AirlineCon_Airline_DB2XML_DTD_R
AirlineCon_Airline_DB2XML_XML_U
AirlineCon_Airline_DB2XML.schxmi
AirlineCon_Airline_OSAMURS2_PAS
AirlineCon_Airline_OSAMURS2_RES
AirlineCon_Airline_OSAMURS2_SCH
AirlineCon_Airline_OSAMURS2.sch
AirlineCon_Airline.dbxmi
AirlineCon.conxmi
passengerList.dad
passengerList.dtd
passengerList.rmx
passengerList.xsd
Servers
Navigator
Outline
passengerList.rmx
passengerList.dad
Design
Source
Tasks (0 items)
C
!
Description
Resource
In Folder
Tasks
Properties

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD PUBLIC "dadId" "dad.dtd">
<DAD>
  <dtdid>passengerList.dtd</dtdid>
  <validation>NO</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE Flight PUBLIC "passengerListId" "passe
    <root_node>
      <element_node name="Flight">
        <RDB_node>
          <table name="OSAMURS2.PASSENGER"/>
          <table name="OSAMURS2.SCHEDULE"/>
          <condition>
            OSAMURS2.PASSENGER.FLIGHT=OSAMURS2.SCHEDULE.FLIG
          </condition>
        </RDB_node>
        <attribute_node name="flightNo">
          <RDB_node>
            <table name="OSAMURS2.PASSENGER"/>
            <column name="FLIGHT" type="VarChar(30)"/>
```

Figure 9-17 Generated DAD file

9.2.5 Create the Web Service from DADX file

We are going to create a Web Service to encapsulate the extraction of the passenger list from the database. Web Service example which lists all the passengers that have tickets for a flight. Here we take a more top-down approach by first designing the format of the output message as an XML Schema. We then map the schema elements with the database fields and express the mapping as a DAD file that can be executed by the DB2 XML Extender. The DB2 XML Extender allows us to handle XML documents that have complex hierarchical structures. We then work on a user interface to display the output into HTML and test its behavior.

The DB2 XML Extender provides stored procedures that can execute DAD files. The Web Service files are going to be in a different project called *Travel.*

DAD Extension

DAD Extension (DADX) is an XML technology for rapidly creating Web services that access relational databases such as DB2 UDB. When combined with the DB2 XML Extender, DADX supports mapping relational data into complex XML documents and storing XML documents in the database. DADX consists of an XML document format and a Java runtime component that work with Apache

SOAP 2.2 and that run on J2EE compliant application servers like WebSphere and Tomcat. To create a new Web Service, the developer authors SQL statements in a DADX document and deploys it to the application server. The DADX runtime executes SOAP requests sent to the new service and provides additional support including HTTP GET and POST bindings, test page and WSDL generation, and translation of DTD into XML Schema. This document describes the DADX document format and runtime. In this section, we will generate the DADX from the DAD.

Web services and DB2 XML Extender

The DB2 XML Extender makes it easy to create XML applications using DB2. DB2 XML Extender consists of a set of stored procedures, user defined types (UDT) and user defined functions (UDF) that enable an application programmer to store and retrieve XML data using DB2. DB2 XML Extender allows XML documents to be stored intact, and optionally indexed in side tables, using the XML Column access method, or as a collection of relational tables using the XML Collection access method. DXX uses a DAD to define the mapping between XML and relational data.

Web services are XML based application functions that can be invoked over the Internet. It is, therefore, natural to use DB2 XML Extender to implement Web services. This document specifies a DADX that makes it easy to create Web services using DB2 XML Extender. A DADX document specifies how to create a Web Service using a set of operations that are defined by DAD documents and SQL statements. A Java component, the DxxInvoker, provides the runtime support for invoking DADX documents as Web services in Apache Simple Object Access Protocol (SOAP) 2.2 which is supported by WebSphere Application Server and other J2EE servlet engines.

The DADX Group

The resources for all DADX Web Service groups are stored in the directory WEB-INF/classes/groups where WEB-INF is the directory used by J2EE Web applications to store resources that are not directly available to HTTP requests. This means that users cannot see the contents of your DADX files. DADX files contain the implementation of the Web services, and are therefore, similar to Java classes.

The classes directory is part of the Java class path for the Web application. This means that your DADX files can be loaded by the Java class loader and that your Web application can execute directly from its WAR file if your application server supports that mode of operation.

Within the groups directory each group of DADX Web services is stored in a directory with the same name as its servlet instance. The DxxInvoker servlet determines where to find DADX files by looking for a directory that matches its servlet name.

Now we are creating a Web Service DADX Group configuration in the Travel Project to contain all the Web Service files. A DADX Group contains connection (JDBC and JNDI) and other information that is shared between DADX files within the group:

1. Select the **Travel** project, click **File—>New—>Other**.
2. In the New Window, create Web Services DADX Group Configuration from the Web Services category. Click **Next.**
3. Select **Travel** project, click **Add Group**, enter `TravelGroup` as the name, and click **Finish.**

The group.properties File

The database connection information, and other parameters, are defined in the group.properties file for the group which, in our example, is stored in the WEB-INF/classes/groups/travelGroup directory.

The group.properties file is a standard Java properties files. The properties have the following meanings:

Properties in group.properties:

initialContextFactory	The Java class name of the JDNI initial context factory that is used to locate the DataSource for the database.
datasourceJNDI	The JNDI name of the DataSource for the database.
dbDriver	The Java class name of the JDBC driver for the database.
dbURL	The JDBC URL of the database.
userID	The user ID for the database.
password	The password for the database.
namespaceTable	The resource name of the namespace table.
autoReload	The boolean automatic reloading mode.
reloadIntervalSeconds	The integer automatic reloading time interval in seconds.
groupNamespaceUri	The URI prefix for automatically generated namespaces for the WSDL and XSD documents.
enableXmlClob	The boolean mode for enabling the XML CLOB stored procedures.

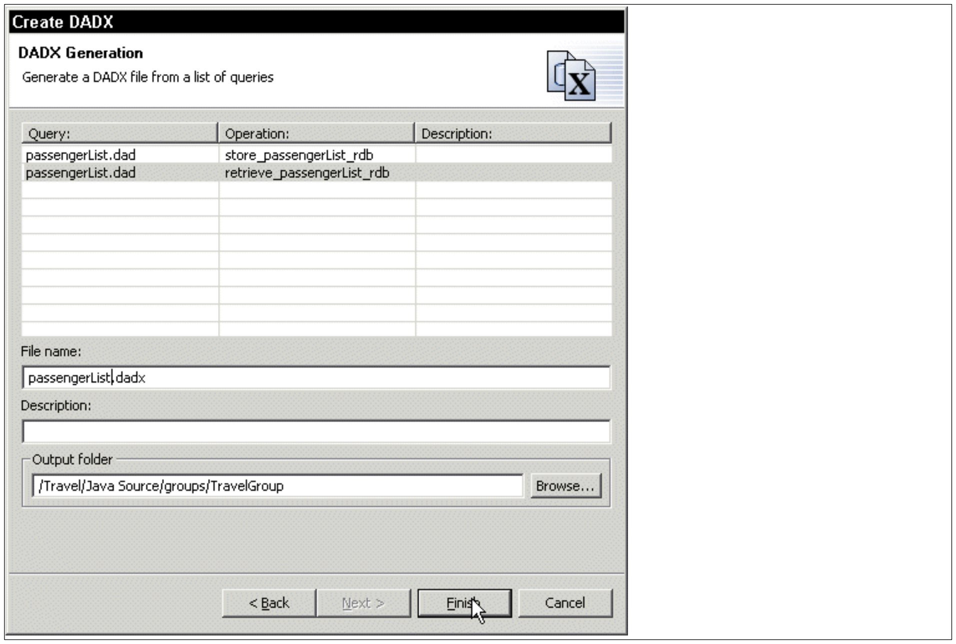
Tip: Since the group.properties file may contain a user ID and password, it should be stored in a secure file system to prevent unauthorized access. In addition, the password can be stored in an encoded format to add further security. The encoding makes the password look like a long, random string of characters that are hard to memorize. This precaution prevents the password from being exposed to coworkers who might be looking over your shoulder while you edit the file. However, encoding is no substitute for proper file access control since the encoded password can be easily decoded.

Generating the DADX

Copy passengerList.DTD and passengerList.DAD files from Airline project into the TravelGroup folder in Travel project.

Now we are going to create the DADX file:

1. Switch to resource perspective if necessary.
2. Select TravelGroup folder, click **File—>New—>Other**.
3. In the New window, create a DADX file from the Web Services category click **Next.**
4. Since we are not using a query we can skip the Select SQL statements by clicking **Next.**
5. Add passengerList.dad file from the TravelGroup folders and click **Next**.
6. It is not mandatory but, we can change the name of the retrieve and store operations. Enter **`passengerList.dadx`** as the name of the file, and modify the name of the store and retrieve operations to `store_passengerList_rdb`, and `retrieve_passengerList_rdb`, respectively as shown in Figure 9-18.


Create DADX

DADX Generation

Generate a DADX file from a list of queries

Query:	Operation:	Description:
passengerList.dad	store_passengerList_rdb	
passengerList.dad	retrieve_passengerList_rdb	

File name: passengerList.dadx

Description:

Output folder: /Travel/Java Source/groups/TravelGroup

Figure 9-18 DADX Generation

DADX operations

The proceeding discussion used a DADX document that employed a three types of operation: query, update, and retrieveXML. The XML Collection operations use the DXX stored procedures. The SQL operations use normal SQL SELECT, UPDATE, INSERT, DELETE and CALL statements, and can be used for XML Column operations by employing the DXX UDTs and UDFs. When using the SQL operations, parameters may be defined using XSD elements as well as simple types. For the query operation, XSD elements may be associated with the column values in the result set. For call operations, parameters may be declared as in, out, or in/out.

XML Collection operations:

- retrieveXML
- storeXML

SQL operations:

- Query
- Update
- Call

<retrieveXML>

The retrieveXML operation generates zero or one XML documents from a set of relational tables using the XML Collection access method.

This operation is currently implemented by the dxxGenXML or dxxRetrieveXML stored procedures, or the new dxxGenXMLClob and dxxRetrieveXMLClob depending on the value of the useXmlClob property defined in the group.properties file.

The new stored procedures are faster and more portable but differ in that they return at most one document. To make the semantics of the operation independent of the stored procedures used to implement them, only a single document is requested when using the old stored procedures. This change affects the XML Schema generated for the output in preliminary versions of this specification (maxOccurs="unbounded" previously, but maxOccurs="1" now).

This operation is implemented by dxxGenXMLClob since a <DAD_ref> element is used but would be implemented by dxxRetrieveXMLClob if a <collection_name> element had been used.

The dxxGenXMLClob stored procedure takes as arguments a DAD document and an optional override. The override can be either SQL or XML. If the override is SQL then the DAD must use SQL mapping. If the override is XML then the DAD must use RDB_node mapping. Our DAD is using RDB_node as well. If an override is defined, then the operation can also define one or more <parameter> elements. The parameters form the input message. DxxInvoker extracts the parameters from the input message, validates them, and then substitutes them into the override by replacing the parameter markers, e.g. :flightNo.

The dxxGenXMLClob returns zero or one XML documents that satisfy that DTD referenced by the DAD file. DxxInvoker invokes the dxxGenXMLClob stored procedure, retrieves the result set and places it in the output message.

Since we want to query the database using a search criteria, we need to use an XML_override tag and a XPath expression in the DADX file to do it. The XML_override tag allows you to specify your XPath expression for doing a query. We need to specify that we are querying using flight number, so we customize our DADX file as shown in Example 9-6:

1. Add an <dadx:XML_override> element in the DADX editor
2. Add an XPath expression to specify the parameter.

Example 9-6 Updated DADX file

```
<dadx:retrieveXML>
            <dadx:DAD_ref>passengerList.dad</dadx:DAD_ref>
            <dadx:XML_override>
              /Flight/@flightNo=:flightNo
            </dadx:XML_override>
            <dadx:parameter name="flightNo" type="xsd:string"/>
</dadx:retrieveXML>
```

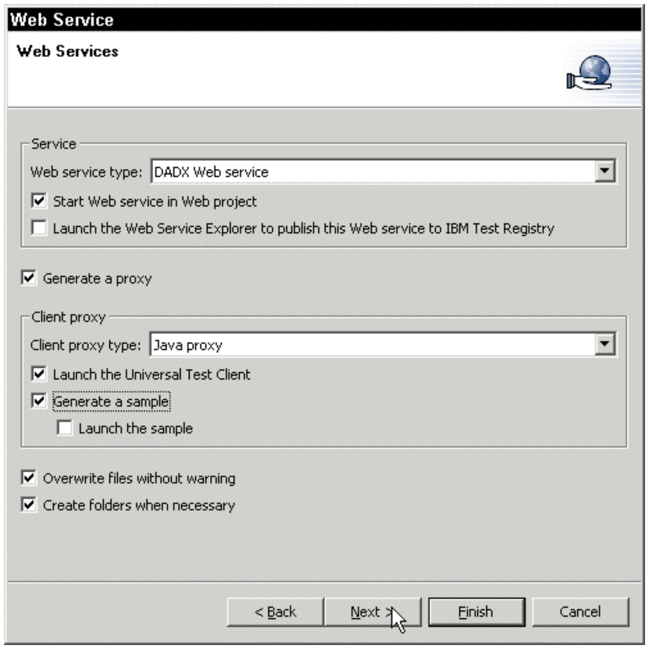
<storeXML>

The storeXML operation stores an XML document in a set of relational tables using the XML Collection access method. The DAD document that defines the collection must use the RDB_node mapping method. This operation is implemented by dxxShredXML if a <DAD_ref> element is used and by dxxInsertXML if a <collection_name> element is used. The dxxShredXML stored procedure takes a DAD document and an XML document as input.

Deploying the Web service

having the dadx file ready, we can create and deploy the web service. We use the Web services wizard, to generate the Java client proxy that we are using in a client application:

1. Select **TravelGroup** folder.
2. Copy passengerList.dad into the TravelGroup directory.
3. Right-click **New—>Other**, and in the New window create a Web Service from Web Services category, click **Next.**
4. Select **DADX Web Service** as Web service type.
5. Set Java proxy as client proxy type, set check boxes, as shown in Figure 9-19, and click **Next.**


Web Service

Web Services

Service

Web service type: DADX Web service

☑ Start Web service in Web project

☐ Launch the Web Service Explorer to publish this Web service to IBM Test Registry

☑ Generate a proxy

Client proxy

Client proxy type: Java proxy

☑ Launch the Universal Test Client

☑ Generate a sample

☐ Launch the sample

☑ Overwrite files without warning

☑ Create folders when necessary

Figure 9-19 Web Service settings

6. In the Web Service setup window, click **Next** to accept defaults.
7. Then choose the DADX file in which the Web service is based, click **Next.**
8. After doing so, we need to set the database name, the user ID and the password to access the database. Also, change the jdbc driver if necessary, click **Next.**
9. In the Binding Proxy Generation, click **Finish** to accept defaults.

We also need to modify the *namespacetable.nst* file in order to add a mapping entry.

10. Double-click **namespacetable.nst** in your TravelGroup folder.
11. Add a mapping entry for passegerList.dtd, as shown in Example 9-7.

Example 9-7 namspacetable mapping entry

```
<mapping dtdid="passengerList.dtd"
              namespace="http://www.airline.com/passengerList.dtd"
              location="/passengerList.dtd/XSD"/>
```

9.2.6 Test the Web Service

A Web Service defined by a DADX file is self-describing. It dynamically generates a documentation and test page, WSDL documents, and XML Schema. The following HTTP GET URL requests the documentation and test page:

```
http://localhost:8080/Travel/TravelGroup/passengerList.dadx/TEST
```

The following HTTP GET URL requests the WSDL description of the service:

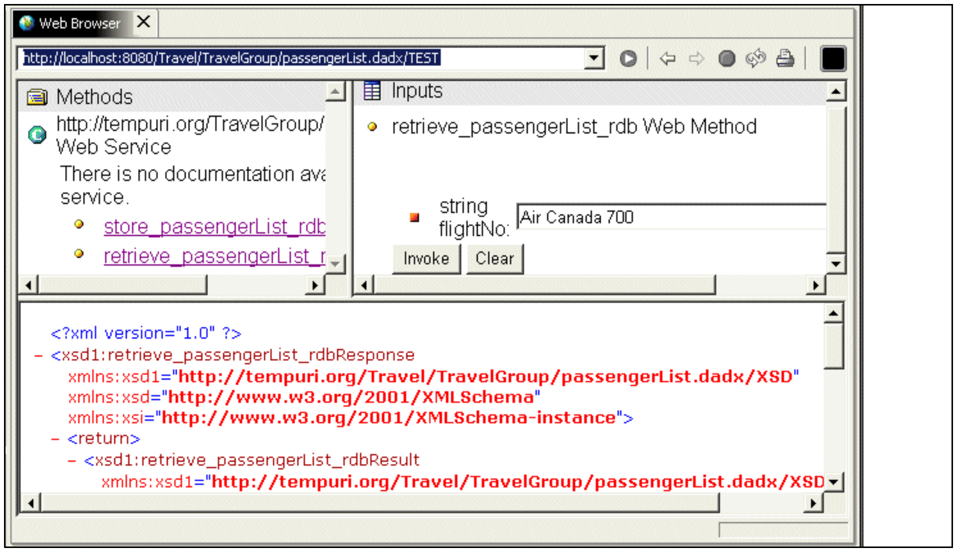
```
http://localhost:8080/Travel/TravelGroup/passengerList.dadx/WSDL
```

For HTTP SOAP, the services are invoked by sending SOAP envelopes using POST to the URL:

```
http://localhost:8080/Travel/TravelGroup/passengerList.dadx/SOAP
```

We are ready to test the Web Service:

1. Switch to server perspective and start the WebSphere test environment.
2. Switch to Web perspective, and open the browser.
3. Enter:
   `http://localhost:8080/Travel/TravelGroup/passengerList.dadx/TEST` as the URL.
4. Click **retrieve_passengerList_rdb** link, enter the flight number and click **Invoke**. See Figure 9-20.


Web Browser
http://localhost:8080/Travel/TravelGroup/passengerList.dadx/TEST
Methods
Inputs
Invoke
Clear

http://tempuri.org/TravelGroup/ Web Service

There is no documentation ava service.

store_passengerList_rdb

retrieve_passengerList_r

retrieve_passengerList_rdb Web Method

string flightNo: Air Canada 700

```
  <?xml version="1.0" ?>
- <xsd1:retrieve_passengerList_rdbResponse
    xmlns:xsd1="http://tempuri.org/Travel/TravelGroup/passengerList.dadx/XSD"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <return>
    - <xsd1:retrieve_passengerList_rdbResult
        xmlns:xsd1="http://tempuri.org/Travel/TravelGroup/passengerList.dadx/XSD
```

Figure 9-20 Testing Web Service

9.2.7 Modify passenger list application to use the Web Service

We are going to modify our passenger List application, to use the Web Service. The new approach includes an input form where the user can enter the flight Number to get the passenger List. The servlet receives this information and uses the Java proxy generated in 9.2.5 “Create the Web Service from DADX file” on page 199. The Web services returns XML data, which is transformed using XSLT Processor by the servlet, and returns and HTML response. See Figure 9-21 on page 208.

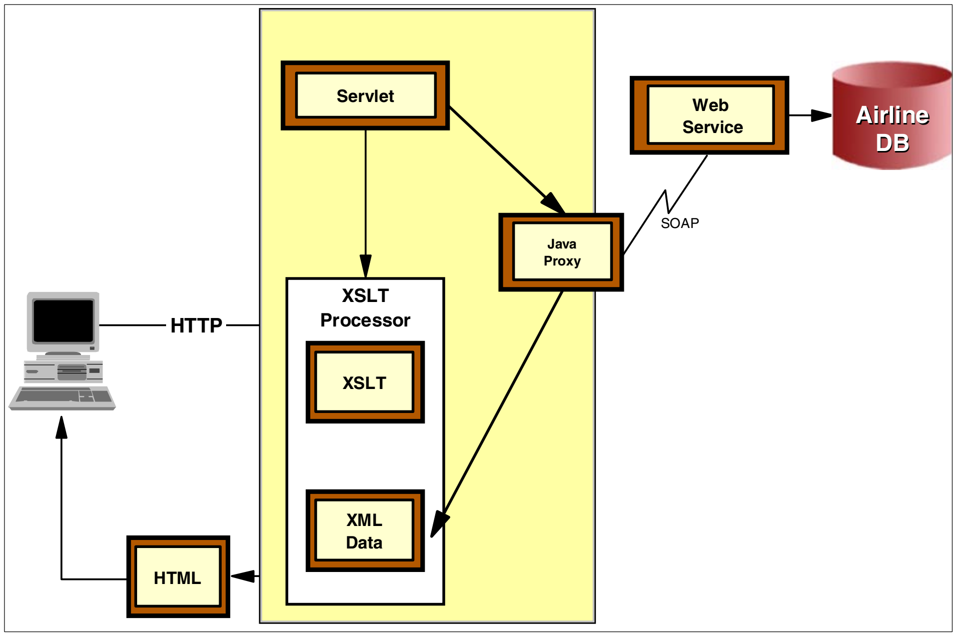


Figure 9-21 Passenger List application

Mapping DTDIDs to XSD namespaces and locations

The DTDID used by DXX is typically the local file path of the DTD. DXX retrieves the DTD either from the file system or the DTD_REF table. However, the use of a local file path is not appropriate for Web services since the result is sent to remote clients that do not have access to the server file system. The correct way to specify the document structure is by giving the namespace and location of an XML Schema (XSD) document for the result. The namespace and location are specified using an <import> element in the WSDL document that describes the Web Service.

DxxInvoker must, therefore, associate an XSD namespace and location with each DTDID that is used in the DAD documents that are referenced by the service. The mapping between DTDIDs and XML namespaces is defined by the following group.properties parameter:

- namespaceTable is a reference to an XML resource that defines the mapping between DTDIDs and XSD namespaces and locations.

The algorithm for mapping a DTDID to an XSD namespace and location is as follows:

- Lookup the DTDID in the namespaceTable. If an entry exists then use the defined namespace and location.
- Otherwise, the DTDID cannot be mapped so throw an exception.

Modifying the XSL file

We need to modify the namespace in *passengerList.xsl* since the Web Service has added some lines that were not specified on the DTD file.

- Add namespaces specified in the Web Service returned format shown Example 9-8.

Example 9-8 Namespaces

```
xmlns:xsd1="http://tempuri.org/Travel/TravelGroup/passengerList.dadx/XSD"
xmlns:pl="http://www.airline.com/passengerList.dtd"
```

The returned root element from the Web Service is different, since it uses the name of the operation. Change XSL file as shown in Example 9-9.

Example 9-9 Changing root element

```
<!--=====================================================================-->
  <!--                          The Root Element                          -->
  <!-- The "Root Element" section specifies which template will be          -->
  <!-- invoked first thus determining the root element of the result tree.  -->
  <!--=====================================================================-->
<xsl:template match="xsd1:retrieve_passengerList_rdbResult">
    <xsl:call-template name="html"/>
  </xsl:template>
```

Since we introduce a variable named *pl* in the namespace, we have to use it in every occurrence of an element like *Flight, Customer,* and so on. See completed xsl file in Example 9-10.

Example 9-10 passengerList.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsd1="http://tempuri.org/Travel/TravelGroup/passengerList.dadx/XSD"
    xmlns:pl="http://www.airline.com/passengerList.dtd"
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt">
<xsl:output method="xml" encoding="UTF-8" indent="yes"
xalan:indent-amount="2"/>
<xsl:strip-space elements="*"/>


  <!--=========================================================================-->
  <!-- This file contains an XSLT transformation stylesheet which           -->
  <!-- constructs a result tree from a number of XML sources by filtering   -->
  <!-- reordering and adding arbitrary structure. This file is             -->
  <!-- automatically generated by the XML Mapper tool from IBM WebSphere    -->
  <!-- Studio Workbench.                                                    -->
  <!--=========================================================================-->



  <!--=========================================================================-->
  <!--                          The Root Element                            -->
  <!-- The "Root Element" section specifies which template will be         -->
  <!-- invoked first thus determining the root element of the result tree.  -->
  <!--=========================================================================-->

  <xsl:template match="xsd1:retrieve_passengerList_rdbResult">
    <xsl:call-template name="html"/>
  </xsl:template>


  <!--=========================================================================-->
  <!--                         Remaining Templates                          -->
  <!-- The remaining section defines the template rules. The last template  -->
  <!-- rule is a generic identity transformation used for moving complete   -->
  <!-- tree fragments from an input source to the result tree.             -->
  <!--=========================================================================-->

  <!-- Newly-defined element template -->
  <xsl:template name="html">
    <html>
      <xsl:call-template name="body"/>
    </html>
  </xsl:template>

  <!-- Newly-defined element template -->
  <xsl:template name="body">
    <body>
      <xsl:attribute name="bgcolor">
        <xsl:value-of select="'#FFFFCC'"/>
      </xsl:attribute>
      <xsl:call-template name="h2"/>
      <hr></hr>
      <xsl:call-template name="table"/>
    </body>
  </xsl:template>

  <!-- Newly-defined element template -->
  <xsl:template name="h2">
    <h2>
      <CENTER>
        <xsl:value-of select="pl:Flight/@flightNo"/>
      </CENTER>
    </h2>
  </xsl:template>

  <!-- Composed element template -->
  <xsl:template match="pl:Customer">
    <tr>
      <td>
        <xsl:value-of select="pl:name/text()"/>
      </td>
      <td>
        <xsl:value-of select="pl:membership/text()"/>
      </td>
    </tr>
  </xsl:template>

  <!-- Newly-defined element template -->
  <xsl:template name="table">
    <table>
      <xsl:apply-templates select="pl:Flight/pl:Customer"/>
    </table>
  </xsl:template>

  <!-- Identity transformation template -->
  <xsl:template match="*|@*|comment()|processing-instruction()|text()">
    <xsl:copy>
      <xsl:apply-templates
select="*|@*|comment()|processing-instruction()|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Creating HTML input form

In order to let the user enter a search key, we are going to design a *HTML form* and a *servlet.* This servlet is going to call a *xs*l processor using *jaxp.* For an example of an input form, see Figure 9-22.

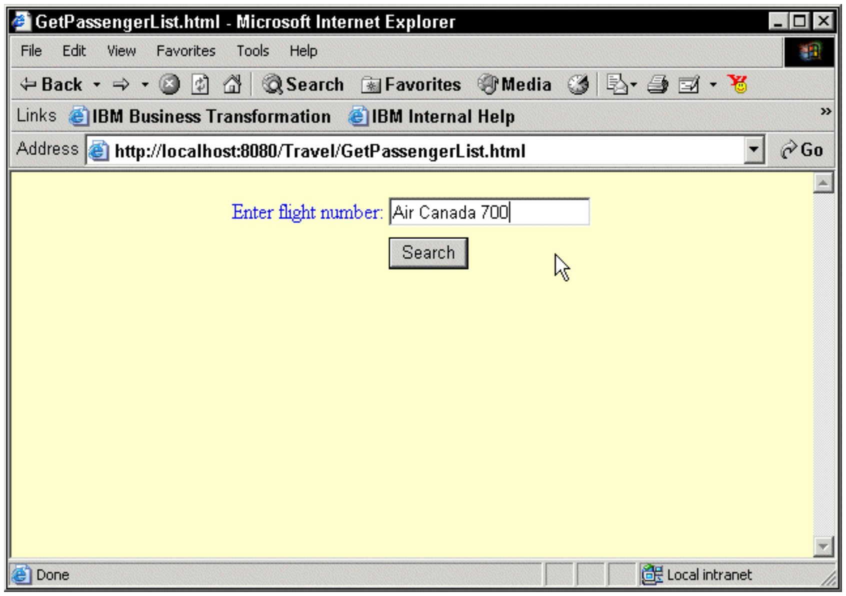

GetPassengerList.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Business Transformation IBM Internal Help
Address http://localhost:8080/Travel/GetPassengerList.html
Go
Enter flight number: Air Canada 700
Search
Done
Local intranet

Figure 9-22 Input form GetPassengerList.html

This input form sends a flight number entered by the end user to the servlet. The servlet calls the Web Service using its Java proxy, and transfer the returned XML format into an HTML page and call it, See Figure 9-23 on page 213. The Java proxy was generated by the Web Service wizard, and the source code of this proxy is provided. We only need to know which method to call.

Using Java proxy

The Java proxy that is generated by Web services wizard enables to access to the Web services by method call instead of standard Web Service invocations. To use PassengerListProxy, instantiate the proxy and call the retrieve_passengerList_rdb method which is we defined in the Web services wizard. The result can be a source for the Transformer class:

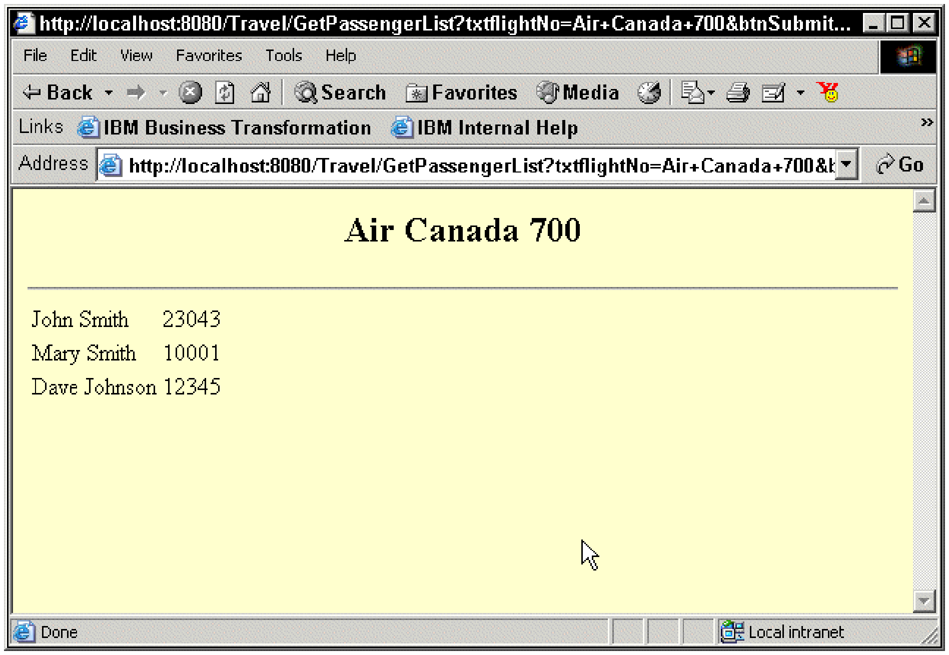
```
passengerListProxy proxy = new passengerListProxy();
proxy.retrieve_passengerList_rdb_(key);
```

JAXP DOM I/O

To make the result as an input for the transformer, creating org.w3c.dom.Document format is the best way to do. To create the document from the result of the Web Service, use following statement:

```
DOMSource domSource = new DOMSource(proxy.retrieve_passengerList_rdb_(key));
transformer.transform(domSource, new StreamResult(pw));
```

Even the DOM transformation is fast, the DOM is memory intensive, the DOM data which is generated dynamically as the result of Web Service (or database query) will be better than generating from the DOM from a stream inside of the transformer.


http://localhost:8080/Travel/GetPassengerList?txtflightNo=Air+Canada+700&btnSubmit...
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Business Transformation IBM Internal Help
Address http://localhost:8080/Travel/GetPassengerList?txtflightNo=Air+Canada+700&
Go
Done
Local intranet

Air Canada 700

John Smith 23043
Mary Smith 10001
Dave Johnson 12345

Figure 9-23 passengerList result page

Conclusion

XML Web services provide a powerful new technology for integrating heterogeneous applications over the Internet. WebSphere Application Server provides a fully supported production-ready deployment environment for Web services base on the Apache SOAP run-time environment. XML plays a central role by providing a data interchange format that is independent of programming languages, operating systems, and hardware.

WebSphere Studio Application Development is a new development environment that supports the fully life cycle for Web services development with support for SOAP, WSDL, and UDDI, and includes a powerful suite of XML tools. Using this environment, developers can easily transform existing components, such as JavaBeans, EJB beans, and SQL statements, into Web services, and can incorporate Web services into new applications.

10

Development of XML-based Enterprise applications

This chapter provides the reader with an opportunity to learn about the development of XML based Enterprise applications. The chapter describes in detail the process of developing such type of applications using Websphere Studio Application Developer. This information is provided through building a sample customer registration scenario.

In this chapter, the following topics are described:

- Architecture of XML-based Enterprise applications
- Development of such applications using Application Developer
- Deployment and testing of the solution on the Application Developer

10.1 XML based Enterprise application architecture

Most of the contemporary Enterprise Edition (J2EE) applications rely on HTML for the presentation layer. This minimizes the benefits provided by the flexibility of the distributed multi tier J2EE, since the target audience is limited to users of Web browsers. As the pervasive world is widely evolving, future audience will be the users of new devices. Accordingly, we need to decouple content from presentation logic and become independent of the audiences' devices. XML's Web publishing capabilities are the key solution for providing that level of isolation. XML's Web publishing capabilities are available using XSLT to transform XML documents into other textual documents, compliant with the desirable target devices. The focus of this chapter will only be on using XSLT to transform XML to HTML, but you can use XSLT to transform XML to any desirable markup language, for example WML, VoxML,.etc.

The J2EE layered architecture, illustrated in Figure 10-1, can minimize the cost of evolving an application's functionality and updating its implementation technology. These layers can be broadly categorized as the client layer, the Web tier layer (which includes the controller servlet, validation logic, presentation data, and presentation logic), the app server layer, and the data server layer.

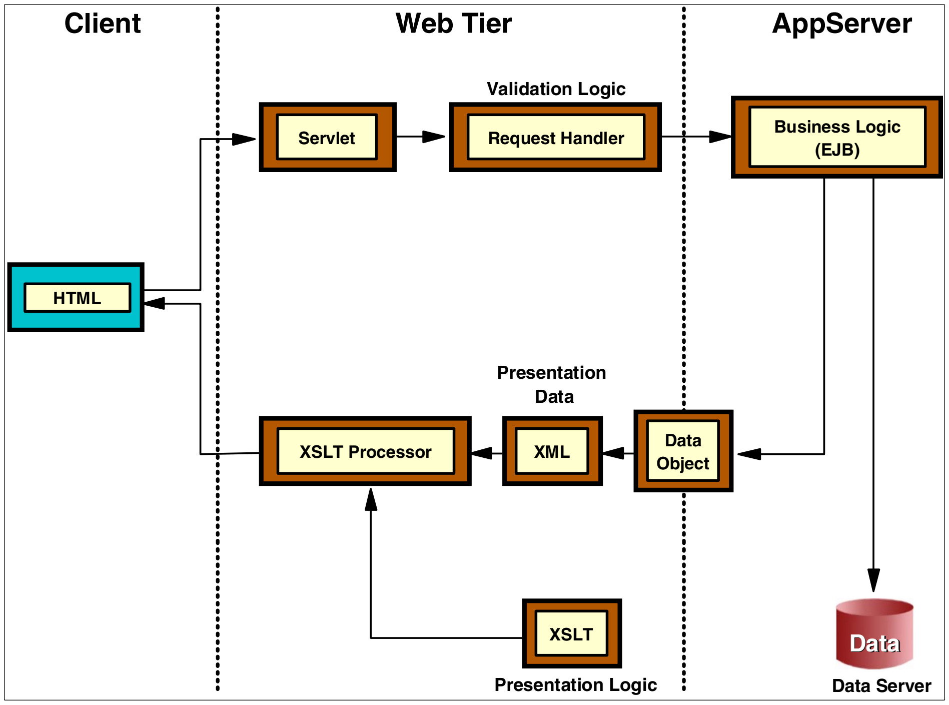


Figure 10-1 Multi-tier solution architecture

In order to understand how these components interact with each other, have a deep look at 10.2, “Solution outline for customer registration sample” on page 217.

10.2 Solution outline for customer registration sample

An overview of the sample application can be found in Chapter 8. This section discusses the interaction between solution components, before we progress to building the application. There are two possible ways of interaction between the components of the system. This difference in interaction is based on the target functionality. We focus on the set of interactions for storing data, then we discuss the set of interactions for retrieving customer information.

Customer registration

When registering, the user enters the necessary registration data in the form provided by the HTML in the client layer. When the user clicks the Submit button, a series of interactions between system components take place, as indicated in Figure 10-2 on page 218, and in the following steps:

1. The HTML sends the customer registration data through the HTTP request to the servlet, invoking the servlet doPost method.
2. The servlet extracts the data from the request, and creates a new CustomerXML object. Now the servlet invokes the create method in the CustomerXML object.
3. The CustomerXML object’s create method, instantiates a new CustomerFactory, and invokes its create method having the customer registration data.
4. The factory creates a customer entity bean with the customer’s registration information.
5. The entity bean’s data is saved into the customer table in the database.
6. The customer entity bean is returned to the CustomerXML.
7. The CustomerXML executes its produceDOMDocument method, which converts the Customer entity bean into XML data.
8. The XSLServlet applies the Customer Result XSL on the generated XML data, by invoking the XSLT Processor.
9. The XSLT Processor applies the stylesheet on the XML data, generating an HTML representation for the customer registration data.

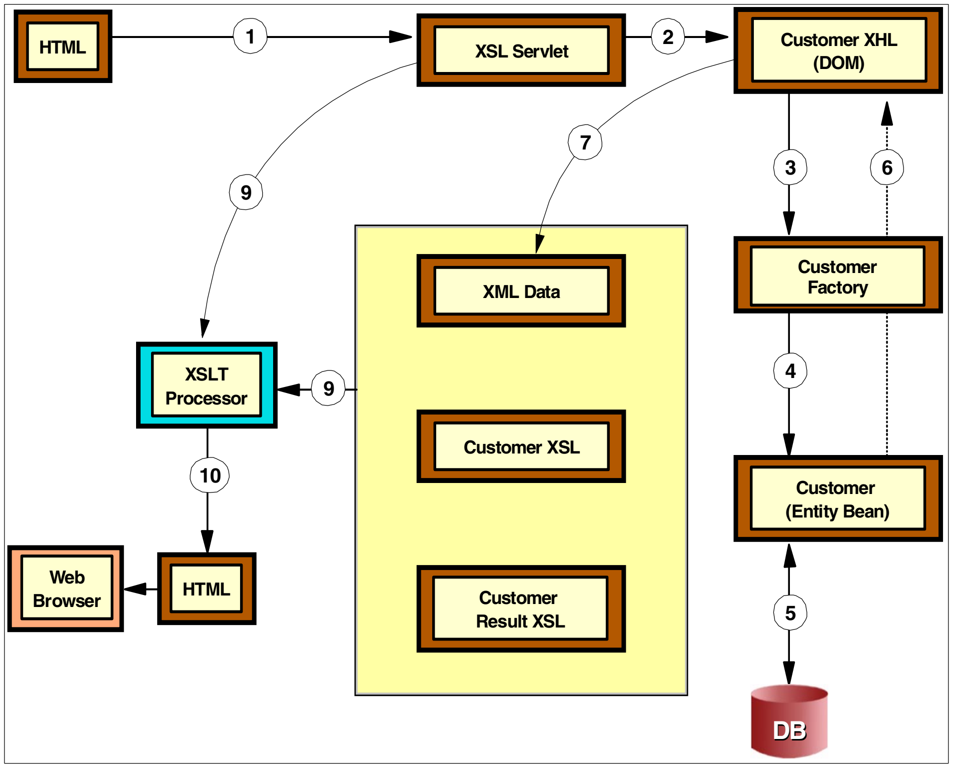


Figure 10-2 Customer registration scenario outline

Retrieving customer information

When retrieving customer data, the user enters the membership number of the customer whose information is to be retrieved in the form provided by the HTML in the client layer. When the user clicks the Submit button, a series of interactions between system components take place, as indicated in Figure 10-3 on page 219, and in the following steps:

1. The HTML sends that value to the servlet through the HTTP request to the servlet, invoking the servlet doGet method.
2. The servlet validates that the user has entered a membership value. Then the servlet invokes the findByPrimaryKey method that belongs to the CustomerFactory class, using the input membership value.
3. The CustomerFactory retrieves the data from the database, and constructs a customer entity bean setting its attributes with the values returned from the database.
4. The servlet provides the returned entity bean to the CustomerXML, which converts that bean into a DOM representation.

5. The servlet invokes the XML to HTML transformation using the XSLT processor given the XSL for the target HTML. When the transformation is performed, the user can view the retrieved customer information on the Web browser.

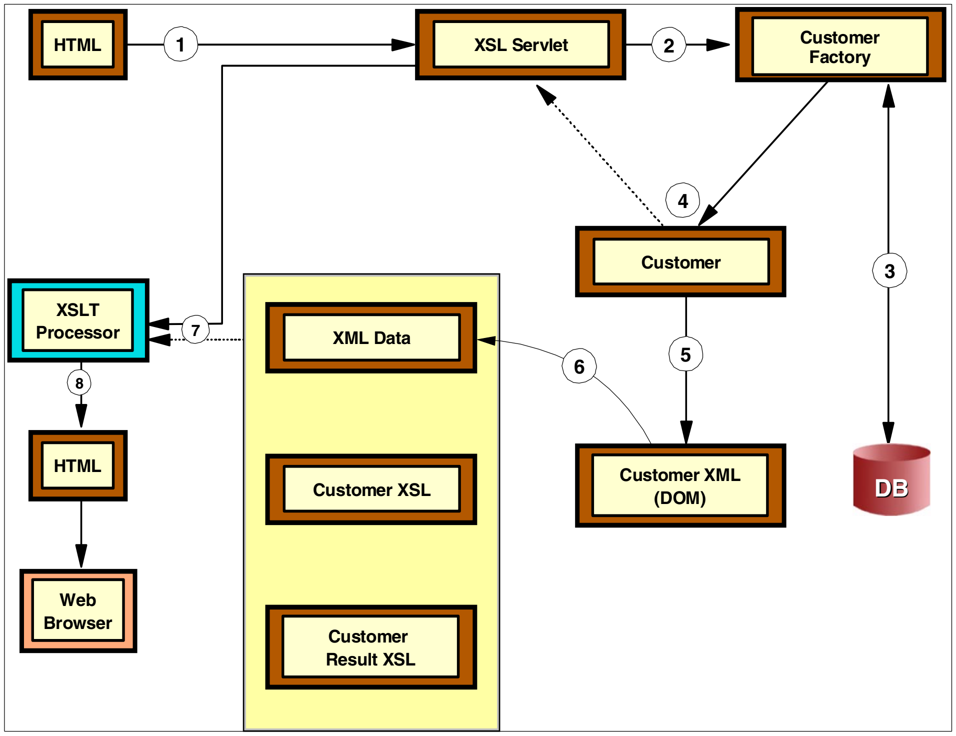


Figure 10-3 Retrieving customer data scenario outline

10.3 Developing the customer registration sample

In order to develop this J2EE application, we divide the process into four phases:

- Creating the Web tier
- Development of the back-end layer, which includes the business logic (EJB) and the database schema
- Integrating the Web tier with the back-end layer
- Development of the client layer

10.3.1 Creating the Web tier

In order to create the Web tier, the Websphere Application Developer provides a wizard for doing that. Given a JavaBean, the wizard generates the Web tier components as shown in Figure 10-4.

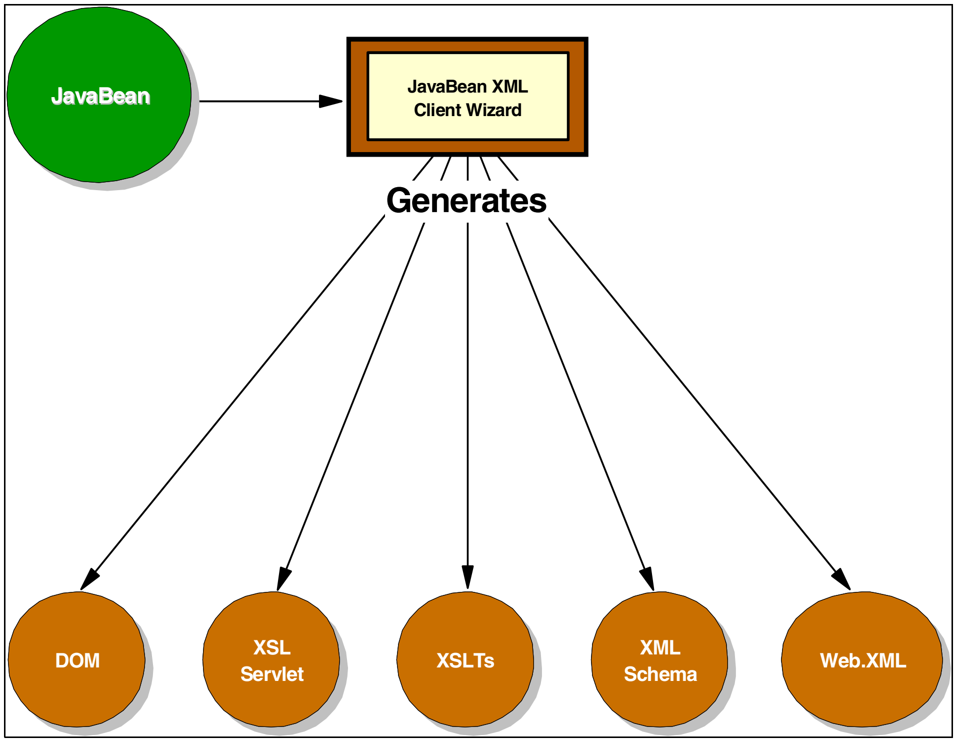


Figure 10-4 JavaBean to XML client wizard

Preparing to create the Web tier

Websphere Application Developer provides a wizard that facilitates the creation of the Web Tier. But, before we start using that wizard, we must perform some configuration steps.

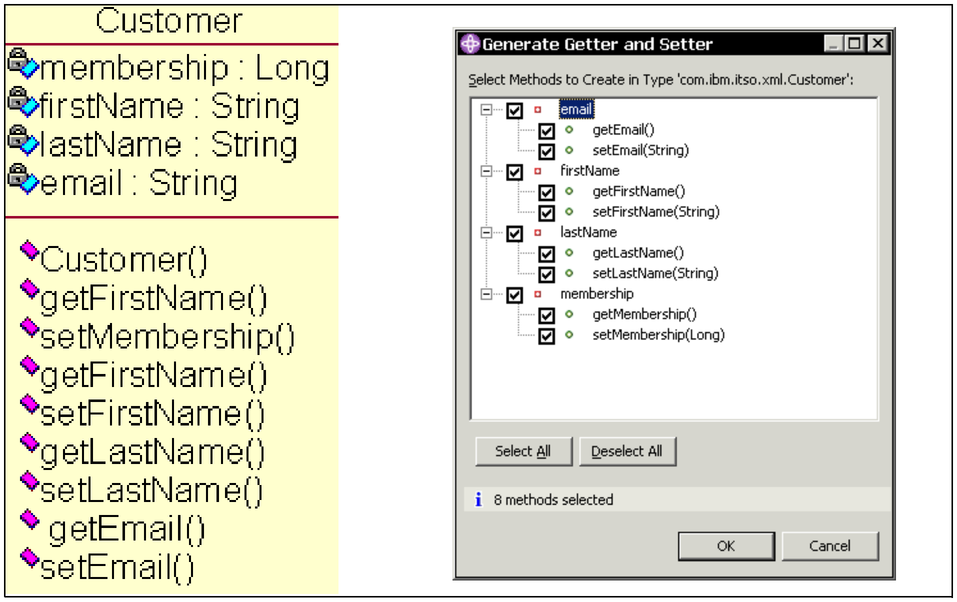
Project configuration

Create a J2EE Enterprise Application project called `CustomerInfo` as J2EE 1.2 Compliant. The following projects are generated automatically unless you change the names:

CustomerInfo	EAR Project
CustomerInfoClient	Test Client
CustomerInfoEJB	EJB Project
CustomerInfoWeb	Web Project

JavaBean creation

Create a new JavaBean called `Customer` into the CustomerInfoWeb project, by choosing the Java perspective, and select **File—>New—>Class,** edit the class name to be `Customer`, and select **Finish**. Edit the Customer class, creating the attributes, and generate setter and getter methods according to the class specification in Figure 10-5.



Select Methods to Create in Type 'com.ibm.itso.xml.Customer':

```
[✓] email
    [✓] getEmail()
    [✓] setEmail(String)
[✓] firstName
    [✓] getFirstName()
    [✓] setFirstName(String)
[✓] lastName
    [✓] getLastName()
    [✓] setLastName(String)
[✓] membership
    [✓] getMembership()
    [✓] setMembership(Long)
```

8 methods selected

Figure 10-5 Customer JavaBean class specification

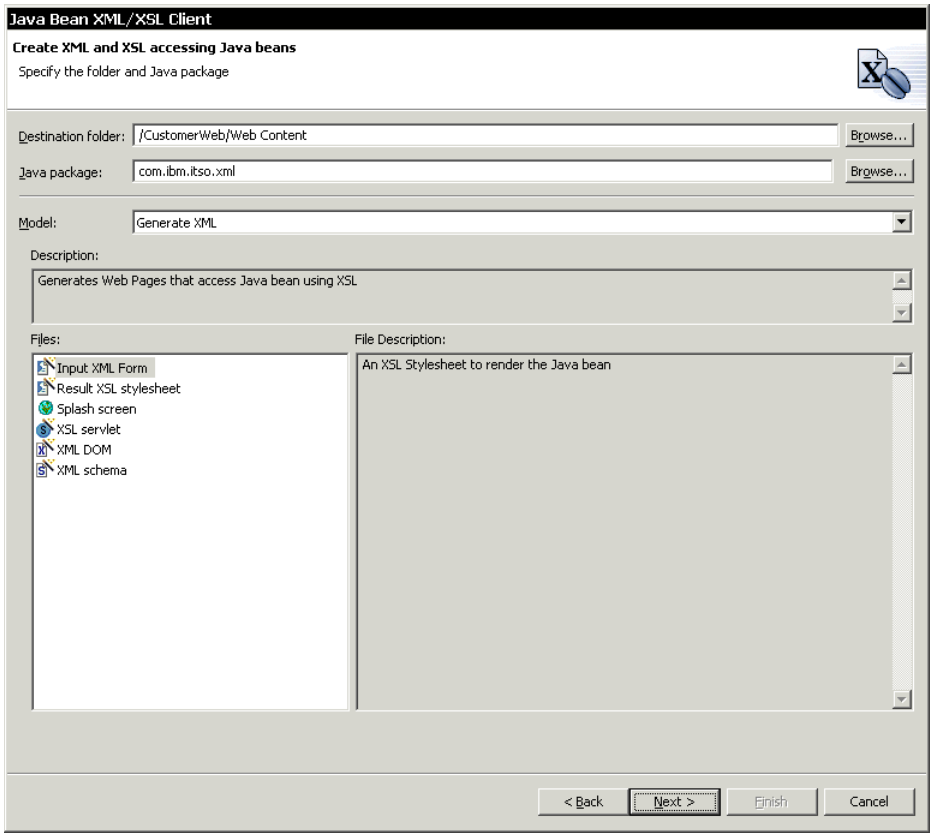
Note: Application Developer V5.0 supports both J2EE 1.2 and 1.3. In this chapter, we are using J2EE 1.2 and EJB 1.1 to run on WebSphere Application Server V4. If you choose J2EE 1.3, you need to use EJB 2.0 and WebSphere Application Server Test Environment V5.

Creating the Web tier

To create the Web tier:

1. Close the Java perspective as we no longer require it, and switch to the XML perspective.
2. At the top menu of your workspace, click **File—>New—>Java Bean XML/XSL Client** to invoke the JavaBean to XML wizard.

   The wizard opens Figure 10-6, indicating the creation of XML and XSL accessing JavaBeans.


Java Bean XML/XSL Client

Create XML and XSL accessing Java beans

Specify the folder and Java package

Destination folder: /CustomerWeb/Web Content

Java package: com.ibm.itso.xml

Model: Generate XML

Description:

Generates Web Pages that access Java bean using XSL

Files:

- Input XML Form
- Result XSL stylesheet
- Splash screen
- XSL servlet
- XML DOM
- XML schema

File Description:

An XSL Stylesheet to render the Java bean

Figure 10-6 JavaBean XML/XSL client wizard

3. Leave all defaults as they are, click **Next**.

   Now we need to choose the JavaBean, the methods, and the properties that we intend to include in our application:

4. Select **Browse** to get a list of the available classes in your workspace. Choose your customer JavaBean, as shown in Figure 10-7.

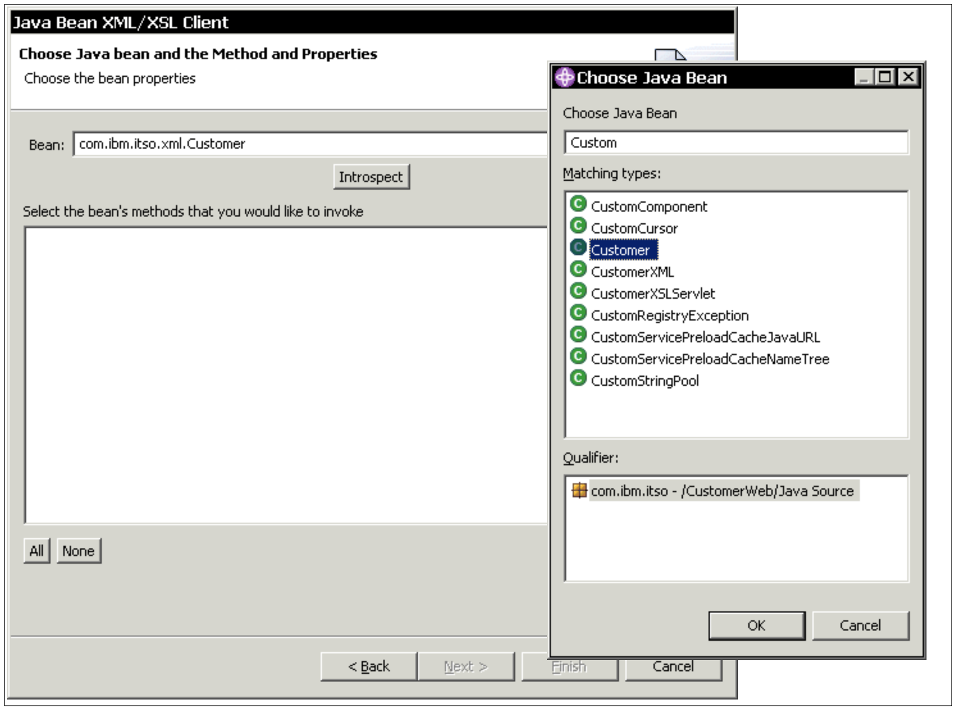
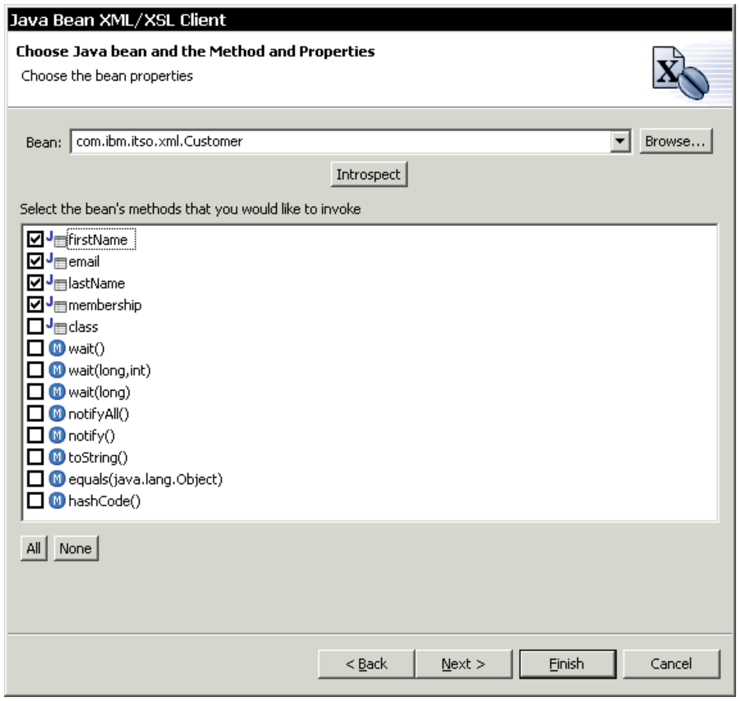

Java Bean XML/XSL Client
Choose Java bean and the Method and Properties
Choose the bean properties
Bean: com.ibm.itso.xml.Customer
Introspect
Select the bean's methods that you would like to invoke
All
None
< Back
Next >
Finish
Cancel
Choose Java Bean
Choose Java Bean
Custom
Matching types:
CustomComponent
CustomCursor
Customer
CustomerXML
CustomerXSLServlet
CustomRegistryException
CustomServicePreloadCacheJavaURL
CustomServicePreloadCacheNameTree
CustomStringPool
Qualifier:
com.ibm.itso - /CustomerWeb/Java Source
OK
Cancel

Figure 10-7 JavaBean XML/XSL client: JavaBean selection

5. After you choose your bean, click **OK** to go to the dialog in Figure 10-8, and click **Introspect** to introspect the bean if you did not get any methods, where you will select the bean properties that you want to include. In this scenario, we choose membership, firstName, lastName, and e-mail.
6. Click **Next**.


Java Bean XML/XSL Client

Choose Java bean and the Method and Properties

Choose the bean properties

Bean: com.ibm.itso.xml.Customer

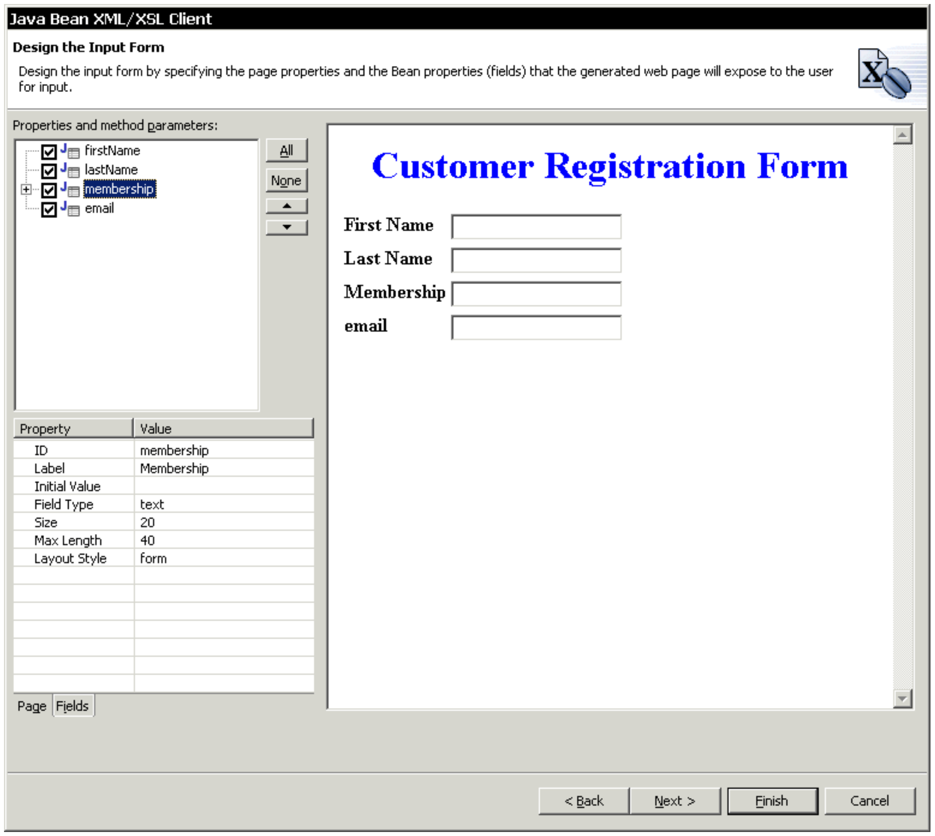
Select the bean's methods that you would like to invoke

- [x] firstName
- [x] email
- [x] lastName
- [x] membership
- [ ] class
- [ ] wait()
- [ ] wait(long,int)
- [ ] wait(long)
- [ ] notifyAll()
- [ ] notify()
- [ ] toString()
- [ ] equals(java.lang.Object)
- [ ] hashCode()

Figure 10-8 JavaBean XML/XSL client: methods and properties selection

Next we design the input form that the wizard will generate, specifying the properties of the page, and the bean fields that the generated Web page will provide to the user for input:

1. Select the input fields that you want, checking the box next to each field as seen in Figure 10-9. Modify also the title of the page, and the field labels if you desire.
2. Click **Next** to continue.


Java Bean XML/XSL Client

Design the Input Form

Design the input form by specifying the page properties and the Bean properties (fields) that the generated web page will expose to the user for input.

Properties and method parameters:

- ☑ firstName
- ☑ lastName
- ☑ membership
- ☑ email

Property	Value
ID	membership
Label	Membership
Initial Value	
Field Type	text
Size	20
Max Length	40
Layout Style	form

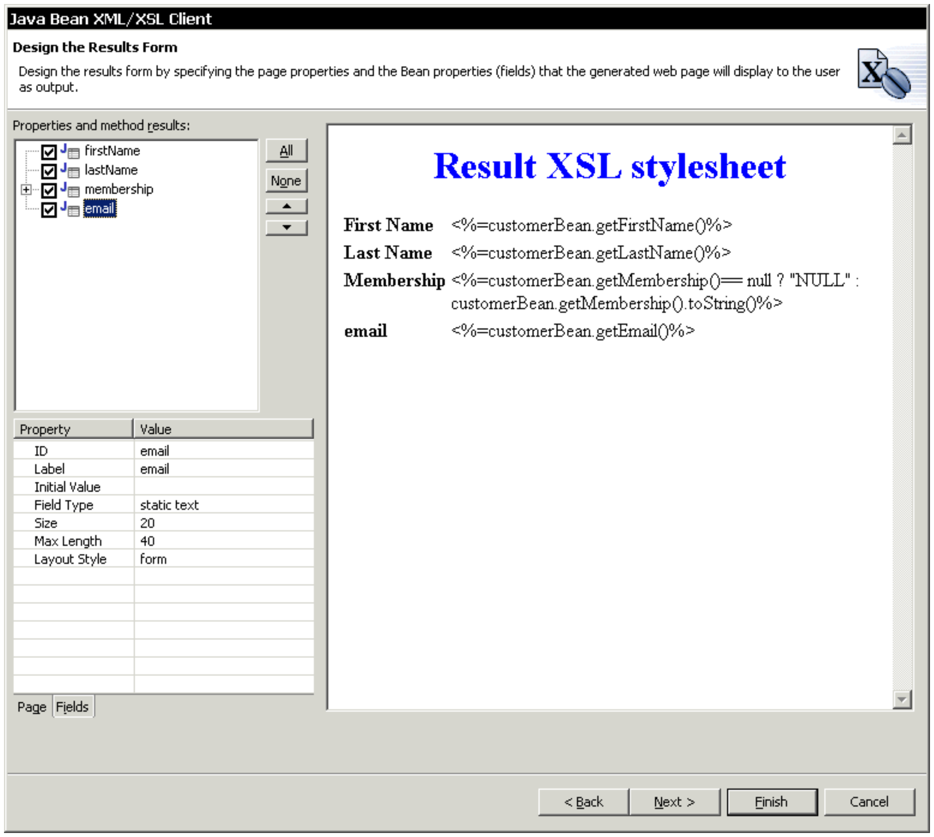
Customer Registration Form

First Name	
Last Name	
Membership	
email	

Figure 10-9 JavaBean XML/XSL client: input form design

Next we design the result form, which shows the customer registration information, that the user has entered during the registration process:

1. Design the form by specifying the page properties, like the page heading for example. Also specify the bean fields that you want to include in the generated Web page, as shown in Figure 10-10.
2. Click **Next**.


Java Bean XML/XSL Client

Design the Results Form

Design the results form by specifying the page properties and the Bean properties (fields) that the generated web page will display to the user as output.

Properties and method results:

- ☑ firstName
- ☑ lastName
- ☑ membership
- ☑ email

Property	Value
ID	email
Label	email
Initial Value	
Field Type	static text
Size	20
Max Length	40
Layout Style	form

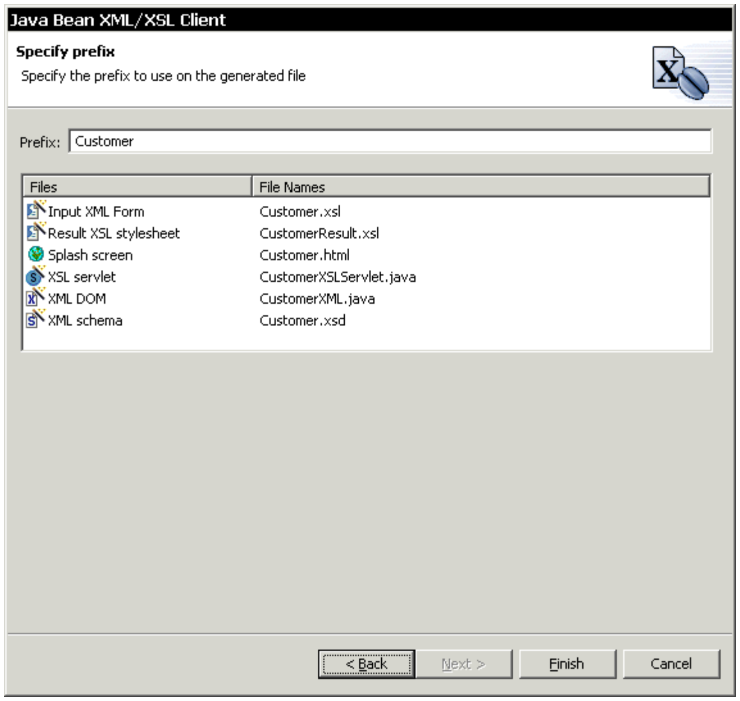
Result XSL stylesheet

First Name	<%=customerBean.getFirstName()%>
Last Name	<%=customerBean.getLastName()%>
Membership	<%=customerBean.getMembership()== null ? "NULL" : customerBean.getMembership().toString()%>
email	<%=customerBean.getEmail()%>

Figure 10-10 JavaBean XML/XSL client: results form design

After finishing the design of the results form, we need to specify a prefix for all generated files:

1. Enter the prefix for the generated files as `Customer`, as indicated in Figure 10-11.
2. Click **Finish**.


Java Bean XML/XSL Client

Specify prefix

Specify the prefix to use on the generated file

Prefix: Customer

Files	File Names
Input XML Form	Customer.xsl
Result XSL stylesheet	CustomerResult.xsl
Splash screen	Customer.html
XSL servlet	CustomerXSLServlet.java
XML DOM	CustomerXML.java
XML schema	Customer.xsd

Figure 10-11 JavaBean XML/XSL client: prefix specification

Investigating the Web-tier generated files

It is worthwhile having a look at the files generated by the wizard to get a better understanding of the Web tier structure. By moving to the Navigator window, expand your Registration project, as shown in Figure 10-12 on page 228. You may get a broken link warning on the web.xml. Rebuilding the project will resolve this warning.

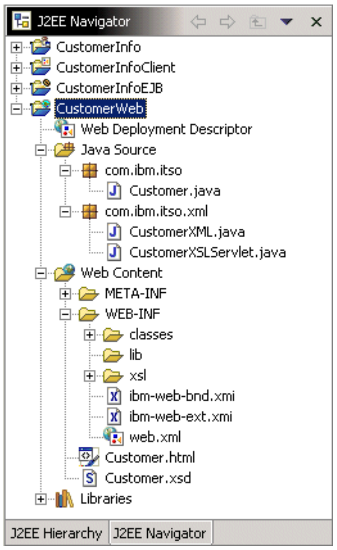


Figure 10-12 Registration project structure after the Web tier creation

Java source files: By expanding the Java Source folder, under the CustomerInfoWeb project, you will find two new Java files, in addition to our original customer bean Java file. The first new file is CutomerXML.java, which is responsible for converting a JavaBean into a DOM representation. The second new file is CustomerXSLServlet.java, which is the controller of the whole process as it is responsible for invoking the required business logic, and applying the XSL to the generated DOM.

Following is a brief explanation of the contents of these two Java classes:

- **CustomerXML** is responsible for converting a JavaBean into a DOM representation. Studying the code for this class, we notice the following:
  - It has an attribute referencing our customer JavaBean. This attribute represents the bean to be transformed to the DOM representation.
  - It has a set of get and set methods to handle the values of the Customer bean attributes. For example, the setMembership method of the CustomerXML, sets the value for the membership attribute of the Customer bean. This applies to all customer bean attributes. These are useful of storing and retrieving the bean data values.
  - The method responsible for the transformation is produceDOMDocument. As shown in Figure 10-13, given the JavaBean as an input to the produceDomdocument method, the method uses Sun's jaxp (Java API for XML processing) to produce the XML data from the JavaBean. The source code for that method is available in Example 10-1. It creates a

DocumentBuilder to use for the document generation. It creates a customer element representing the document root. It adds that element to the created document, and starts creating elements for the customer's attributes, adding them to the document. It retrieves that values for these attributes from the customer JavaBean, using the bean's Get methods. After the document construction with the data, the method returns the document.

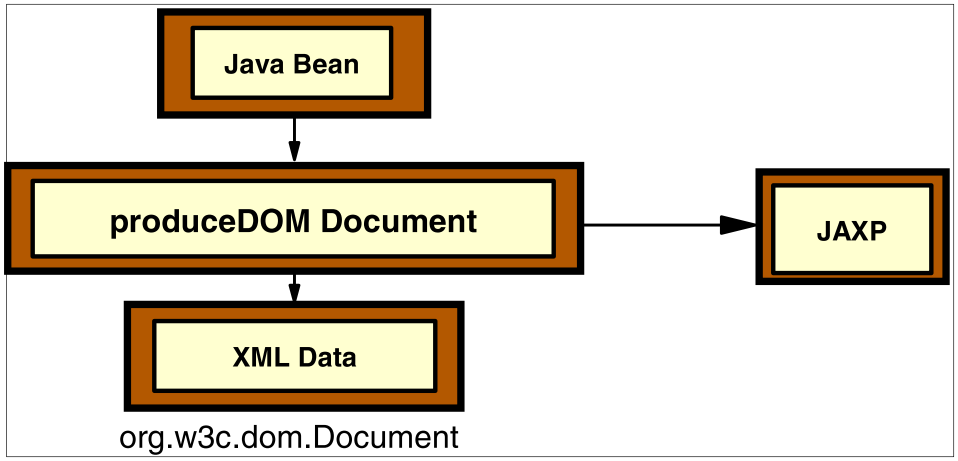


Figure 10-13 Produce DOM Document from JavaBean

Example 10-1 Source code for CustomerXML produceDOMDocument

```
public Document produceDOMDocument()
   throws ParserConfigurationException
   {// use Sun's JAXP to create the DOM Document
     DocumentBuilderFactory dbf =
       DocumentBuilderFactory.newInstance();
     DocumentBuilder docBuilder = dbf.newDocumentBuilder();
     Document doc =  docBuilder.newDocument();
     Element parent = null;
     // create the root of the document
     Element rootElement = doc.createElement("Customer");
     doc.appendChild(rootElement);
   addElement(doc, rootElement, "membership",
                 convertToString(getBean().getMembership()));
     addElement(doc, rootElement, "firstName",
                 convertToString(getBean().getFirstName()));
     addElement(doc, rootElement, "lastName",
                 convertToString(getBean().getLastName()));
     addElement(doc, rootElement, "email",
                 convertToString(getBean().getEmail()));
     return doc;
}
```

- **CustomerXSLServlet:** This is responsible for applying the XSL to the DOM generated from the CustomerXML. This process transforms and formats the DOM information into a rendered result. This is called styling. For the styling to be possible, two components come together: XSL Transormations (XSLT), which allows for a reorganization of information; and XSL, which specifies the formatting of the information for rendering. A XSL processor takes a stylesheet consisting of a set of XSL commands, and transforms an input XML document. Studying the code for this servlet, we noticed the following:
  - The init method does the necessary configuration and initialization for the sevlet's operation. For the servlet to handle the process of transforming the XML data to the desirable format, based on an XSL stylesheet, the servlet needs to do some configuration steps before invoking the XSLT processor. The init method first creates a new instance of the TransformerFactory class.
  - Having the stylesheet XSL files present in the file system, the Transformer factory instance needs to reference them. So the servlet creates the template's objects using the XSL files. A template's object is a compiled representation of a XSL file. Compilation of XSL is provided by JAXP. Invoking the newTemplates method of the TransformerFactory, and passing to it an XSL file, it processes the file, and generates a compiled representation of that XSL file in the form of the template's object.
  - This template's object may then be used concurrently across multiple threads. Creating a templates object allows the TransformerFactory to do detailed performance optimization of transformation instructions, without penalizing runtime transformation. As shown in Example 10-2, the init method retrieves the files using the path specifies in the begining of the method, and creates the template's objects using the transformer factory. The XSLT processor should used these compiled instances of the style sheets to transform a XML document. Exceptions are thrown if the servlet cannot locate the XSL files in the specified location, or if the files contain errors.

Example 10-2 Source code for CustomerXSLServlet init method

```
public void init(ServletConfig config)
      throws ServletException
  { super.init(config);
   TransformerFactory transFact =
         TransformerFactory.newInstance();
    String xslName = "/WEB-INF/xsl/Customer.xsl";
    String xslResult = "/WEB-INF/xsl/CustomerResult.xsl";
    try
    { URL stylesheetURL =
      getServletContext().getResource(xslName);
     String xsltSystemID = stylesheetURL.toExternalForm();
     mainStylesheet = transFact.newTemplates(
         new StreamSource(xsltSystemID));
  stylesheetURL = getServletContext().
         getResource(xslResult);
     xsltSystemID = stylesheetURL.toExternalForm();
     resultStylesheet = transFact.newTemplates(
         new StreamSource(xsltSystemID));
   } catch (TransformerConfigurationException tce)
   { log("Unable to compile stylesheet", tce);
     throw new UnavailableException(
         "Unable to compile stylesheet");
   } catch (MalformedURLException mue)
   { log("Unable to locate XSLT file: " + xslName);
     throw new UnavailableException(
         "Unable to locate XSLT file: " + xslName);
   }
 }
```

– The showPage method applies a style sheet to a DOM tree. As shown in Figure 10-14 on page 232, and in the source code in Figure 10-3 on page 232, the method first invokes the DOM generation using the produceDOMDocument in the CustomerXML class, mentioned in Example 10-2 on page 230. Following the DOM generation, a new transformer is created. This transformer applies the style sheet on the generated DOM, and writes the transformed HTML result to the Web browser.

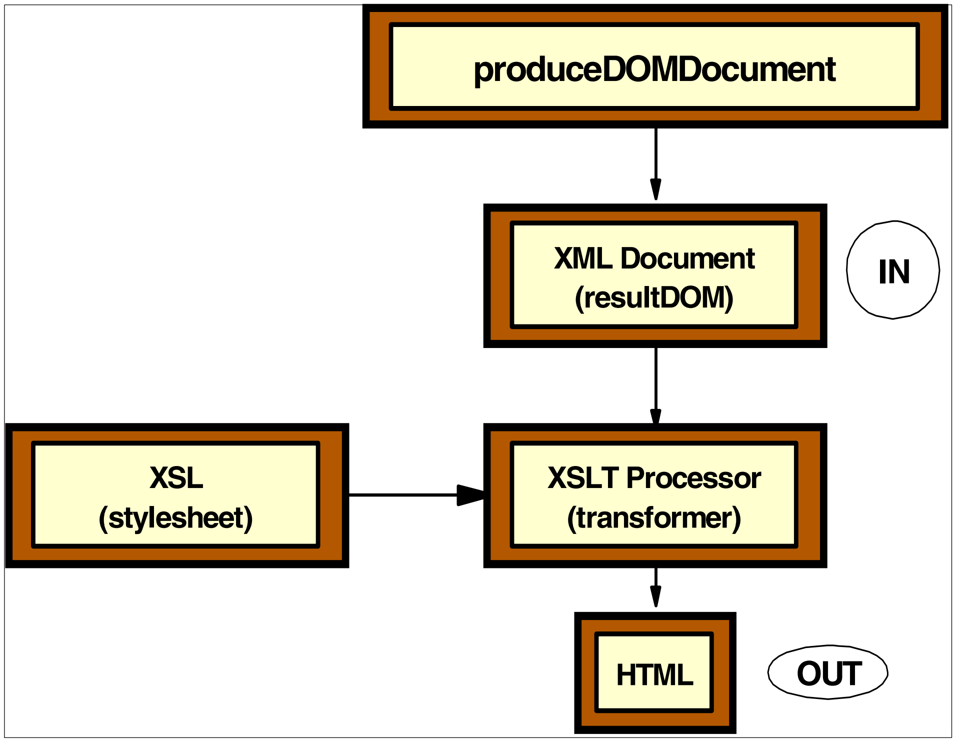


Figure 10-14 ShowPage method responsible for XSL transformation

Example 10-3 Source code for CustomerXSLServlet showPage method

```
private void showPage(Templates stylesheet, HttpServletResponse response)throws
IOException{
    try {
      org.w3c.dom.Document resultDOM = getCustomerXML().produceDOMDocument();
      Transformer transformer = stylesheet.newTransformer();
      response.setContentType("text/html");
      PrintWriter writer = response.getWriter();
      transformer.transform(new DOMSource(resultDOM),
                new StreamResult(writer));
    } catch (Exception ex){
      PrintWriter pw = response.getWriter();
      pw.println("<html><body>
          <h2>Transformation Error</h2><pre>");
      ex.printStackTrace(pw);
      pw.println("</pre></body></html>");
    }
  }
```

– The HTTP Post method of the servlet references the CustomerXML class, as shown in Example 10-4. It sets all the attribute values, after extracting them from the request. After all values are set in the customer bean present in CustomerXML, showPage is invoked.

Example 10-4 Source code for the servlet doPost method

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException
  {
    CustomerXML xml = getCustomerXML();
    xml.setMembership(new java.lang.Long(
      request.getParameter("membership")));
    xml.setFirstname(new java.lang.String(
      request.getParameter("firstName")));
    xml.setLastname(new java.lang.String(
      request.getParameter("lastName")));
    xml.setEmail(new java.lang.String(
      request.getParameter("email")));
    showPage(resultStylesheet, response);
  }
```

XSL Files: By expanding the folder /Web Content/WEB-INF/xsl, under the Registration project, you will find two XSL files. The Cutstomer.xsl file represents the xsl for the input customer data, while the CustomerResult.xsl file represents the xsl for the data resulting from the registration process:

- **Customer.xsl** is the style sheet for the customer registration form. Note the following:
  – The stylesheet code in Example 10-5 represents the style and the structure of the registration form. It is clear that it is stylesheet for an HTML form.

Example 10-5 Part of customer stylesheet specifying the overall form structure

```
<xsl:template match="/">
    <html>
      <head>
          <title>Customer Registration Form</title>
          <style type="text/css">
               <![CDATA[
               body
               {
                 background-color: #f8f7cd;
               }
               h1
               {
                 color: #0000ff;
            text-align: center;
              }
               ]]>
          </style>
      </head>
      <body>
        <xsl:apply-templates select="Customer"/>
      </body>
    </html>
  </xsl:template>
```

– The style for the body of this form is clear in stylesheet part present in Example 10-6. It shows that there is a heading for the body, followed by an html form. The data is structured in table inside the form. And finally the form has a submit button. According to the form's specification, pressing the submit button, redirects the user to the doPost method of the CustomerXSLSevlet.

Example 10-6 Part of registration form stylesheet focusing on form structure

```
<xsl:template match="Customer">
    <h1>Customer Registration Form</h1>
   <form action="/Registration/CustomerXSLServlet"
      method="post">
      <table border="0" cellpadding="2" cellspacing="0">
        <xsl:apply-templates select="membership"/>
        <xsl:apply-templates select="firstName"/>
        <xsl:apply-templates select="lastName"/>
        <xsl:apply-templates select="email"/>
      </table>
      <div align="center">
        <hr/>
        <input type="submit" name="submitBtn"
                   value="Submit"/>
      </div>
    </form>
  </xsl:template>
```

– Each of the attribute values has got its own template for representation. Example 10-7 shows the template for the membership attribute. Each of the attributes has an identical template. So each attribute is included in a table row, consisting of two columns; one for the label, and the other for the input field.

Example 10-7 Stylesheet template for presenting the membership field

```
<xsl:template match="membership">
    <tr>
      <td>Membership: </td>
      <td><input name="membership" type="text" size="20"
         maxlength="40" value="membership"/></td>
    </tr>
  </xsl:template>
```

- **CustomerResult.xsl** represents the stylesheet for the target HTML representation of the retrieved customer registration information.
- The structure of the XML input data, which we can apply to this style sheet is shown in Example 10-8.

Example 10-8 Sample structure for the input XML data to the stylesheet

```
<Customer>
   <membership>........</membership>
   <firstName>........</firstName>
   <lastName>........</lastName>
   <email>........</email>
</Customer>
```

- The structure of the target HTML format for the customer XML data is shown in Example 10-9.

Example 10-9 Format of the target HTML representation for the customer data

```
<html>
<head>
   <title>Customer Registration Information</title>
</head>
<body>
   <h3>Customer info</h3>
   <table cellspacing="0" cellpadding="2" border="0">
      <tr>
         <td>Membership: </td>
         <td><b>........</b></td>
      </tr>
      <tr>
         <td>First Name: </td>
         <td><b>........</b></td>
      </tr>
      <tr>
         <td>Last Name: </td>
         <td><b>........</b></td>
      </tr>
      <tr>
         <td>Email: </td>
         <td><b>........</b></td>
      </tr>
   </table>
</body>
```

```
</html>
```

– The CustomerResult.xsl needs to construct an HTML file, and it should match that data from the XML data representation to insert it into the HTML file. So, in the code in Example 10-10, the XSL template puts the skeleton of the HTML document, specifying the necessary HTML tags, like head, body, etc. In the HTML body section, the XSL file specifies another template the matches the customer root element of the XML data. So, the HTML body represents a customer.

Example 10-10 Sample of CustomerResult stylesheet form

```
<xsl:template match="/">
    <html>
      <head>
          <title>Customer Registration Information</title>
          <style type="text/css">
               <![CDATA[
               body
               {
                 background-color: #f8f7cd;
               }
               h3
               {
                 color: #0000ff;
            text-align: center;
              }
               ]]>
          </style>
      </head>
      <body>
        <xsl:apply-templates select="Customer"/>
      </body>
    </html>
  </xsl:template>
```

– Example 10-11 shows the code for the customer template representing to be inserted in the HTML body. First it includes a heading representing the title of the page, then an HTML table is created for displaying the data. The table includes four templates to be matched from the XML data, which are the membership, firstName, lastName, and e-mail. So, the XSL selected from the XML files the element values representing the data.

Example 10-11 Stylesheet template for customer information representation

```
<xsl:template match="Customer">
    <h3>Customer Registration Information</h3>
      <table border="0" cellpadding="2" cellspacing="0">
        <xsl:apply-templates select="membership"/>
        <xsl:apply-templates select="firstName"/>
        <xsl:apply-templates select="lastName"/>
        <xsl:apply-templates select="email"/>
      </table>
  </xsl:template>
```

HTML Files: Under the /Web Content/WEB-INF folder, you will find the file Customer.html. This file only has a link to the CustomerXSLServlet. Clicking that link will invoke the servlet.

Schema Files: Under the same folder as the HTML file, you will find the schema file Customer.xsd. This is the schema for the XML data representing the customer registration information. Example 10-12 shows the contents of the schema file. It indicates that the root element customer is a complex type. It contains four elements; it shows their names and data types.

Example 10-12 Schema file for the XML data

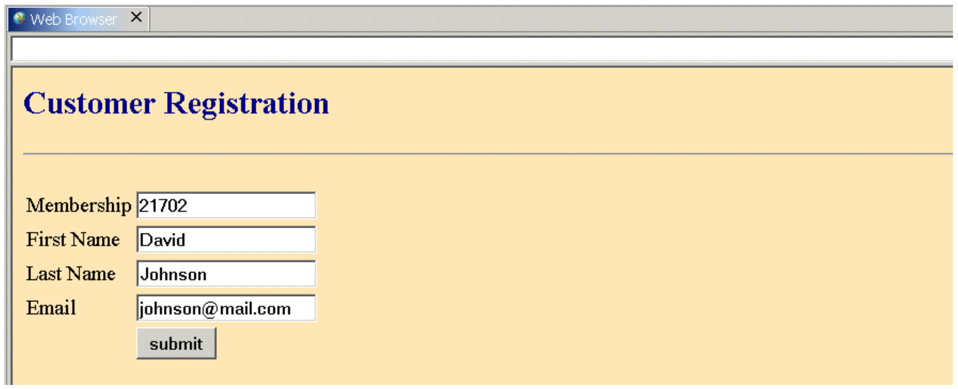
```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com" xmlns:Cu="http://www.ibm.com">
    <complexType name="Customer">
        <sequence>
            <element name="membership" type="Cu:Long"/>
            <element name="firstName" type="string"/>
            <element name="lastName" type="string"/>
            <element name="email" type="string"/>
        </sequence>
    </complexType>
    <complexType name="Long">
        <sequence>
        </sequence>
    </complexType>
</schema>
```

Validating the Web tier

Before we continue to the following step, we recommend that you validate the Web tier first. The validation process is very simple. Please do is the following:

1. Righ- click on the **Customer.html file**, and select **Run on Server** option. Choose to run the HTML file on the Websphere test environment server. This will automatically close the XML perspective, and open the server perspective. The Web browser will open running the HTML file. It might take some time to start the Websphere test environment.
2. Clink on **Click here to invoke the CustomerXSLServlet.** The registration form will load on the browser.

3. Enter the customer registration information, and click **Submit**, as shown in Figure 10-15.

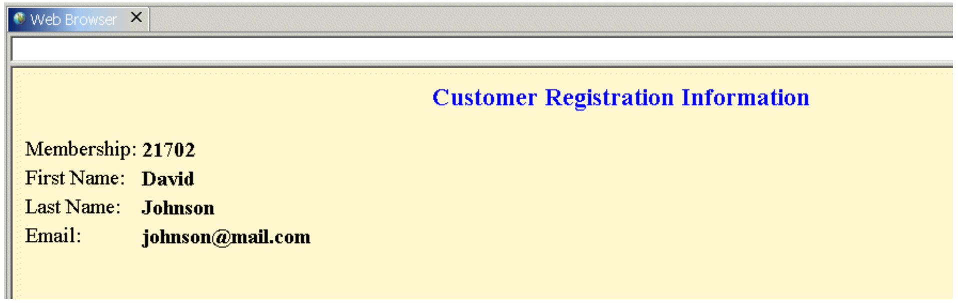


Customer Registration

Membership	21702
First Name	David
Last Name	Johnson
Email	johnson@mail.com

Figure 10-15 Customer registration form

The Submit button will call the servlet, which will in turn invoke the generation of a DOM representation of the entered data, then the result XSL will be applied on the generated DOM, in order to get the result form showing the customer's data, as shown in Figure 10-16.



Customer Registration Information

Membership:	21702
First Name:	David
Last Name:	Johnson
Email:	johnson@mail.com

Figure 10-16 Result customer registration information

10.3.2 Building the entity EJB and the database schema

We are now ready to develop our back end layer, which includes the database, and the business logic represented by the EJB, by performing the following steps:

1. Creating an entity EJB.
2. Creating an EJB to RDB mapping.
3. Generating the database schema and table.
4. Creating an access bean.
5. Creating a JDBC data source.

6. Binding the EJB to the JDBC data source.

EJB mapping approaches review

All of the EJB to relational mapping capabilities previously available in VisualAge for Java are now also available in Websphere Application Developer, although some techniques require a slightly different approach. As before, there are three different mechanisms:

- **Top down:** The developer creates the CMP entity beans in the application, and the database schema and mapping are generated from this definition.
- **Bottom up:** An existing database schema exists for the application, which is imported into the development environment, and used to generate the CMP entity beans and the mappings.
- **Meet in the middle:** This scenario is useful when there is a requirement to map a series of new or existing entity beans into an existing database schema.

For simplicity, we are using the top down approach, where we create the entity bean, and use it to create the database schema and table definition.

Preparing to create the entity EJB

Websphere Studio Application Developer provides a wizard that facilitates the creation of the entity EJB. Before we start using that wizard, we must perform an important step.

Project configuration

Make sure that you create CustomerInfoEJB project and it is EJB 1.1 complient. Switching to the J2EE perspective, and in particular J2EE view. The CustomerInfoEJB project should appear attached to the node EJB modules.

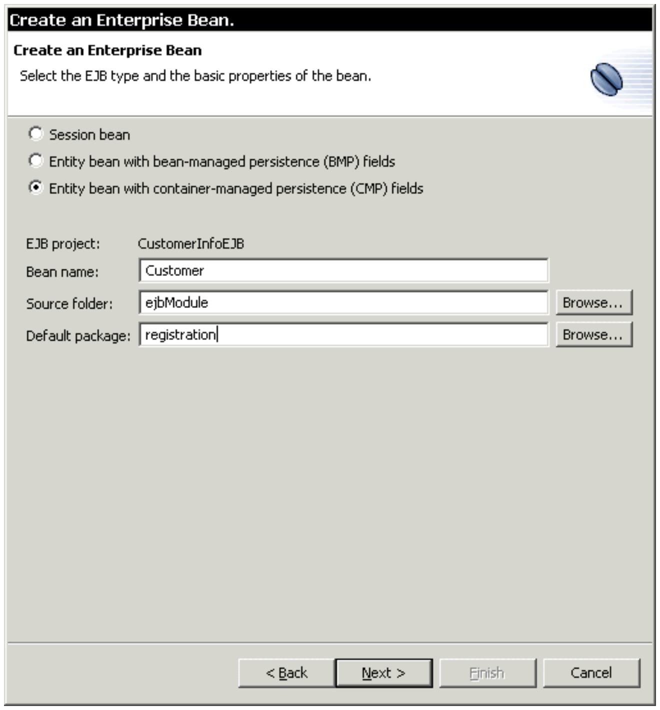
Creating the entity EJB

To create the entity EJB using the top down approach, we do the following:

Under the EJB Modules category in the J2EE perspective, select the CustomerInfo project, right-click, and select **New—>Enterprise Bean.**

The wizard for creating the Enterprise bean opens Figure 10-17. To create the bean do the following:

1. Select **Entity Bean** with container-manages persistence (CMP) fields. Enter the bean name as `Customer`, and the package name as `registration.`
2. Click **Next**.


Create an Enterprise Bean.
Create an Enterprise Bean
Select the EJB type and the basic properties of the bean.
Session bean
Entity bean with bean-managed persistence (BMP) fields
Entity bean with container-managed persistence (CMP) fields
Browse...
Browse...
< Back
Next >
Finish
Cancel

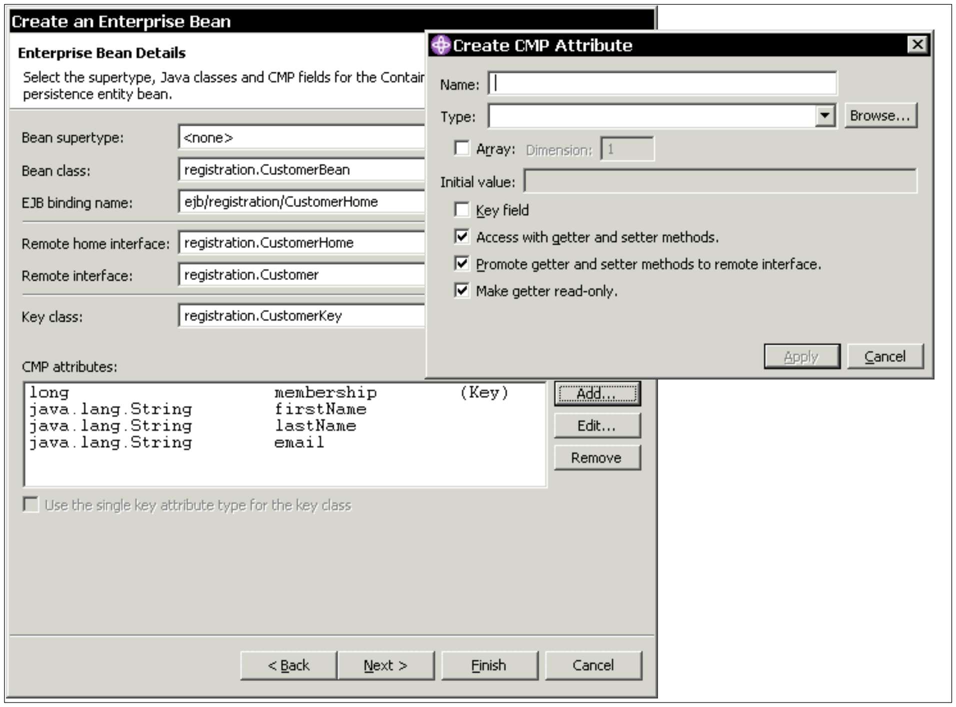
( ) Session bean
( ) Entity bean with bean-managed persistence (BMP) fields
(•) Entity bean with container-managed persistence (CMP) fields

EJB project: CustomerInfoEJB
Bean name: Customer
Source folder: ejbModule
Default package: registration

Figure 10-17 Create an Enterprise Bean wizard

Now we need to define all the CMP attributes that we need to include in the entity bean. To do that:

1. Click the **Add** button, to open the window for CMP attribute creation, as shown in Figure 10-18 on page 241.
2. Enter membership as the attribute name, and choose its type to be long. Select the **Key Field** option to indicate that membership is key for that entity, then click **Apply**.
3. After adding the membership, add the rest of the attributes, which are the firstName, lastName, and e-mail. Note that these attributes are of type string, and that you do not need to select the key field option, because these attributes are not keys. The membership attribute is the only key for the customer.
4. Click **Finish**.


Create an Enterprise Bean
Enterprise Bean Details
Select the supertype, Java classes and CMP fields for the Contair persistence entity bean.
Create CMP Attribute
Browse...
Apply
Cancel
Add...
Edit...
Remove
< Back
Next >
Finish
Cancel

Bean supertype: <none>
Bean class: registration.CustomerBean
EJB binding name: ejb/registration/CustomerHome
Remote home interface: registration.CustomerHome
Remote interface: registration.Customer
Key class: registration.CustomerKey

CMP attributes:

```
long                 membership       (Key)
java.lang.String     firstName
java.lang.String     lastName
java.lang.String     email
```

Use the single key attribute type for the key class

Name:
Type:
Array: Dimension: 1
Initial value:
Key field
✓ Access with getter and setter methods.
✓ Promote getter and setter methods to remote interface.
✓ Make getter read-only.

Figure 10-18 Enterprise bean details

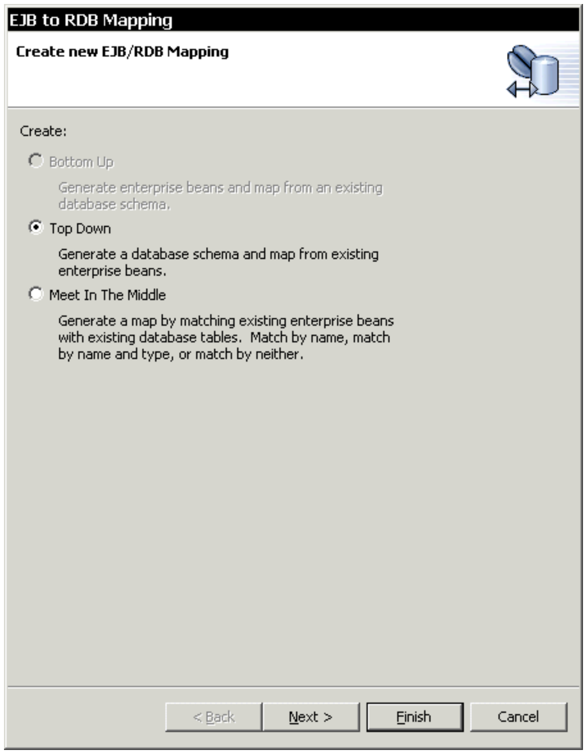
Creating the database mapping and tables

Now that the entity bean is ready, we need to create the database mapping:

1. Under the EJB modules category, select the **CustomerInfoEJB** project, and select **Generate —> EJB to RDB Mapping** from the menu.

   The mapping wizard opens Figure 10-19 on page 242. Because we have no existing schema currently defined in our project, Bottom Up is disabled at this time.

2. Select **Top Down**.
3. Click **Next**.


EJB to RDB Mapping

Create new EJB/RDB Mapping

Create:

( ) Bottom Up

Generate enterprise beans and map from an existing database schema.

(•) Top Down

Generate a database schema and map from existing enterprise beans.

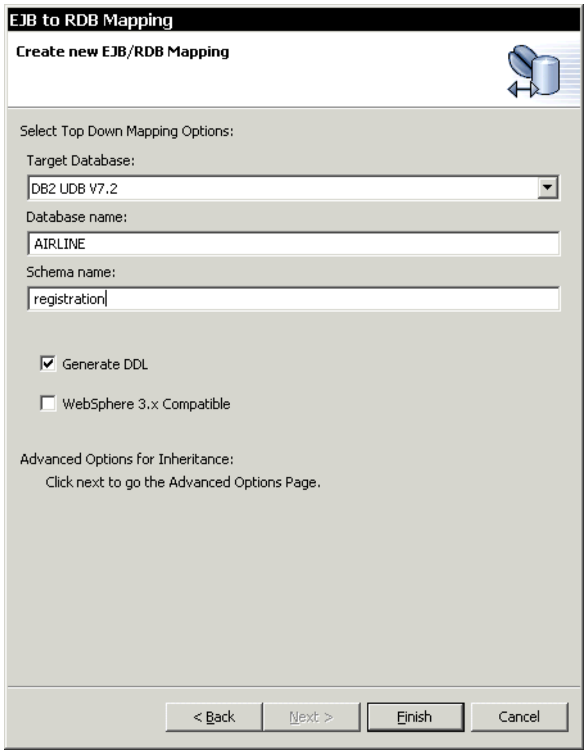
( ) Meet In The Middle

Generate a map by matching existing enterprise beans with existing database tables. Match by name, match by name and type, or match by neither.

Figure 10-19 EJB to RBD Mapping wizard for CustomerInfo module

To define the database information on the target database information window:

1. Complete the database information as shown in Figure 10-20. Enter the database name. In our case, we will use the same **`Airline`** database, instead of creating a new one. You may want to create a new database for this scenario. This is totally up to you. Select a schema name (for example, **Registration**) and make sure that the Generate DDL check box is selected.
2. Click **Finish**.


EJB to RDB Mapping

Create new EJB/RDB Mapping

Select Top Down Mapping Options:

Target Database: DB2 UDB V7.2

Database name: AIRLINE

Schema name: registration

[✓] Generate DDL

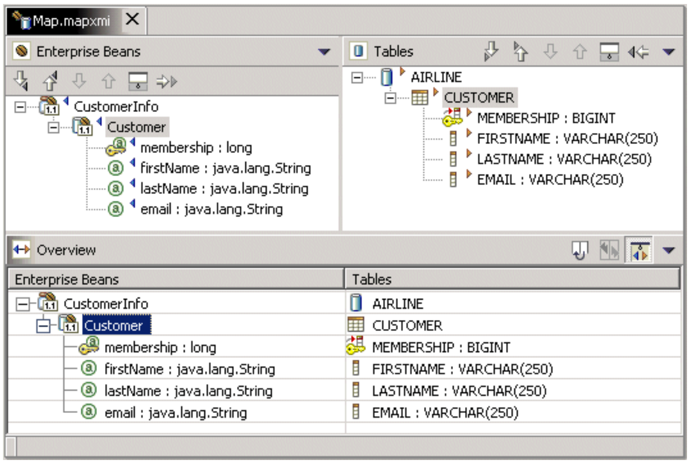
[ ] WebSphere 3.x Compatible

Advanced Options for Inheritance:

Click next to go the Advanced Options Page.

Figure 10-20 Target database specification

After the EJB to RDB mapping is generated, the mapping editor opens. The editor shows the mapping between our EJB project CustomerInfoEJB, and the Airline database, as shown in Figure 10-21. The customer entity bean is mapped to the customer table, and so are the EJB attributes and table columns.

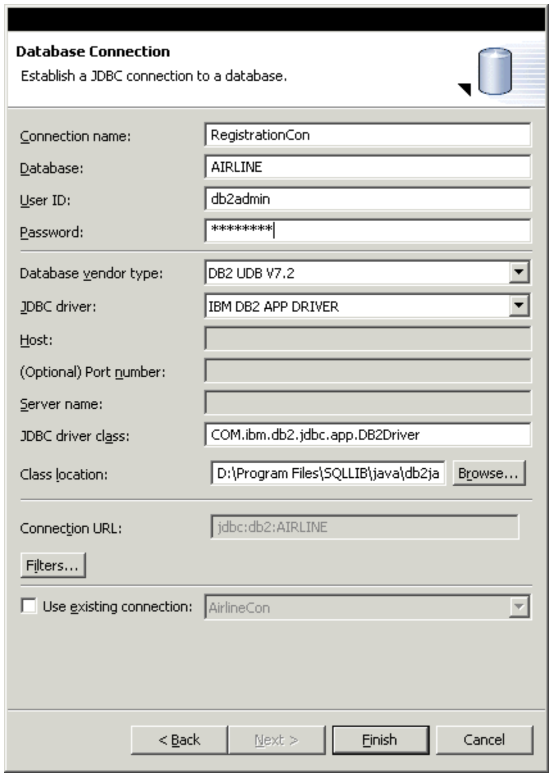


Enterprise Beans	Tables
CustomerInfo	AIRLINE
Customer	CUSTOMER
membership : long	MEMBERSHIP : BIGINT
firstName : java.lang.String	FIRSTNAME : VARCHAR(250)
lastName : java.lang.String	LASTNAME : VARCHAR(250)
email : java.lang.String	EMAIL : VARCHAR(250)

Figure 10-21 EJB to RDB mapping editor

To create the customer table in the database:

1. Close the mapping editor.
2. Close the J2EE perspective, and open the data perspective. Choose the data definition view.
3. Expand the `CustomerInfoEJB` project. Under the `/ejbModule/META-INF`, you will find `Table.ddl` file.
4. Run the `Table.ddl` file on server. This will launch the ddl file.
5. Make sure that all of the script commands are selected. Click **Next**.
6. Select **Commit changes** only upon the success option.
7. Click **Next**.
8. Complete the database connection information as shown in Figure 10-22.
9. Click **Finish** to create the schema and the table.



Database Connection

Establish a JDBC connection to a database.

Connection name:	RegistrationCon
Database:	AIRLINE
User ID:	db2admin
Password:	*********
Database vendor type:	DB2 UDB V7.2
JDBC driver:	IBM DB2 APP DRIVER
Host:	
(Optional) Port number:	
Server name:	
JDBC driver class:	COM.ibm.db2.jdbc.app.DB2Driver
Class location:	D:\Program Files\SQLLIB\java\db2ja
Connection URL:	jdbc:db2:AIRLINE
☐ Use existing connection:	AirlineCon

Figure 10-22 Database connection definition

Access beans

Access beans are Java components that adhere to the JavaBeans specification, and are meant to simplify the development of EJB clients. An access bean adapts an Enterprise bean to the Javanese programming model, by hiding the home and remote interfaces from the developer. They provide fast access to Enterprise beans by letting the developer maintain a local cache of Enterprise bean attributes. Access beans make it possible to use an Enterprise bean in much the same way that a JavaBean is used.

There are three types of access beans, which are listed in ascending order of complexity:

- **Data class**: The simplest to use. It is designed to allow the Enterprise bean to be used like a standard JavaBean, and it hides the Enterprise bean home and remote interfaces from the developer.
- **Copy helpe**r: This has all the characteristics of the data class access bean, but it also incorporates a single copy helper object that maintains a local copy

of attributes from a remote entity bean. A program can retrieve the entity bean attributes from the local copy helper object that resides in the access bean, which eliminates the need to access the attributes from the remote entity bean.

- **JavaBean Wrapper**: This has all the characteristics of both the data class and the copy helper access beans. However, instead of a single copy helper object, it contains multiple copy helper objects. Each copy helper object corresponds to a single Enterprise bean instance.

Now that we have described access beans, we should realize that they improve performance, because they provide the client a caching mechanism for accessing homes. Moreover, we can use access beans in servlets and JSPs, because they simplify coding.

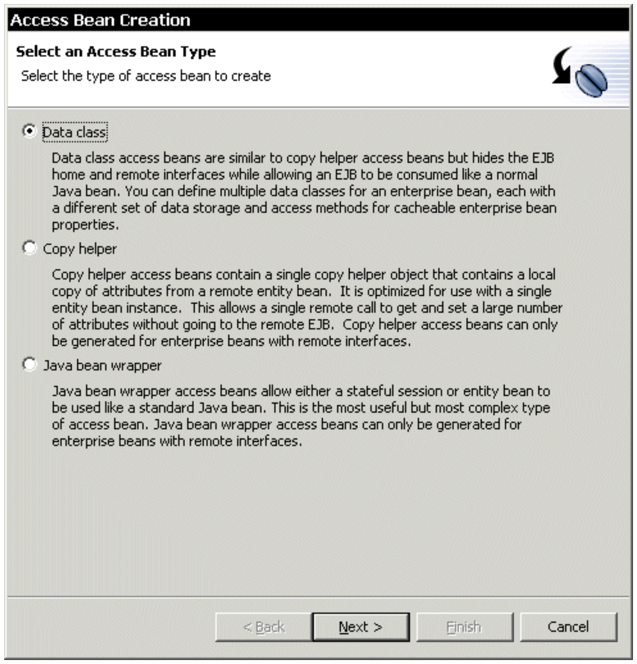
Creating an access bean for the entity bean

To create the access bean for the entity EJB:

1. Close the data perspective, and open the J2EE perspective, and choose the J2EE Hierarchy.
2. Under the EJB Modules category, select the **CustomerInfo project**, right-click, and select **New—>Access Bean.**

The wizard for creating the Enterprise bean opens Figure 10-23. To create the bean do the following:

1. Choose the type of the access bean to be a Data class, which will allow the EJB to be consumed like a normal JavaBean.


Access Bean Creation

Select an Access Bean Type

Select the type of access bean to create

(•) Data class

Data class access beans are similar to copy helper access beans but hides the EJB home and remote interfaces while allowing an EJB to be consumed like a normal Java bean. You can define multiple data classes for an enterprise bean, each with a different set of data storage and access methods for cacheable enterprise bean properties.

( ) Copy helper

Copy helper access beans contain a single copy helper object that contains a local copy of attributes from a remote entity bean. It is optimized for use with a single entity bean instance. This allows a single remote call to get and set a large number of attributes without going to the remote EJB. Copy helper access beans can only be generated for enterprise beans with remote interfaces.

( ) Java bean wrapper

Java bean wrapper access beans allow either a stateful session or entity bean to be used like a standard Java bean. This is the most useful but most complex type of access bean. Java bean wrapper access beans can only be generated for enterprise beans with remote interfaces.

Figure 10-23 Access Bean creation wizard

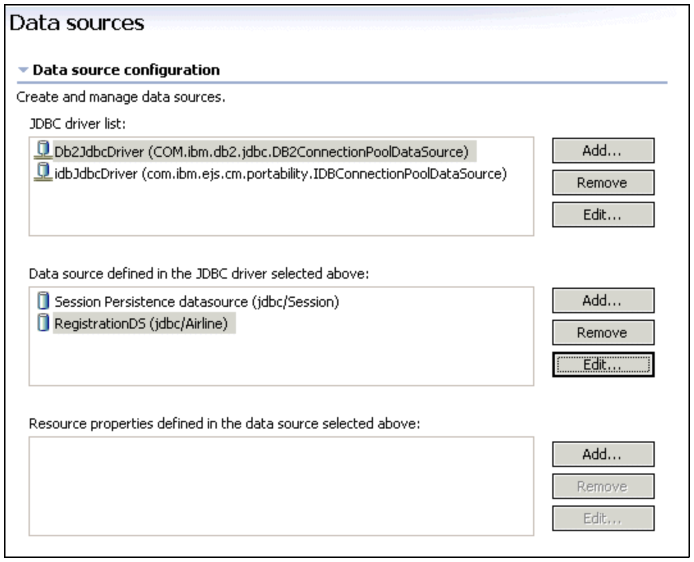
2. Click **Next**.
3. Check to see that the EJB project selected is the **CustomerInfoEJB** project. From the list of Enterprise JavaBeans, select the check box next to the customer entity bean, and click **Next**.
4. Select the Enterprise bean from the list to change the default access bean info to create data class access bean.
5. Click **Finish**.

In the J2EE perspective, expand the `CustomerInfoEJB` project in the Navigator view. You will find two new java classes CustomerData.java and CustomerFactory.java under /ejbModule/registration. These two represent the generated data class access bean.

Creating a JDBC data source

For our application to access the database, we need to create a datasource. The data source must be defined in the application server configuration, and in our case the application server is the Websphere Test Environment v4.0. In order to create the data source, we need to do the following:

1. In the server perspective and in particular in the server configuration window, select the WebSphere administrative domain attached to the server configurations category. Right-click and select **Open** from the menu to open the editor.
2. Switch to the Data Source tab Figure 10-24.


Data sources

Data source configuration

Create and manage data sources.

JDBC driver list:

- Db2JdbcDriver (COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource)
- idbJdbcDriver (com.ibm.ejs.cm.portability.IDBConnectionPoolDataSource)

Data source defined in the JDBC driver selected above:

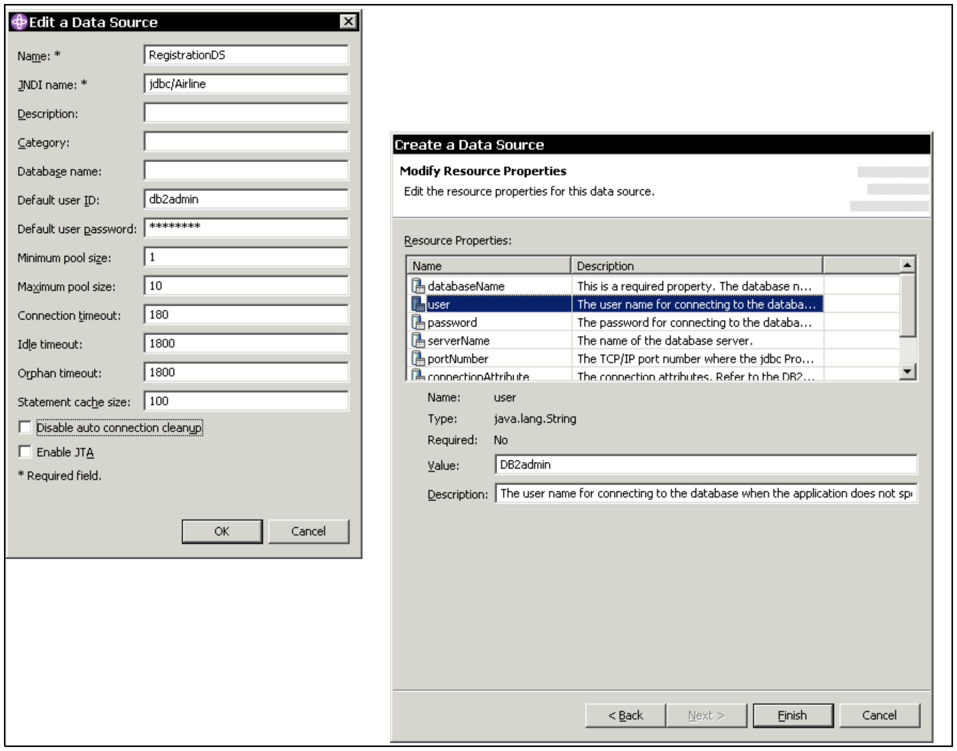
- Session Persistence datasource (jdbc/Session)
- RegistrationDS (jdbc/Airline)

Resource properties defined in the data source selected above:

Figure 10-24 Data source definition for a server configuration

To add a new DB2 JDBC data source:

1. Select the **DB2JdbcDriver** item from the list.
2. Click the **Add** button next to the data source list.
3. Complete the data in the dialog box (Figure 10-25.) Enter the datasource name as **`RegistrationDS`**, and the JNDI name as **`jdbc/Airline`**, and set the user ID and password as **`db2admin`**.
4. Set database name as **`AIRLINE`**.
5. Click **OK**.
6. Close the configuration editor, and save the changes.



Name: *	RegistrationDS
JNDI name: *	jdbc/Airline
Description:	
Category:	
Database name:	
Default user ID:	db2admin
Default user password:	*********
Minimum pool size:	1
Maximum pool size:	10
Connection timeout:	180
Idle timeout:	1800
Orphan timeout:	1800
Statement cache size:	100

☐ Disable auto connection cleanup

☐ Enable JTA

* Required field.

Modify Resource Properties

Edit the resource properties for this data source.

Resource Properties:

Name	Description
databaseName	This is a required property. The database n...
user	The user name for connecting to the databa...
password	The password for connecting to the databa...
serverName	The name of the database server.
portNumber	The TCP/IP port number where the jdbc Pro...
connectionAttribute	The connection attributes. Refer to the DB2...

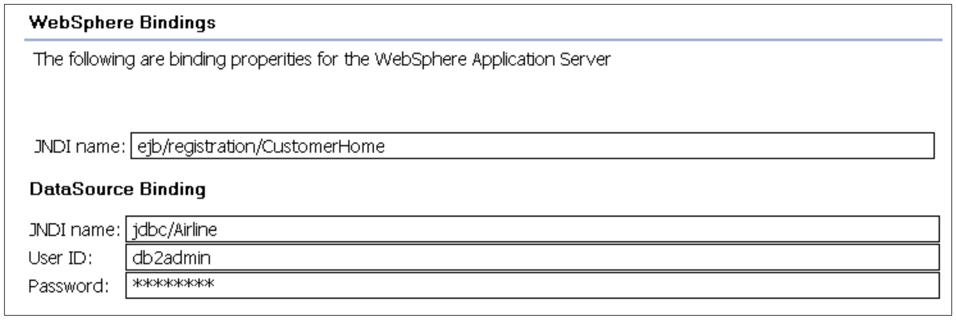
Name:	user
Type:	java.lang.String
Required:	No
Value:	DB2admin
Description:	The user name for connecting to the database when the application does not sp

Figure 10-25 JDBC data source definition

Binding the EJB to a JDBC data source

Each EJB module must be assigned to a JDBC data source reference in order for its CMP entities to persist correctly. To perform this assignment:

1. Switch the J2EE perspective. Select the **J2EE Hierarchy** view. Select the **Customer** bean. Right-click and select **Open With —>Deployment Descriptor Editor**. The editor will open with the bean's tab selected.
2. Complete the data in the section labeled DataSource Binding Figure 10-26. Enter the JNDI name as **`jdbc/Airline`**, which matches the JNDI name for the data source created in Figure 10-25. Enter the user ID and password as **`db2admin`**.
3. Save the changes and close the editor.


WebSphere Bindings

The following are binding properities for the WebSphere Application Server

JNDI name: ejb/registration/CustomerHome

DataSource Binding

JNDI name: jdbc/Airline
User ID: db2admin
Password: ********

Figure 10-26 Defining the JDBC data source for the EJB

10.3.3 Integrating the entity EJB with the Web tier

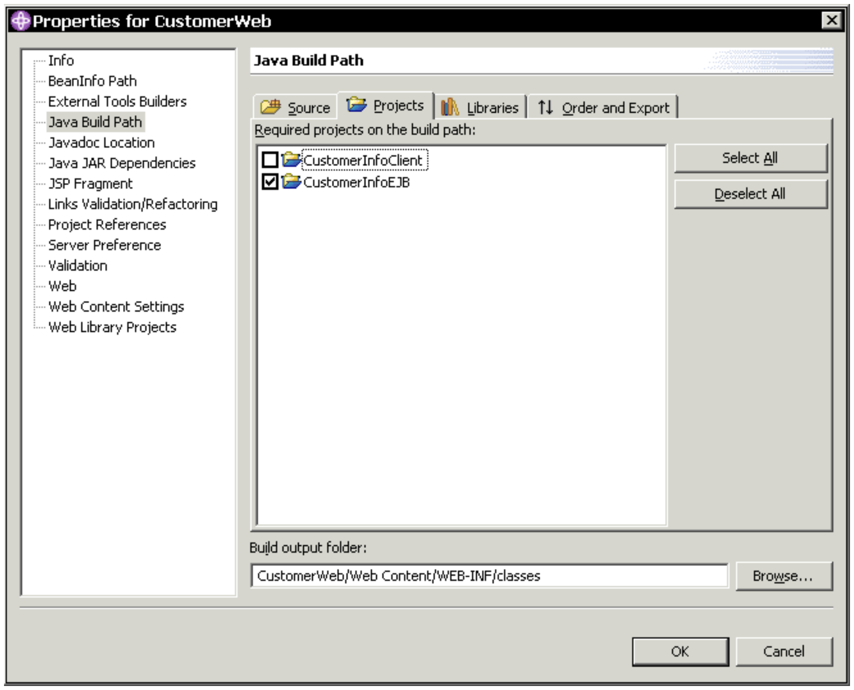
In order to integrate the Customer EJB that we have created, with our Web tier, we need to do the following:

1. Modify the `CustomerXML` (DOM) included in the Web tier, to reference the generated `Customer` EJB instead of the original `Customer` JavaBean.
2. Add a create method to the CustomerXML (DOM) to construct the entity bean using the CustomerFactory. The original code uses the JavaBean constructor to create a new Customer object. We cannot do that with the entity bean, so we use the EJB factory's create method.
3. Modify the CustomerXSLServlet to allow storage and retrieval of customer data.
4. Modify the CustomerXSLServlet to invoke the create method we created.

Modifying CustomerXML

To make the `CustomerXML` (DOM) reference our `Customer` EJB, we first need to include the `CustomerInfoEJB` project, into the class path of the `CustomerInfoWeb` project.

1. Switch to the Web perspective.
2. Right-click on the CustomerInfoWeb project. Choose the project properties from the menu. Choose the Java Build Path, and in the Projects tab, select the check box next to the CustomerInfoEJB project, as in Figure 10-27 on page 251.
3. Click **OK** to save your changes to the project properties.



Required projects on the build path:

- [ ] CustomerInfoClient
- [✓] CustomerInfoEJB

Build output folder: CustomerWeb/Web Content/WEB-INF/classes

Figure 10-27 CustomerInfoWeb project properties

You can start now on modifying the `CustomerXML` file, by doing the following:

- Open the `CustomerXML.java` file to start editing it. Modify the import statements to include exactly the following:

```
import javax.ejb.*;
import java.rmi.*;
import registration.Customer;
import registration.CustomerKey;
import registration.CustomerFactory;
import org.w3c.dom.*;
import javax.xml.parsers.*;
```

- Implement a create method that constructs a new CustomerFactory, and invoke its create method using Customer bean's properties, as shown in Example 10-13.

Example 10-13 create source code

```
protected void create() throws
RemoteException,CreateException {
   CustomerFactory factory = new CustomerFactory();
   registration.Customer ejbean =
                      factory.create(getBean().getMembership().longValue());
   ejbean.setFirstName(getBean().getFirstName());
   ejbean.setLastName(getBean().getLastName());
   ejbean.setEmail(getBean().getEmail());
}
```

Modifying CustomerXSLServlet

The changes to the CustomerXSLServlet are in two directions. Changes to the doPost method of the servlet that enables storing new customer records in the database. While the changes to the doGet method of the servlet enables retrieving customer information from the database, based on a membership input value. To achieve our target, we need to perform the following to the CustomerXSLServlet:

1. Modify the servlet’s import statements. Add the following imports:

   ```
   import registration.*;
   import javax.ejb.*;
   ```

2. Now in the doPost method, invoke the CustomerXML create method, after setting up all property values. You have to catch the exceptions that might be thrown, and return to the input form. See Example 10-14.

Example 10-14 CustomerXSLServlet doPost source code

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
    CustomerXML xml = getCustomerXML();

   xml.setMembership(new Long(request.getParameter("membership")));
    xml.setFirstName(new java.lang.String(request.getParameter("firstName")));
    xml.setLastName(new java.lang.String(request.getParameter("lastName")));
    xml.setEmail(new java.lang.String(request.getParameter("email")));

    try{
      //Write to EJB.
      xml.create();
    }catch(CreateException crateEx){
      showPage(mainStylesheet, response);
    }
    showPage(resultStylesheet, response);
  }
```

10.3.4 Retrieval function

This registration application does not offer the retrieval function. To add the retrieval function to CustomerXML class, we add retrieve method as Example 10-15. This function retrieves Enterprise bean by the membership key, then copy the properties from customer EJB to customer bean.

Example 10-15 Retrieval function

```
protected void retrieve(Long membership) throws RemoteException,
                                         FinderException
  {
   CustomerFactory factory = new CustomerFactory();
   registration.Customer ejbean = factory.findByPrimaryKey(new
CustomerKey(membership));

   setFirstName(ejbean.getFirstName());
   setLastName(ejbean.getLastName());
   setEmail(ejbean.getEmail());
   setMembership(membership);
  }
```

In the next chapter, we show how to include the retrieve function into the registration form. At this time, we add a statement to confirm the registration. In the doPost method, we retrieve the customer using the membership key just after create (see Example 10-16).

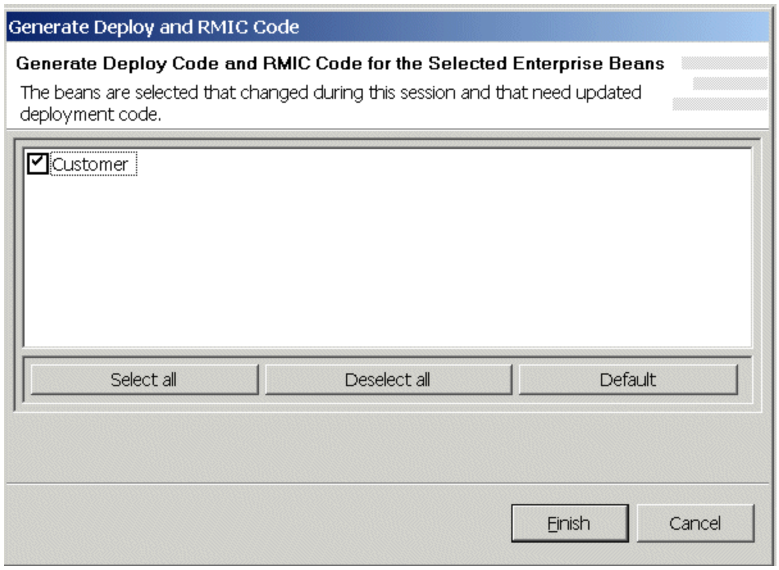
Example 10-16 Confirming the creation

```
   try{
       xml.retrieve(new Long(request.getParameter("membership")));
       showPage(resultStylesheet, response);
    }catch(FinderException remoteEx){
    }
```

10.4 Application deployment and testing

We are now ready to generate the deployed code to support the execution on WebSphere test environment:

1. Switch to the J2EE Hierarchy view, select the **CustomerInfoEJB** under the EJB Modules, and select **Generate—>Deploy and RMIC Code.**
2. Make sure that the customer entity EJB is selected, and click **Finish,** as in Figure 10-28.


Generate Deploy and RMIC Code

Generate Deploy Code and RMIC Code for the Selected Enterprise Beans

The beans are selected that changed during this session and that need updated deployment code.

☑ Customer

Figure 10-28 Generating the deployed code for the EJB

After switching to the Navigator view, expand the CustomerInfo project, which is the registration node under the ejbModule folder to see the deployed code.

Now that we have completed the development of the whole application, and deployed the EJB, it is time to move to testing that application using WebSphere Test Environment Version 4.0.

10.4.1 Testing the registration application

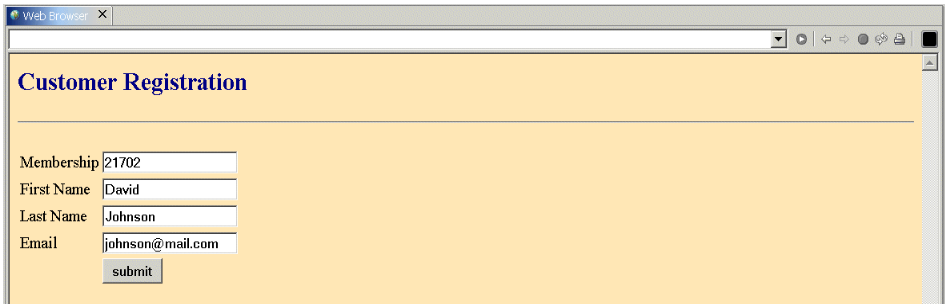
We need to test the registration process:

1. Switch to the server perspective, select the registration project. Select **CustomerXMLServlet,** and select the option **Run on Server** from the menu.

   The WebSphere test Environment will start, launching the browser, and invoking the JSP in Figure 10-29.

2. Enter the necessary data, which are the membership, first name, last name, and e-mail.

3. Click the **Submit** button.

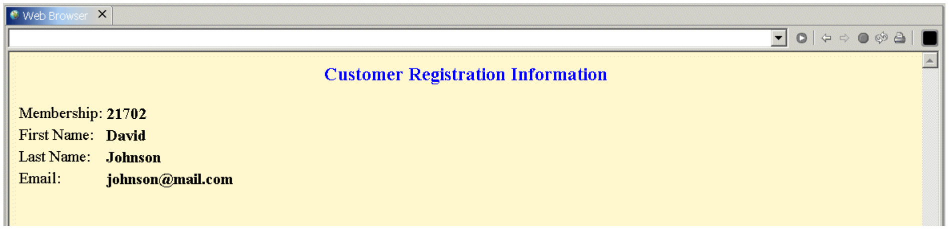


Customer Registration

Membership	21702
First Name	David
Last Name	Johnson
Email	johnson@mail.com

Figure 10-29 Customer Registration form

Once the **Submit** button is clicked, the doPost is invoked. It extracts the data from the HTTP request, and constructs a new customer entity bean, and the data is inserted into the database. Then the CustomerXML converts the bean into a DOM representation. The servlet finally applies the output style sheet on the create DOM representation creating the HTML output in Figure 10-30.



Customer Registration Information

Membership:	**21702**
First Name:	**David**
Last Name:	**Johnson**
Email:	**johnson@mail.com**

Figure 10-30 HTML resulting from the XSLT transformation

11

Light weight XML-based Enterprise Application

Chapter eleven provides an another approach to develop an XML based Enterprise application without Enterprise JavaBean. Application Developer has two libraries to query and insert/update the database. The SQLToXML library is used to perform a query and the XMLToSQL library is used to perform an insert or update. Both libraries work with all JDBC databases such as DB2, Oracle, Sybase, MS-SQL Server, etc. These are useful to the simple application that does not need to use Enterprise JavaBeans.

In this chapter, the following topics are described:

- Architecture of light weight XML based Enterprise applications
- Development of such applications using Application Developer
- Deployment and testing of the solution on the Application Developer

11.1 SQL to/from XML libraries

In this chapter, we replace the EJB part which is described in previous chapter with SQL to/from XML libraries. These libraries are exactly same as used in XML from SQL query and XML, to SQL wizard. You can use the XML from SQL wizard to unit test your query to make sure it produces the expected result (see 6.1, "The SQL to XML wizards" on page 102). And you should use the XML to SQL wizard to unit test the XML document to make sure it updates the right data.

This simple architecture (illustrated in Figure 11-1) can keep the application simple. The application server layer has been removed from J2EE architecture and SQL <-> XML libraries will handle the database directory from Web tier inside.

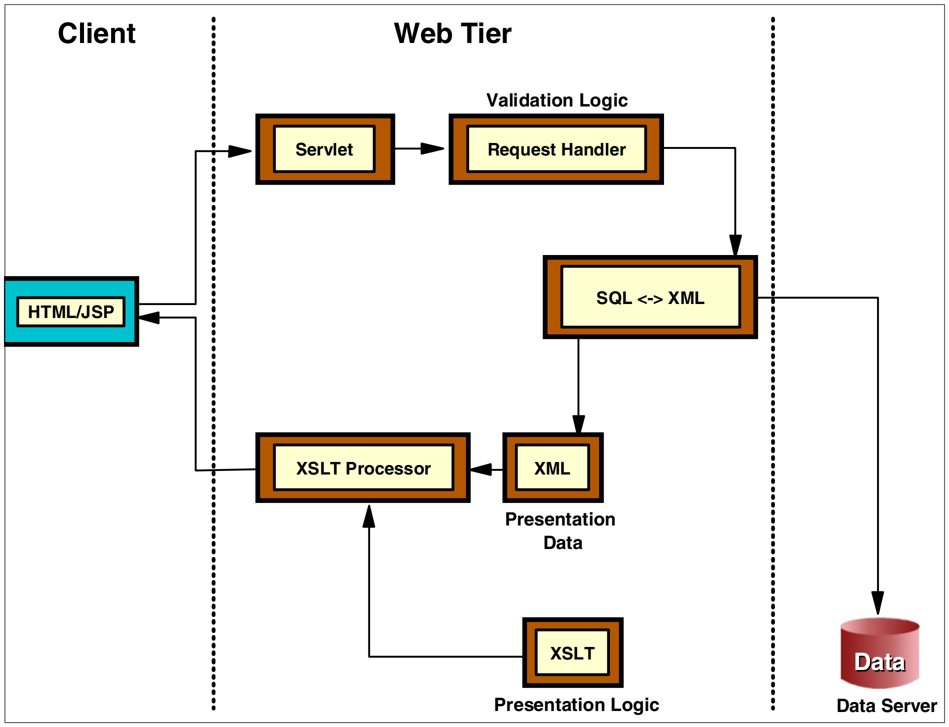


Figure 11-1 SQL<->XML solution architecture

In order to understand how these components interact with each other, have a deep look at 11.2, "Solution outline for customer registration sample" on page 259.

11.2 Solution outline for customer registration sample

An overview of the sample application can be found in Chapter 8., “WebSphere and XML approaches” on page 163. This chapter discusses the interaction between solution components, before we progress to building the application. There are two possible ways of interaction between the components of the system. This difference in interaction is based on the target functionality. We focus on the set of interactions for storing data, then we discuss the set of interactions for retrieving customer information.

11.2.1 Customer registration

When registering, the user enters the necessary registration data in the form provided by the HTML in the client layer. When the user presses the Submit button, a series of interactions between system components take place, as indicated in Figure 11-2 on page 260, and in the following steps:

1. The HTML sends the customer registration data through the HTTP request to the servlet, invoking the servlet doPost method.
2. The servlet extracts the data from the request, and creates a new CustomerXML object and set required properties with the extracted registration data.
3. The servlet invokes Customer XML object’s produceDOMSource method and get an XML document.

> **Tip:** The XML document can be created manually using JAXP.

4. The servlet invokes XMLToSQL library.
5. XMLToSQL executes INSERT using the XML document.
6. The servlet instantiate XSLT Processor.
7. The XML document is input for the XSLT Processor.
8. The pre-designed XSL applied and result HTML has been generated and transferred to the browser.

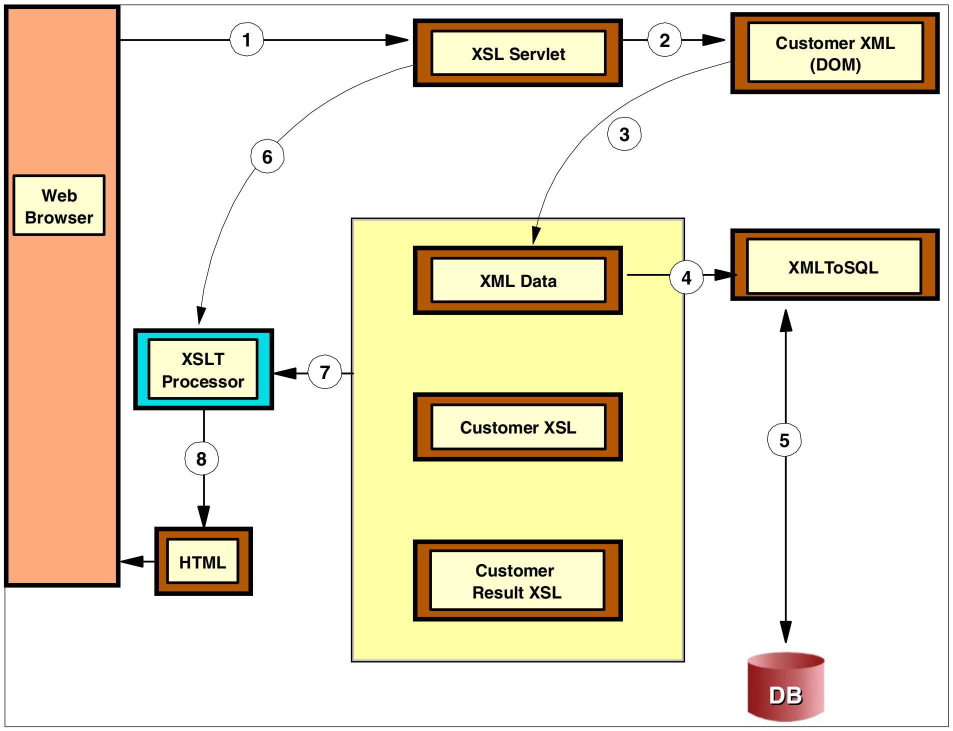


Figure 11-2 Customer registration scenario outline

11.2.2 Retrieving customer information

When retrieving customer data, the user enters the membership number of the customer whose information is to be retrieved in the form provided by the HTML in the client layer. When the user presses the Query button, a series of interactions between system components take place, as indicated in Figure 11-3, and in the following steps:

1. The HTML sends that value to the servlet through the HTTP request to the servlet, invoking the servlet doPost method.
2. The servlet validates that the user has entered a membership value. Then the servlet invokes the SQLToXML library with one parameter, membership number.
3. The SQLToXML retrieves the data from the database and constructs an XML document.

Tip: Application Developer 4.0.3 contains SQLToXML library, but it returns results as a stream data, which means you need to parse it for a document. Later versions may have a function to get a document directly.

4. The servlet gets an XML document from SQLToXML library.
5. The servlet instantiate XSLT Processor.
6. The XML document that is the output of SQLToXML is an input for the XSLT Processor.
7. The pre-designed XSL applied and result HTML has been generated and transferred to the browser, so when the transformation is performed, the user can view the retrieved customer information on the Web browser.

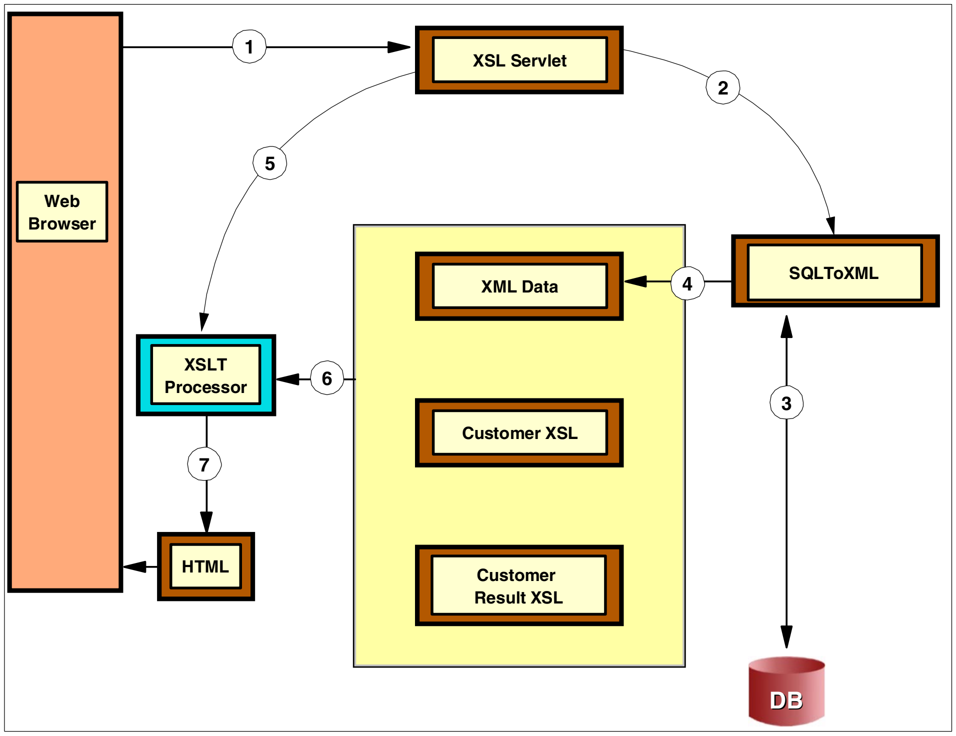


Figure 11-3 Retrieving customer data scenario outline

11.3 Developing the customer registration sample

We are going to reuse customer registration sample code that we developed in previous chapter. In order to develop this application, modify CustomerXSL servlet and XSLs:

- Re-generate all related files from customer bean.
- Modify the CustomerXSL servlet to allow storage and retrieval of customer data.

11.3.1 Adding the libraries to the project

You need to add two libraries to your project. Use the property window of the project. Two variables are defined in Application Developer. To add the jars to the classpath, use add variable button, and add XMLTOSQL and SQLTOXML. For your information, the jars are located under the following directories:

XMLToSQL	plugins\com.ibm.etools.sqltoxml_5.0.0\jars\xmltosql.jar
SQLToXML	plugins\com.ibm.etools.sqltoxml_5.0.0\jars\sqltoxml.jar

11.3.2 XML Document format

One thing to be aware of is that the XML format is not compatible with our customerXSL. The following XML is a sample that the libraries accept:

```
<SQLResult>
 <Customer>
  <firstname>firstname</firstname>
  <lastname>lastname</lastname>
  <email>email</email>
  <membership>membership</membership>
 </Customer>
</SQLResult>
```

The document root is typically `SQLResult`. And the first child, in this case, `Customer`, is used as a table name, afterwards each column should follow (Figure 11-6).

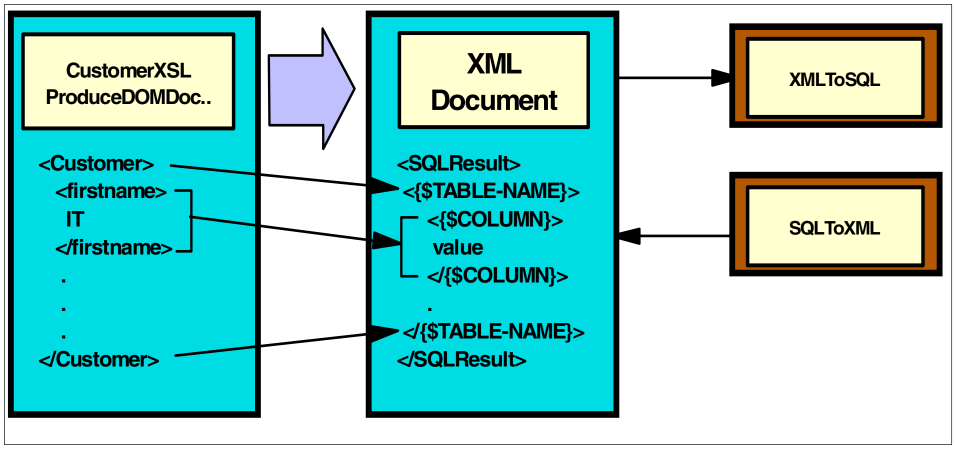


Figure 11-4 Converting an XML Document

11.3.3 XMLToSQL architecture

The XMLToSQL takes one parameter to initialize. That parameter is SQLProperty class and it contains following data:

JdbcDriver	JDBC Driver name
LoginId	Database login id
Password	Database login password
JdbcServer	Server name (ex. jdbc:db2:Airline)
Schema	Set Schema if it is different
Action	INSERT or UPDATE

Tip: What is the schema? When you use SELECT * from db2admin.employee, *db2admin* is the schema.

Once the XMLToSQL is instantiate, invoke the execute method to insert or update. XMLToSQL to connect to the database, and insert or update the data, then disconnect from the database (Figure 11-5).

Connection pooling or datasource is not supported in this release. But XMLToSQL has a property to set a connection. By using this property, an application can set the connection. We will discuss how to do it later.

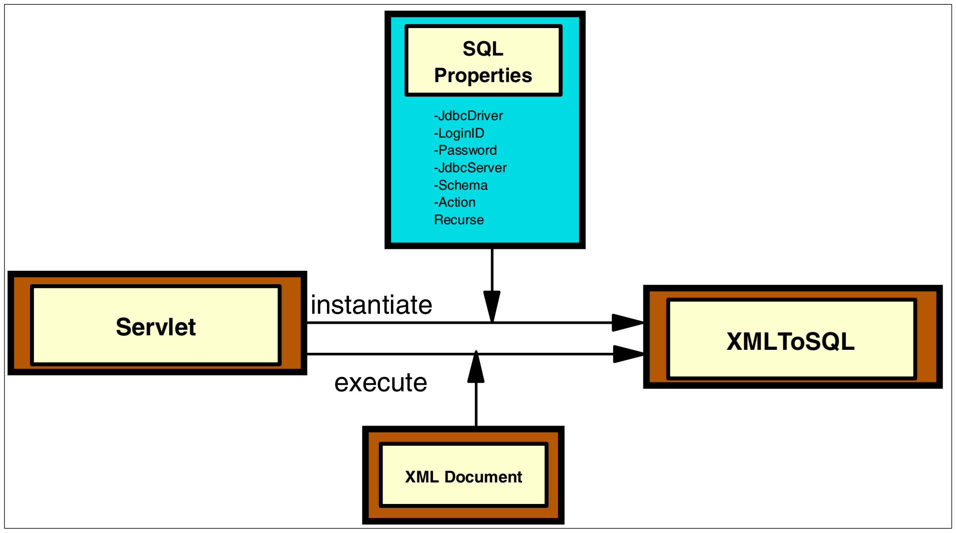


Figure 11-5 XMLToSQL

11.3.4 Modifying CustomerXSLServlet

The change to the CustomerXSLServlet is the doPost method of the servlet, which enables storing new customer records in the database. Since doGet returns the blank input form, we left as is. To achieve our target, we need to perform the following to the CustomerXSLServlet:

- Initialize XMLToSQL to work with the database.
- Modify the doPost method to execute the insert.

Initializing the XMLToSQL class

We created separate initSQLProperties method to setup the property. Since we use same property during this application running, we initialize this property in the init method.

- Add SqlProperties as a global variable:

  ```
  private SQLProperties sqlProperties;
  ```

- Add XMLToSQL as a global variable:

  ```
  private XMLToSQL xml2sql;
  ```

- Add initSQLProperties method (Example 11-1);

  We recommend that you create a .properties file to keep the information to set the SQLProperties.

- In the init method, add a statement to invoke the initSQLProperties method:

```
initSQLProperties();
```

- Also, we instantiate XMLToSQL in the init method:

```
xml2sql = new XMLToSQL(sqlProperties);
```

Example 11-1 initSQLProperties method

```
private void initSQLProperties()
  {
    sqlProperties = new SQLProperties();
    sqlProperties.setJdbcDriver("COM.ibm.db2.jdbc.app.DB2Driver");
    sqlProperties.setLoginId("db2admin");
    sqlProperties.setPassword("db2admin");
    sqlProperties.setSchema("registration");
    sqlProperties.setJdbcServer("jdbc:db2:AIRLINE");
    sqlProperties.setAction(SQLProperties.INSERT);
  }
```

Creating an XML Document

We reused CustomerXSL to generate an XML Document which contains new customer data (Figure 11-6). To do this:

1. Instantiate CustomerXSL.
2. Extract customer data from HTTPRequest.
3. Set data to CustomerXSL.
4. Get the XML Document invoking CustomerXSL’s produceDOMDocument.

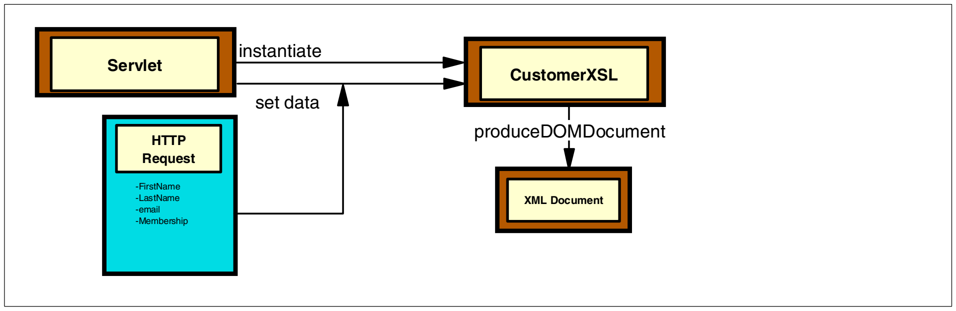


Figure 11-6 Getting an XML Document using CustomerXSL

Modifying the produceDOMDocument method

As we mentioned, the XMLToSQL does not accept the document. XMLToSQL is expecting the following document. Currently, CustomerXSL produces following XML Document:

```
<Customer>
  <firstname>firstname</firstname>
  <lastname>lastname</lastname>
  <email>email</email>
  <membership>membership</membership>
</Customer>
```

We need to add SQLResult as a root element for this document and change all tags related to the table name or column names to CAPITAL. The easiest way is to modify the produceDOMDocument method in the CustomerXSL class. The following code fragment is the original code.

```
Element rootElement = doc.createElement("Customer");
     doc.appendChild(rootElement);
```

We need to change to:

```
Element superRootElement = doc.createElement("SQLResult");
     doc.appendChild(superRootElement);
Element rootElement = doc.createElement("Customer");
     superRootElement.appendChild(rootElement);
```

We created a new produceDOMDocumentforTools method as follows.

Example 11-2 ProduceDOMDocumentforTools

```
public Document produceDOMDocumentforTools()
      throws ParserConfigurationException
   {
     // use Sun's JAXP to create the DOM Document
     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
     DocumentBuilder docBuilder = dbf.newDocumentBuilder();
     Document doc =  docBuilder.newDocument();

     Element parent = null;

     //Now we need SQLResult as a root.
    Element superRootElement = doc.createElement("SQLResult");
     doc.appendChild(superRootElement);
    Element rootElement = doc.createElement("CUSTOMER");

     superRootElement.appendChild(rootElement);
     addElement(doc, rootElement, "FIRSTNAME",
                convertToString(getFirstName()));
     addElement(doc, rootElement, "LASTNAME",
                convertToString(getLastName()));
     addElement(doc, rootElement, "MEMBERSHIP",
                convertToString(getMembership()));
     addElement(doc, rootElement, "EMAIL",
                convertToString(getEmail()));
return doc;
}
```

Or, you can use JAXP to change the document structure dynamically. Using JAXP following the code fragment does this. If this document's root element is not SQLResult, add SQLResult and add the first child (in our case, it is customer), to the SQLResult element. At this time, the customer element was removed from the root and be a child of the SQLResult. Then add the SQLResult to the document, SQLResult will be a root and customer will be a first child:

```
if( !(doc.getFirstChild().getNodeName().equalsIgnoreCase("SQLResult")) ){
        Element rootElement = doc.createElement("SQLResult");
       rootElement.appendChild(doc.getFirstChild());
       doc.appendChild(rootElement);
}
```

We introduce a wrapper of XMLToSQL to resolve this and add a capability of the datasource.

Executing the XMLToSQL class

We created doRegister method to instantiate CustomerXSL, get an XML Document, and execute XMLToSQL in one method (Example 11-3). Since XMLToSQL can throw an exception, we need to catch them. This sample shows if SQLException has been thrown, the member ship already exists.

This method returns a true if it succeeds, false if it fails.

Example 11-3 CustomerXSLServlet doRegister method

```
private boolean doRegister(HttpServletRequest request){
    CustomerXML xml = getCustomerXML();
    xml.setFirstname(new java.lang.String(request.getParameter("firstname")));
    xml.setLastname(new java.lang.String(request.getParameter("lastname")));
    xml.setEmail(new java.lang.String(request.getParameter("email")));
    xml.setMembership(new
java.lang.String(request.getParameter("membership")));
    try{
    xml2sql.execute(xml.produceDOMDocumentforTools(),false);
    }catch(SQLException sqlexception){
    System.out.println(sqlexception);
    xml.setMembership("Membership is already exist");
    return false;
    }catch(Exception exception){
    System.out.println(exception);
    return false;
    }
    return true;
  }
```

Modifying the doPost method

Now we need to modify the doPost method. All we need is to invoke is the doRegister method with HttpServletRequest. Once the doRegister succeeds, go showpage to show results. If it fails, go back to the input form, which will be reproduced by the doGet method. The doGet method reuses the XML Document that is currently instantiated in the servlet instance, and the membership field has been replaced with **-1** message.

Example 11-4 CustomerXSLServlet doPost method

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
   if( doRegister(request) == true){
       showPage(resultStylesheet, response );
   }else{
       doGet(request, response);
   }
   }
```

If the membership already exists, Customer.xsl will be told as the membership is -1. To handle this to the meaning message, we used xsl:if tag.

Example 11-5 The xsl:if tag

```
<td>membership
            <xsl:if test = "$membr = -1" >
             *
      </xsl:if>
      :</td>
```

In this case, add * next to the membership. Or, use this tag to show “Membership is already exist” in the input field, or message area.

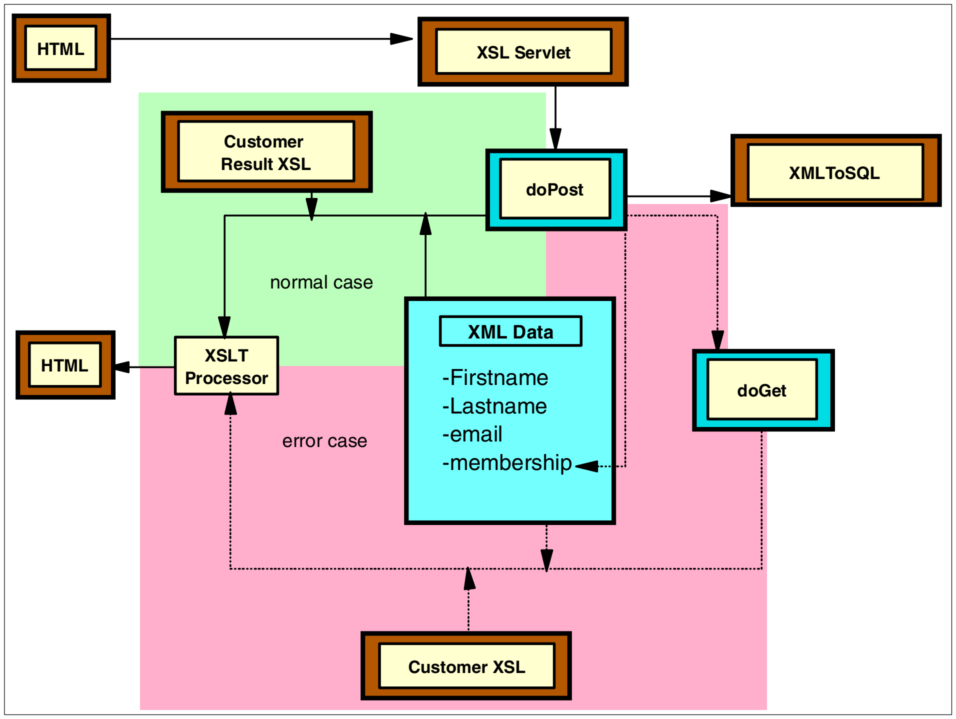


Figure 11-7 The doPost method

Adding the error case

In the error case, we need to set back the data that was originally posted from the browser. Unless this functions, all user typed information has been cleared when the error occurred. To bring back the data to the browser, we need to modify the template. The original value attribute of the input tag is blank. To set back with the value that was previously typed (the XML data contains that data), use variable {$fname} in the value attribute. To set this variable, use xsl:variable tag and use the value of the current position. You need this modification to all fields.

Example 11-6 Customer.XSL firstname template

```
<xsl:template match="firstname">
    <xsl:variable name="fname">
     <xsl:value-of select="."/>
    </xsl:variable>
    <tr>
      <td>Firstname: </td>
      <td><input name="firstname" type="text" size="20" maxlength="40"
value="{$fname}"/></td>
    </tr>
```

```
</xsl:template>
```

11.3.5 Retrieving a customer

We used SQLToXML library to retrieve a customer.

The SQLToXML takes one parameter to initialize. That parameter is the QueryProperty class and it contains following data:

JdbcDriver	JDBC driver name
LoginId	Database login ID
Password	Database login password
JdbcServer	Server name (ex. jdbc:db2:Airline)
Statement	Query statement
Format	Use GENERATES_AS_ELEMENTS
Recurse	False

Once the SQLToXML is instantiate, invoke the execute method to query. The execute method takes a parameter to set a host variable. SQLToXML connects to the database and executes a query then generates the result as an XML form and disconnects from the database (Figure 11-8).

The host variable can be set as comma delimited. For example, to set two variables for the following select statement, the parameter will be `new String("Air Canada 700, Jet 787");`

```
SELECT *  FROM PASSENGER, SCHEDULE
 WHERE  PASSENGER.FLIGHT = SCHEDULE.FLIGHTNO
   AND PASSENGER.FLIGHT = :flight
   AND SCHEDULE.AIRCRAFT = :aircraft
```

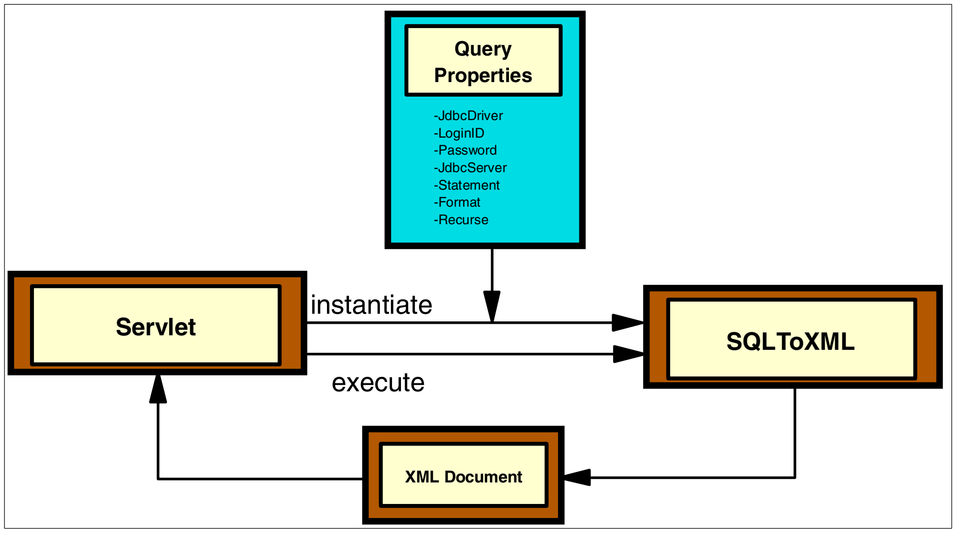


Figure 11-8 SQLToXML

Initializing the SQLToXML class

We created a separate initQueryProperties method to setup the property. Since we use same property during this application running, we initialize this property in the init method.

- Add QueryProperties as a global variable:

  ```
  private QueryProperties queryProperties;
  ```

- Add initQueryProperties method (Example 11-7).
- In the init method, add a statement to invoke the initQueryProperties method.

  ```
  initQueryProperties();
  ```

- Also, we instantiated SQLToXML in the init method:

  ```
  sql2xml = new SQLToXML(queryProperties);
  ```

Example 11-7 initQueryProperties method

```
private void initQueryProperties()
{
    queryProperties = new QueryProperties();
    queryProperties.setJdbcDriver(
             "COM.ibm.db2.jdbc.app.DB2Driver");
    queryProperties.setLoginId("db2admin");
    queryProperties.setPassword("db2admin");
    queryProperties.setJdbcServer("jdbc:db2:AIRLINE");
    queryProperties.setStatement(
       "SELECT * FROM CUSTOMER WHERE SERIAL = :membership");
```

```
    queryProperties.setFormat("GENERATE_AS_ELEMENTS");
    queryProperties.setRecurse(false);
  }
```

Executing the SQLToXML class

We created the doQuery method to execute the query and get an XML Document as a result (Example 11-8). SQLToXML returns XML data as a stream, and it only accepts an instance of PrintWriter. To get it as a string data, you need to create an instance of StringWriter, then create an instance of PrintWriter from the StringWriter.

Tip: A future release of Application Developer contains a method to get a document. We tested this method and it works. When it is available, you can get a document directory from SQLToXML.

Example 11-8 CustomerXSLServlet doQuery method

```
//For Application Developer 4.0.3
private StringWriter doQuery(HttpServletRequest request){
   StringWriter xmldata = new StringWriter();
   PrintWriter xmlwr = new PrintWriter(xmldata);
    try{
    sql2xml.execute(new java.lang.String(
          request.getParameter("memberhsip")),xmlwr,null,null,null);
    }catch(Exception exception){
    System.out.println(exception);
    }
    return xmlData;
  }
//For Application Developer 5.0
private org.w3c.dom.Document doQuery(HttpServletRequest request){
   StringWriter xmldata = new StringWriter();
   PrintWriter xmlwr = new PrintWriter(xmldata);
    try{
   sql2xml.setParameters(new java.lang.String(
          request.getParameter("memberhsip")));
    sql2xml.execute();
    }catch(Exception exception){
    System.out.println(exception);
    }
    return sql2xml.getDocument();

}
```

Modifying the doPost method

Now we need to modify the doPost method. All we need to do is invoke the doQuery method with HttpServletRequest. We are reusing same form with registration, we need to add another button to query. We define the button as *queryBtn*, and check in the doPost method. If queryBtn is pressed, doPost proceeds to doQuery and shows. Once the doQuery succeeds, go showpage to show results. If it fail, go back to the input form which will be reproduced by doGet method. The doGet method reuses the XML Document that is currently instantiated in the servlet instance, and the membership field has been replaced with `-1`, which is the `Membership already existed` message.

Example 11-9 CustomerXSLServlet doPost method for query

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
   if( request.getParameter("queryBtn") != null ){
       showPage(resultStylesheet, response, doQuery(request));
   }else{
      if( doRegister(request) == true){
          showPage(resultStylesheet, response, doQuery(request));
      }else{
         doGet(request, response);
      }
   }    }
```

Modifying the showPage method

The showPage method takes an output document as the third parameter. The parameter is org.w3c.dom.Document type, so to run under Application Developer 4.0.3, you need to use following statement:

```
transformer.transform(new StreamSource(
             new StringReader(resultdata)), new StreamResult(writer));
```

Example 11-10 showPage method

```
private void showPage(Templates stylesheet, HttpServletResponse response,
org.w3c.dom.Document doc)
    throws IOException
  {
    try
    {
      Transformer transformer = stylesheet.newTransformer();
      response.setContentType("text/html");
      PrintWriter writer = response.getWriter();
      transformer.transform(new DOMSource(doc), new StreamResult(writer));
}
    catch (Exception ex)
    {
      PrintWriter pw = response.getWriter();
      pw.println("<html><body><h2>Transformation Error</h2><pre>");
      ex.printStackTrace(pw);
      pw.println("</pre></body></html>");
    }
  }
```

Modifying the doGet method

Regarding the above modification, doGet must be changed to set document.

Example 11-11   showPage method

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {
   try{
      showPage(mainStylesheet, response,
          getCustomerXML().produceDOMDocument());
   }catch(Exception e){
      System.out.println(e);

   }

}
```

Modifying the XSL file

Since our main HTML is a dynamic thing, we can modify the transformation file to add Query Button to the main form (Figure 11-9). Open Customer.xsl file and look for membership template (Example 11-12).

Example 11-12   Customer.XSL membership template

```
<xsl:template match="membership">
    <xsl:variable name="num">
     <xsl:value-of select="."/>
    </xsl:variable>
    <tr>
      <td>Membership: </td>
      <td><input name="memberhsip" type="text" size="20" maxlength="40"
value="{$num}"/>
          <input type="submit" name="queryBtn" value="Query"/>
     </td>
    </tr>
```

```
</xsl:template>
```

Also, result XSL needs a modification. As we mentioned about the SQLResult element, the result document's root is also SQLResult. You need to modify the result XSL to match the document, and all returned tags are capitalized as we mentioned. Complete Result XSL is as follows (Figure 11-13).

Example 11-13 Result XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/1999/xhtml">

  <xsl:output method="xml" indent="yes" encoding="UTF-8"
     doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
     doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>

  <xsl:template match="/SQLResult|/">
    <html>
      <head>
          <title>Registration Result</title>
          <style type="text/css">
               <![CDATA[
               body
               {
                 background-color: #ffffff;
               }
               h3
               {
                 color: #0000ff;
            text-align: center;
               }
               ]]>
          </style>
      </head>
      <body>
        <xsl:apply-templates select="CUSTOMER"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="Customer|CUSTOMER">
    <h3>Registration Result</h3>

      <table border="0" cellpadding="2" cellspacing="0">
        <xsl:apply-templates select="firstName|FIRSTNAME"/>
        <xsl:apply-templates select="lastName|LASTNAME"/>
        <xsl:apply-templates select="membership|MEMBERHSIP"/>
        <xsl:apply-templates select="email|EMAIL"/>

      </table>
  </xsl:template>

  <xsl:template match="firstName|FIRSTNAME">
    <tr>
      <td>firstName: </td>
      <td><b><xsl:value-of select="."/></b></td>
    </tr>
  </xsl:template>

  <xsl:template match="lastName|LASTNAME">
    <tr>
      <td>lastName: </td>
      <td><b><xsl:value-of select="."/></b></td>
    </tr>
  </xsl:template>

  <xsl:template match="membership|MEMBERSHIP">
    <tr>
      <td>membership: </td>
      <td><b><xsl:value-of select="."/></b></td>
    </tr>
  </xsl:template>

  <xsl:template match="email|EMAIL">
    <tr>
      <td>email: </td>
      <td><b><xsl:value-of select="."/></b></td>
    </tr>
  </xsl:template>


</xsl:stylesheet>
```

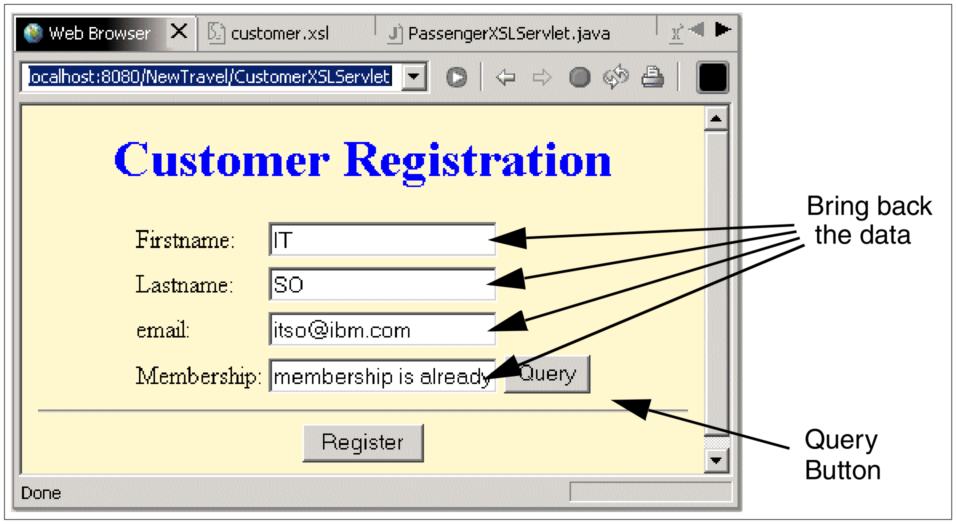


Figure 11-9 Customer Registration form

11.3.6 Using datasource with SQLToXML and XMLToSQL class

Since SQLToXML and XMLToSQL were developed as a part of wizard, they were not designed to accept the datasource. To use with datasource and the connection pooling, we can use set/getConnection methods, and once the connection has been set to the class, it will not try to connect to the database itself. So, we extended both classes to support the datasource. The new classes inherit the original class, so new classes can use all functions of the originals (Figure 11-10).

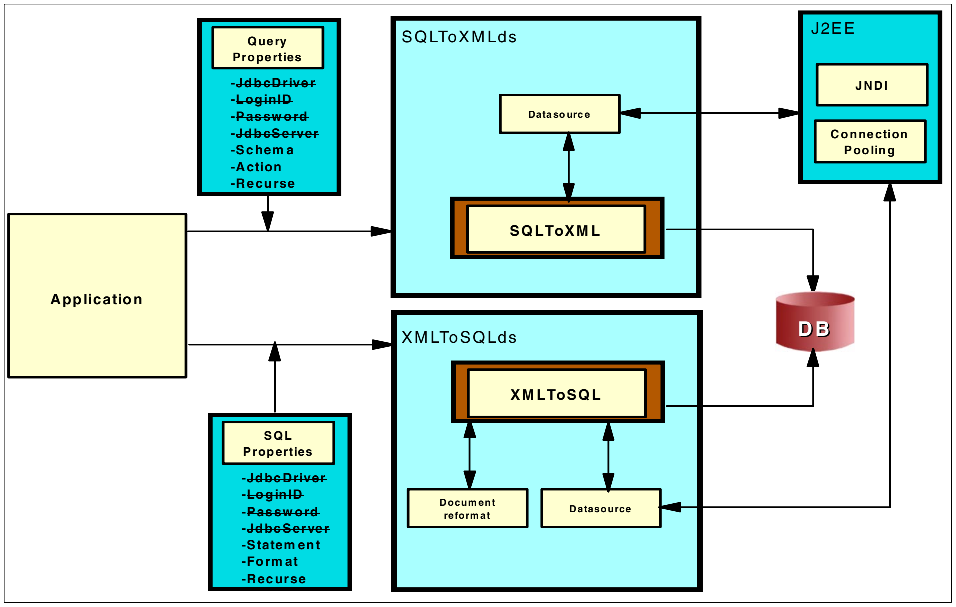


Figure 11-10 SQLToXMLds and XMLToSQLds

Tip: You need to add naming.jar from runtime library under runtime/base_v5/lib.

Class definition

Both SQLToXMLds and XMLToSQLds extend SQLToXML and XMLToXML. Each class has a global variable to keep the datasource. The following code fragment shows the class definition:

```
public class XMLToSQLds extends XMLToSQL{

   private DataSource datasource;
}

public class SQLToXMLplus extends SQLToXML{

   private DataSource datasource;
}
```

Getting the datasource

We add to methods to set datasource. We used two approaches to get the datasource.

Example 11-14 setDatasourceMethods

```
/**
      * Setting up datasource
      * @param dsname String.
      * @param dsclassname String DataSource class name
ex.COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource.
      * @exception CMFactoryException
      * @exception ClassNotFoundException
      */
    public void setDataSource(String dsname, String dsclassname)
     throws CMFactoryException,
          ClassNotFoundException
    {
      java.util.Properties prop = new java.util.Properties();
      prop.put(DataSourceFactory.NAME, dsname);
      prop.put(DataSourceFactory.DATASOURCE_CLASS_NAME,dsclassname);
      datasource = DataSourceFactory.getDataSource(prop);
    }

    /**
      * Setting up datasource
      * @param dsname String.
      * @exception javax.naming.NamingException
      */
    public void setDataSource(String dsname)
     throws NamingException
    {
      java.util.Properties prop = new java.util.Properties();
      prop.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
      Context ctx = new javax.naming.InitialContext(prop);
      datasource = (DataSource)ctx.lookup(dsname);

}
```

Constructors

New constructors have a second parameter to set the datasouce. It accepts the datasource itself or the datasource name.

Example 11-15 setDatasourceMethods

```
   /**
    * Constructor for SQLToXMLds.
    */
   public SQLToXMLds(QueryProperties qprop, DataSource ds) {
      super(qprop);
      datasource = ds;
   }

   /**
    * Constructor for SQLToXMLds.
    */
   public SQLToXMLds(QueryProperties qprop, String dsname) {
      super(qprop);

    try{
         setDataSource(dsname);
    }catch(NamingException e){
      System.out.println(e);
    }
   }
```

Execute methods

All execute methods are overridden and try to get a connection through the datasource:

```
   public void execute(String s0, String s1, String s2, String s3) throws
Exception{
      setConnection();
      super.execute(s0,s1,s2,s3);
   }

   private void setConnection() throws SQLException{
      if(datasource != null ){
         super.setConnection(datasource.getConnection());
      }
   }
```

Once you created the code, modify your servlet to use the SQLToXMLds, XMLToSQLds instead of SQLToXML and XMLToSQL. You need to define the datasource in the WebSphere Test Environment setting, and need to add the datasource name where you are invoking the constructor of them. Now you can use datasource. You still need to set the QueryProperties or SQLProperties, jdbc driver name, loginid, password, and jdbcservernames are no longer needed.

SQLResult is also resolved in the execute method of XMLToSQLds class. The overrided execute class is doing this function as follows:

```
public void execute(Document doc, boolean continueOnSQLError)
       throws SQLException,           // connection creation failed
              ClassNotFoundException, // jdbc driver is missing
              IOException,            // invalid input stream
              SAXException,           // parse error
              ParserConfigurationException
```

```
        {
          if(datasource != null ){
             super.setConnection(datasource.getConnection());
          }

          if( !(doc.getFirstChild().getNodeName().equalsIgnoreCase("SQLResult"))
){

              Element rootElement = doc.createElement("SQLResult");
             rootElement.appendChild(doc.getFirstChild());
             doc.appendChild(rootElement);
          }

          super.execute(doc,continueOnSQLError);
        }
```

If this Document's root element is not SQLResult, add SQLResult, and add the first child (in our case, it is customer), to the SQLResult element. At this time, the customer element was removed from the root and is a child of the SQLResult. Then add the SQLResult to the document, SQLResult will be a root, and customer will be a first child.

11.3.7 Conclusion

SQLToXML and XMLToSQL can be the XML front end of the database. Using these libraries, we can use the XML data as the core data of the application (Figure 11-11).

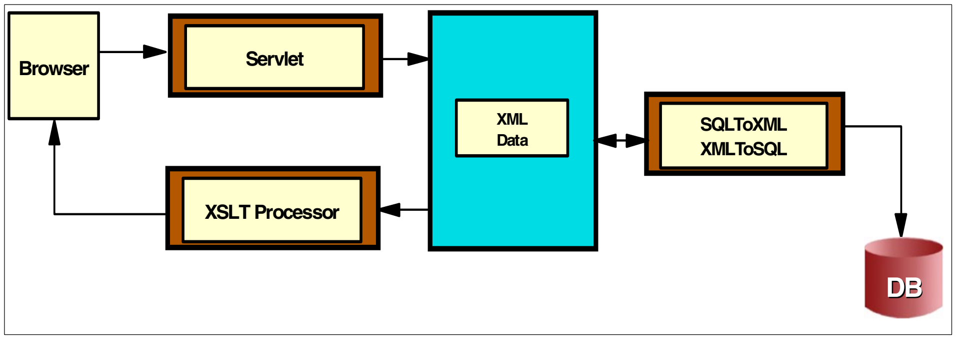


Figure 11-11 Core data is an XML

You have seen how to use Application Developer to:

- Update the database with dynamically generated XML within a servlet using XMLToSQL class library.

- Dynamically generate XML from SQL queries at run time using the SQLToXML class library within a servlet.
- How to use the datasource and the connection pooling within XMLToSQL and SQLToXML libraries within a servlet.

12

Deploying your Web application

Deployment of a Web application can be done manually or automatically using Application Developer.

This chapter describes the following:

- Manual deployment:
  - Exporting your EAR from Application Developer
  - Installing the EAR file on WebSphere AEs
  - Starting the WebSphere AEs admin GUI
  - Installing the EAR on WebSphere AEs
  - Configuring your server
  - Testing the application
- Automatic deployment to a remote server:
  - Creating a remote server instance
  - Publishing to remote server
  - Testing the application

12.1 Manual deployment

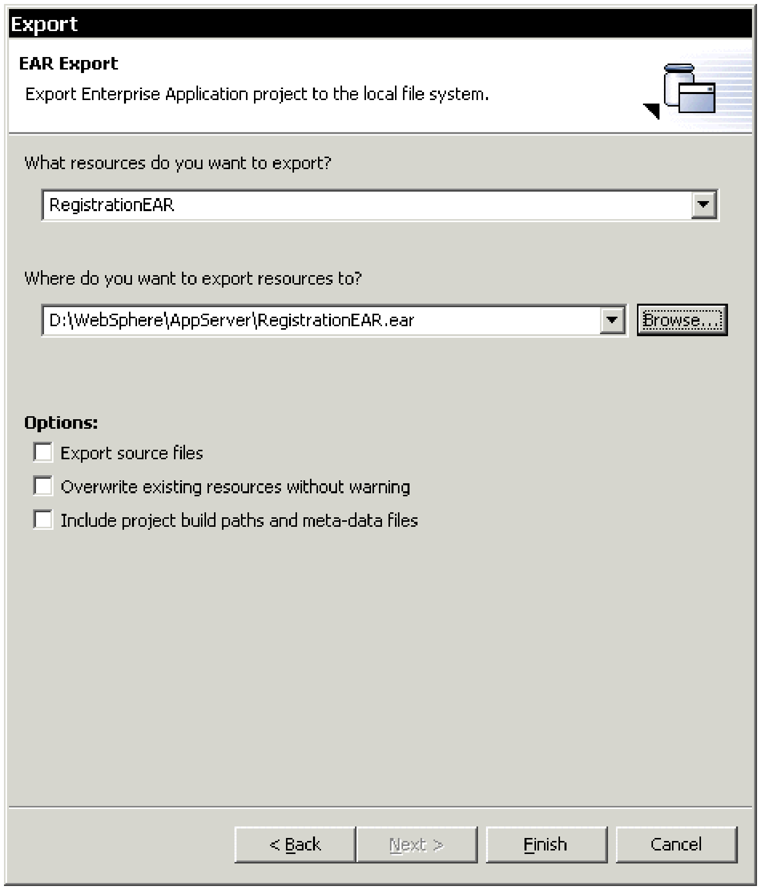
This section explains how to deploy the application manually to WebSphere AEs.

12.1.1 Exporting your project from Application Developer

Complete the following steps to export an EAR file:

1. From the File menu, select **Export.**
2. Select the **EAR file export** wizard and click **Next**.

The window as shown in Figure 12-1 will appear.


Export

EAR Export

Export Enterprise Application project to the local file system.

What resources do you want to export?

RegistrationEAR

Where do you want to export resources to?

D:\WebSphere\AppServer\RegistrationEAR.ear

Options:

- [ ] Export source files
- [ ] Overwrite existing resources without warning
- [ ] Include project build paths and meta-data files

Figure 12-1 Exporting a project EAR file

In this window you can select the EAR project that you want to export, as well as select the location where you want to export.

You can further specify additional options as appropriate using the options check boxes. To export the source files, select the **Export source files** check box. If you are exporting to an existing file, and you do not want to be warned about overwriting it, select the **Overwrite existing files without warning** check box. Click **Finish**.

12.1.2 Installing the EAR file on WebSphere AEs

This topic discusses how to install the EAR file on WebSphere.

After exporting the EAR file, FTP or copy the EAR file to the `../WebSphere/AppServer/installableApps` directory in the WebSphere AEs product directory structure.

The EAR file can be installed by either using the `WebSphere Admin Console` or manually with the command line option `SEAppInstall`. An example of manual installation for the registration would be:

```
SEAppInstall -install ../installableApps/Registration.ear
```

12.1.3 Starting the WebSphere AEs Admin Console

Start the application admin server by selecting **Start—>Programs—>IBM WebSphere Application Server V4.0 AES—>Start Admin Server**, on a Windows machine, or use the startServer.sh script on a UNIX machine.

The server is started when the following message is shown in the console:

```
SPL0057I: The server Default Server is open for e-business.
Please review the server log files for additional information.
Standard output: C:\WebSphere\AppServer/logs/default_server_stdout.log
Standard error: C:\WebSphere\AppServer/logs/default_server_stderr.log
```

Once you have started the Application Server, start the administrator's console by one of the two methods. Select **Start—>Programs—>IBM WebSphere Application Server V4.0 AES—>Administrator's Console**, on a Windows NT machine, or by accessing it from a browser by typing:
`http://localhost:9090/admin/`

Enter a user ID in the window, and click the **Submit** button.

Note: This user ID does not need to be a valid user ID on the system. It is only used only for tracking user-specific changes to the configuration data.

12.1.4 Installing the EAR

This topic discusses how to install an EAR file in WebSphere AEs using the admin console. During this task, you will install the application files (.ear, .jar, and .war).

To install an application:

1. Expand the tree on the left side of the console to locate **Nodes—>hostname—>Enterprise Applications**.
2. Select **Enterprise Applications**.

   The right side of the console should display the list of zero or more installed applications (as .ear files).
3. Click the **Install** button displayed on the right side of the console, above the list of installed applications.

   Here you need to specify the location of the application or module, referring to the application property reference as needed to fill in the field values, and the application name.

   > **Note:** Use the first set of fields if the console and application files are on the same machine (whether or not the server is on that machine, too). Use the second set of fields (including remote path) if the application files already reside on the same machine as the server, and the console is being run from a remote machine. See the field help for further discussion of the remote file path.

4. Click **Next**.
5. On the next page select **Virtual Host Name**.

   Follow the instructions on the resulting Task wizard. Depending on the components in your application, various panels will be displayed:

   – Modify `Role to User Mapping` (valid for all applications).

   – Modify `EJB Run as Role to User Mapping` (valid for applications containing EJB modules with one or more entity beans that use the IBM deployment descriptor extension for `Run As Settings, Run As Mode, Run As Specified Identity`).

   – Modify `EJB to JNDI Name Mapping` (valid for applications containing EJB modules).

   – Modify `EJB Reference to JNDI Name Mapping` (valid for applications containing EJB modules with EJB references in their deployment descriptors).

- Modify `Resource Reference to JNDI Name Mapping` (valid for applications containing Web modules with resource references in their deployment descriptors).
- Specify `Virtual Host Mapping` (valid for applications containing Web modules) and includes JSP precompilation option.
- Specify `CMP Data Source Binding` (valid for applications containing EJB modules with container managed entity beans).

6. Click **Finish** when you have completed the wizard.

Note: Be patient if the application you are installing contains EJB modules for which code must be generated for deployment. This step can take a while.

7. Verify that the new application is displayed in the tree on the left side of the console. It should be located at **Nodes—>hostname—>Enterprise Applications**.
8. To save your configuration click the **Save** button.
9. Select the application and click the **Start** button.
10. (Optional) To have the configuration take effect:
    - Stop the application server.
    - Start the application server again.
    - Stop the HTTP Web server.
    - Start the HTTP Web server again.

Note: If you installed an application while the server was running, the newly installed application and its modules can be viewed in the list of installed Enterprise applications, from which applications and their modules can be started, stopped, and restarted. However, the application and modules will remain in a `cannot be run` state until the server is stopped and started again.

12.1.5 Testing the application

You can now test your application on the remote WebSphere Application Server. To do so, start the Sample application start page (index) by typing **`http://localhost:9080/Registration/`** in your browser.

12.2 Publishing to a remote server (AEs)

Prerequisites: With Application Developer, you can deploy the application to a remote server using a remote server instance and configuration. If you want to publish your projects remotely on WebSphere Application Server AEs, you must install the following applications on the remote server:

- IBM WebSphere Application Server Advanced Single Server Edition for multiplatforms
- IBM Agent Controller
- (Optional) FTP server

Important: Ensure the IBM Agent Controller is running on the remote server and is inside a firewall.

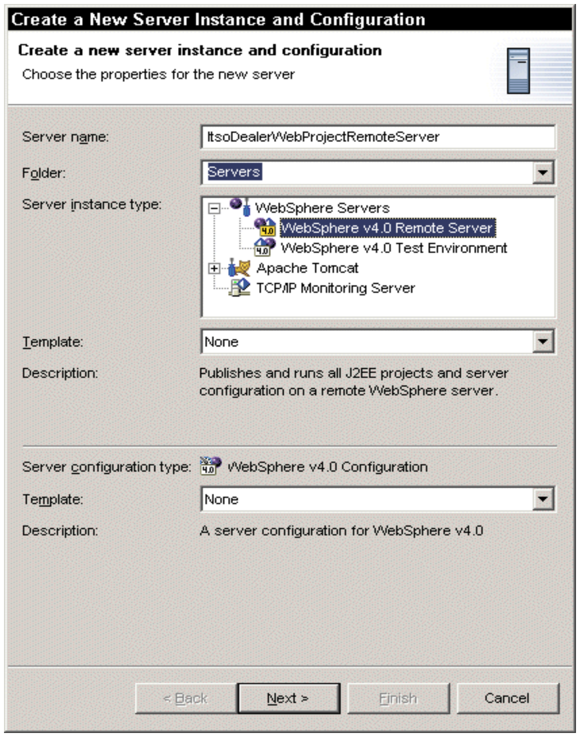
Tip: In a team environment, care must be taken when publishing code to a remote server. Make sure the members of the team do not step on each other when deploying. It is a good idea to assign the deployment process to a lead developer to ensure correct code versions are deployed. This document does not define the details of the application build process.

12.2.1 Creating a remote server instance

1. From the Server view of the server perspective, create the server configuration for the remote WebSphere AEs Server. Select **New—>Server Instance** or **New—>Server Instance and configuration**.

   On the first page of the Create a New Server Instance wizard or the Create a New Server Instance and Server Configuration wizard (Figure 12-2):

   – In the Server Name field, type a name for the new server.
   – In the Folder field, enter a folder name for the server.
2. Select **WebSphere Remote Server** as the instance type. The Next button is enabled allowing you to specify additional remote server information needed to transfer files remotely.
3. Click **Next**.


Create a New Server Instance and Configuration

Create a new server instance and configuration

Choose the properties for the new server

Server name: ItsoDealerWebProjectRemoteServer

Folder: Servers

Server instance type:
- WebSphere Servers
  - WebSphere v4.0 Remote Server
  - WebSphere v4.0 Test Environment
- Apache Tomcat
- TCP/IP Monitoring Server

Template: None

Description: Publishes and runs all J2EE projects and server configuration on a remote WebSphere server.

Server configuration type: WebSphere v4.0 Configuration

Template: None

Description: A server configuration for WebSphere v4.0

Figure 12-2 Creating a remote server instance

4. The second page of the server creation wizard opens allowing you to provide additional remote server instance information required for using the WebSphere Application Server remotely (Figure 12-3).

 Note: All paths on this page are seen from the remote machine.

 – In the Host address field, type the fully qualified DNS name or the IP address of the remote machine that WebSphere Application Server is running on. The field is pre-filled with the default address for the local host (127.0.0.1).

 Tip: For more information about any of the fields on this and other wizards, select the field and then press F1.

 – In the WebSphere installation directory field, type the path where you installed WebSphere Application Server on the remote machine. This path is the same as the WAS_ROOT path mappings as defined by the WebSphere server configuration.

If you have installed WebSphere Application Server in the default directory, use **`c:/WebSphere/AppServer`** directory as the WebSphere installation path.

If you select the `Use default WebSphere deployment directory` check box, then you want to use the default WebSphere deployment directory. The WebSphere deployment directory field is then pre-filled with the default value. Otherwise, in the WebSphere deployment director**y** field, type the path of the directory where the Web application and server configurations will be published. This directory is any existing directory that can be seen from the remote server machine.

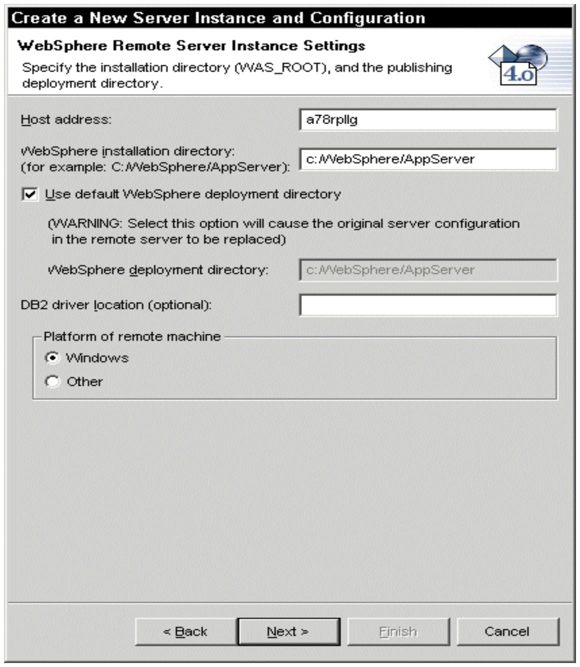
If the directory E:/testdir resides on the remote machine, then type **`E:/testdir`** in this field.

If you are following WebSphere naming conventions and install WebSphere Application Server in the C:/WebSphere/AppServer directory, then the WebSphere deployment directory is C:/WebSphere/AppServer.

Note: When publishing to the remote server, the server configuration and the Web application will be published to a directory under the remote deployment directory called config and installedApps respectively.

If you select the `Use default WebSphere deployment directory` check box when creating a remote server instance, and then publish using this instance, the default WebSphere Application Server server-cfg.xml file and plugin-cfg.xml files are replaced with the published version

Optional: In the DB2 driver location field, type the DB2 location where the DB2 classes reside in the remote machine. If the default value is set in the Preference WebSphere page, this field is pre-filled with the DB2 location.



Create a New Server Instance and Configuration

WebSphere Remote Server Instance Settings

Specify the installation directory (WAS_ROOT), and the publishing deployment directory.

Host address: a78rpllg

WebSphere installation directory: (for example: C:/WebSphere/AppServer): c:/WebSphere/AppServer

[✓] Use default WebSphere deployment directory

(WARNING: Select this option will cause the original server configuration in the remote server to be replaced)

WebSphere deployment directory: c:/WebSphere/AppServer

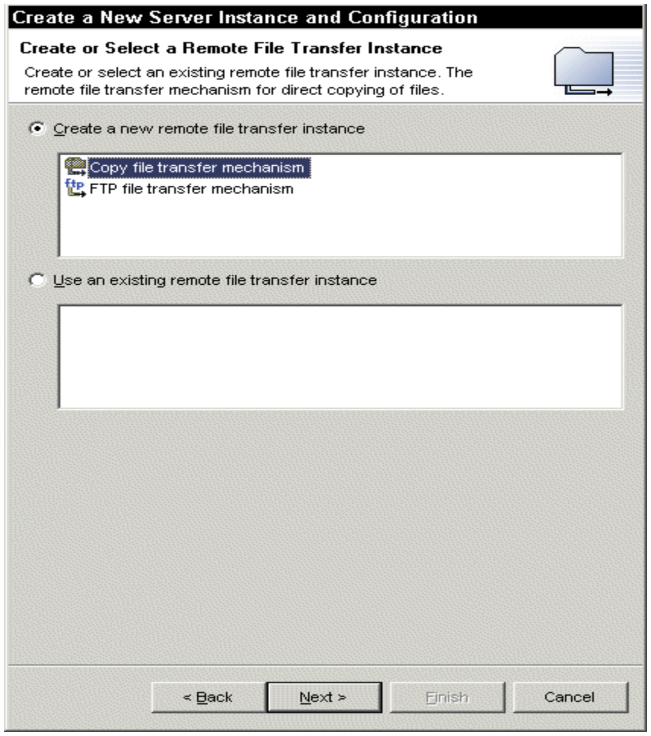
DB2 driver location (optional):

Platform of remote machine: [● Windows] ○ Other

Figure 12-3 Remote server instance settings

5. Click **Next** again to display the third page of the server creation wizard.

This page allows you define a remote file transfer instance. A *remote file transfer instance* contains information for transferring Web applications and server configurations to the remote server during publishing (Figure 12-4).


Create a New Server Instance and Configuration

Create or Select a Remote File Transfer Instance

Create or select an existing remote file transfer instance. The remote file transfer mechanism for direct copying of files.

(●) Create a new remote file transfer instance

- Copy file transfer mechanism
- FTP file transfer mechanism

( ) Use an existing remote file transfer instance

Figure 12-4 Remote file transfer option

6. Select one of the following radio buttons:
   – Create a new remote file transfer instance, defines a new set of parameters and environment settings needed to transfer files remotely.
   – Use an existing remote file transfer instance, lists the already defined remote file transfer instances that you can use for transferring files remotely.
7. If Create a new remote file transfer instance radio button is selected:
   – From the list, select one of the following:
      • **Copy file transfer mechanism**, to copy resources directly from one machine to another in the file system.
      • **FTP file transfer mechanism**, to copy resources from one machine to another using File Transfer Protocol (FTP).
8. Click **Next** to display the fourth page of the server creation wizard.

If **Use an existing remote file transfer instance** radio button is selected, the Next button is not enabled (Figure 12-5).

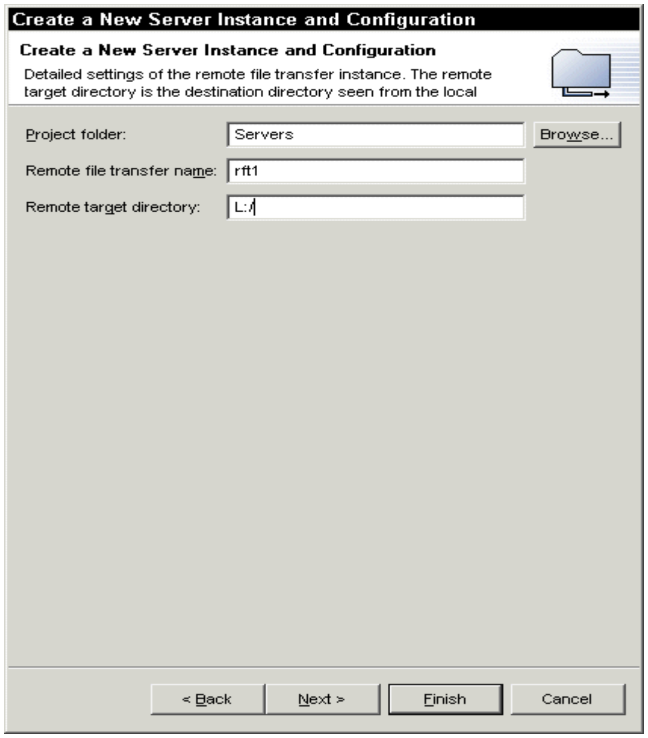
– Select the remote file transfer instance that you want to use to transfer files remotely. Omit the next step.

If **Copy file transfer mechanism** is selected, the next page of the wizard appears (Figure 12-5).

– In the Project folder field, type the name of the project folder where the remote file transfer instance will reside.

– In the Remote file transfer name field, type the name of the remote file transfer instance.

Note: For more information about any of the fields on this and other wizards, select the field, and then press F1.


Create a New Server Instance and Configuration

Create a New Server Instance and Configuration

Detailed settings of the remote file transfer instance. The remote target directory is the destination directory seen from the local

Project folder: Servers

Remote file transfer name: rft1

Remote target directory: L:/

Figure 12-5 Remote copy options

- In the Remote target directory field, type the remote target directory where you want your applications and server configuration published. This remote target directory is the one seen by the local machine. If WebSphere Application Server is installed on a different machine, then the remote target directory is the network drive that maps to the WebSphere deployment directory. If WebSphere Application Server is installed on the same machine as the workbench, then the remote target directory should be the same as the contents in the WebSphere deployment directory field.

If the remote Server is on the local machine, then C:/WebSphere/AppServer is the directory. If the remote Server is on a remote machine, then we use the mapped drive name for the shared directory (mapped as L:/ in our example).

The Next button is enabled only if you are creating a new server instance and server configuration together.

9. Click **Next** if you want to change the HTTP port number.
10. Click **Finish** to create a remote file transfer instance and a remote server instance. These instances will reside locally on your machine. The server instances appear in the Server Configuration view. The remote file transfer instances appears in the Navigator view.

If **FTP file transfer mechanism** is selected, the next page of the wizard appears.

- In the Project folder field, type the name of the project folder where the remote file transfer instance will reside.
- In the Remote file transfer name field, type the name of the remote file transfer instance.
- In the Remote target director**y** field, type the remote target directory where you want your application and server configuration published. This remote target directory points to the WebSphere deployment directory that is seen from the workbench using the FTP client program.

  If the WebSphere deployment directory is C:/WebSphere/AppServer and your FTP server route directory is C:/, then your remote target directory is /WebSphere/AppServer.

  > **Note:** To determine whether a beginning slash is required, log on to the FTP server using a FTP client program, and then type the `pwd` command. If the results containing the default log on directory begins with a slash, then a slash is required prior to typing the remote target directory in the Remote target directory field.

- In the FTP URL field, type the URL that is used to access the FTP server.
- In the User login field and the User password field, type the FTP user ID and password, which will be used to access the FTP server.

- In the Connection timeout field, type the time (in milliseconds) that the workbench will wait this long while attempting to contact the FTP server before timing out.
- Select the **Use PASV Mode (Passive Mode) to go through the firewall** check box, if you want to pass through a firewall provided that one is installed between your FTP server and the workbench.
- Select the **Use Firewall** check box, if you want to use the firewall options.
- To change the firewall options, select the **Use Firewall** check box, and then click **Firewall Settings**.
- The Next button is enabled only if you are creating a new server instance and server configuration together. Click **Next** if you want to change the HTTP port number.

11. Click **Finish** to create a remote file transfer instance and a remote server instance. These instances will reside locally on your machine. The server instances appear in the Server Configuration view. The remote file transfer instances appears in the Navigator view.
12. Add the project to the new server configuration.
13. Make sure the project is removed from the other server configurations.

12.2.2 Publishing to remote server

1. Select **Run On Server** from the context menu of Application Entry view of the server perspective.

On the remote machine you will see the new directory added to the installedApps directory.

12.2.3 Testing the application

You can now test your application on the remote WebSphere Application Server.

Part 4

Appendixes

A

Installing WebSphere Studio Application Developer

In this appendix we describe how to install WebSphere Studio Application Developer. The installation instructions specifically apply to Version 5, but should be very similar for more recent versions of the product.

Things to do before installation

Before you proceed with installation, please verify that your hardware and software configuration meets the following prerequisites:

- Windows 2000, Windows ME, Windows 98, or Windows NT 4.0 with Service Pack 6a or higher
- Microsoft Internet Explorer 5.5 or higher
- TCP/IP installed and configured
- A mouse or alternative pointing device
- Pentium II processor or higher recommended
- SVGA (800x600) display or higher (1024x768 recommended)
- 256 MB RAM minimum (512 MB recommended)
- A minimum of 400 MB free disk space based on NTFS. Actual disk space on FAT depends on hard disk size and partitioning

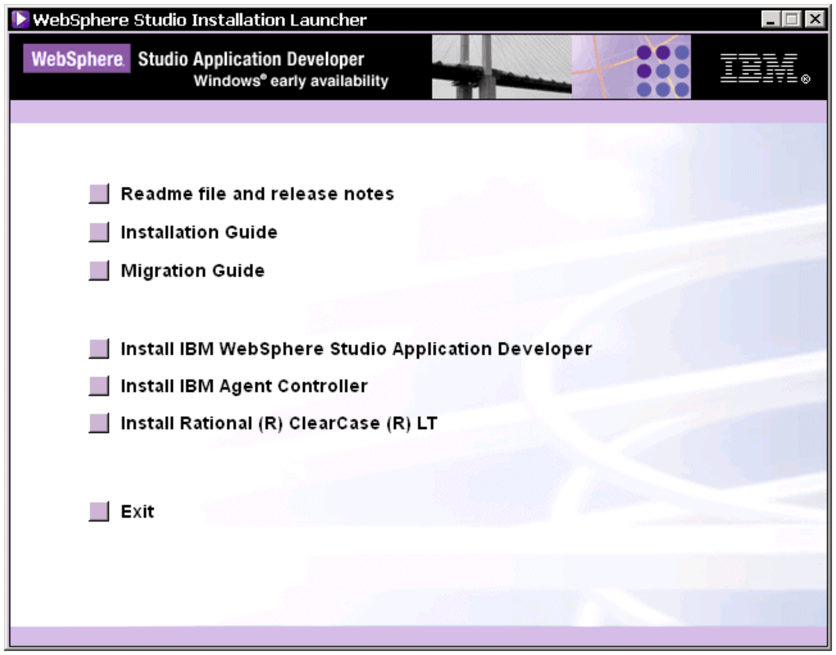
Also please check the following before beginning the install:

- In addition to the disk space requirements for the product, you need to have at least 50 MB of space available on your Windows system drive and the TMP or TEMP environment variable must be pointing to a directory with at least 10 MB free.
- If you have the IBM HTTP Server or WebSphere Application Server running, they need to be shut down.
- The Services window should not be open. If it is, the Remote Agent Controller cannot be installed.

Note: If you have VisualAge for Java or any version of WebSphere Studio already installed, there is no need to un-install before installing Application Developer.

Installing Application Developer

Start the installation by inserting IBM WebSphere Studio Application Developer Early Availability Version 5 CD-ROM, then select **Install IBM WebSphere Studio Application Developer** (Figure 12-6) or running SETUP.EXE from the directory IBM_WS_Application_Developer.


WebSphere Studio Installation Launcher
WebSphere Studio Application Developer
Windows® early availability
IBM®

Readme file and release notes

Installation Guide

Migration Guide

Install IBM WebSphere Studio Application Developer

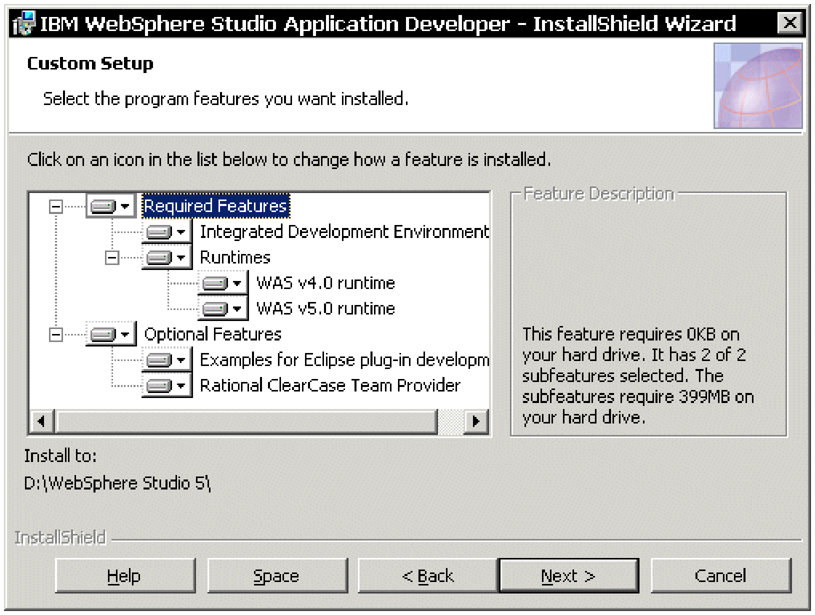
Install IBM Agent Controller

Install Rational (R) ClearCase (R) LT

Exit

Figure 12-6 WebSphere Studio Application Developer installation

After the first couple of panels where you have to accept license information and choose a directory where to install Application Developer, you will be promoted to select installation components (Figure 12-7).


IBM WebSphere Studio Application Developer - InstallShield Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- Required Features
  - Integrated Development Environment
  - Runtimes
    - WAS v4.0 runtime
    - WAS v5.0 runtime
- Optional Features
  - Examples for Eclipse plug-in developm
  - Rational ClearCase Team Provider

Feature Description

This feature requires 0KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 399MB on your hard drive.

Install to:
D:\WebSphere Studio 5\

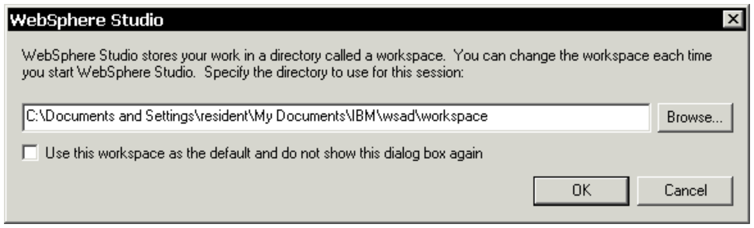
Figure 12-7 Select options

On this page you select what components you will use. Options are, plug-in development samples and ClearCase plug-in. Select options, depends on your needs.

Click **Install** on the final page to start the installation.

Selecting your workspace

You need to create your workspace when you launch Application Developer (Figure 12-8). By clicking the check box, the workspace will be the default.


WebSphere Studio

WebSphere Studio stores your work in a directory called a workspace. You can change the workspace each time you start WebSphere Studio. Specify the directory to use for this session:

C:\Documents and Settings\resident\My Documents\IBM\wsad\workspace

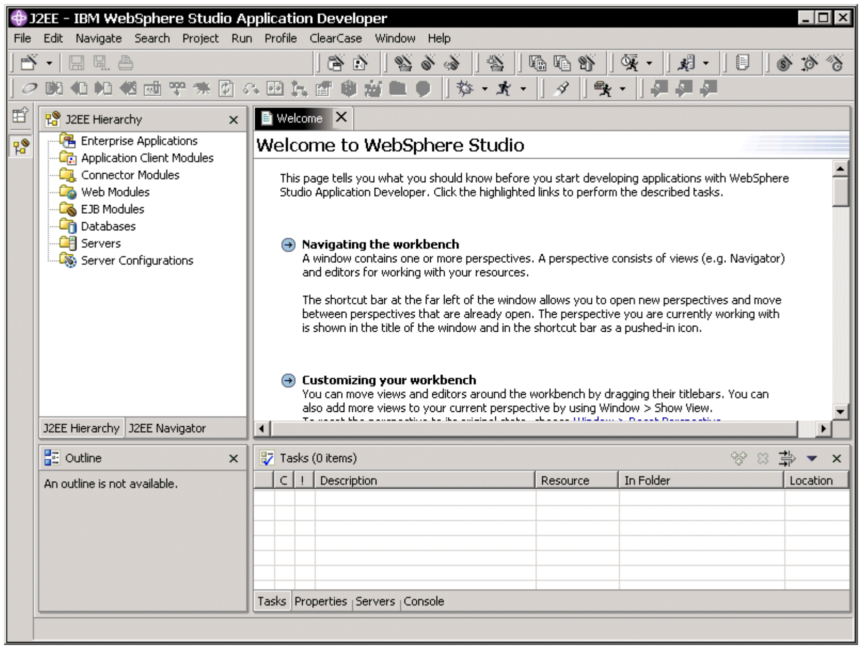
☐ Use this workspace as the default and do not show this dialog box again

Figure 12-8 Creating the Workspace

Verifying the installation

Once the installation program has run, click the Windows **Start** button and select **Programs—>IBM WebSphere Studio Application Developer—>IBM WebSphere Studio Application Developer.**

If the installation has worked correctly, you should see your default (J2EE) perspective with the Application Developer welcome page (Figure 12-9).



Welcome to WebSphere Studio

This page tells you what you should know before you start developing applications with WebSphere Studio Application Developer. Click the highlighted links to perform the described tasks.

Navigating the workbench
A window contains one or more perspectives. A perspective consists of views (e.g. Navigator) and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open new perspectives and move between perspectives that are already open. The perspective you are currently working with is shown in the title of the window and in the shortcut bar as a pushed-in icon.

Customizing your workbench
You can move views and editors around the workbench by dragging their titlebars. You can also add more views to your current perspective by using Window > Show View.

An outline is not available.

Figure 12-9 Application Developer J2EE perspective (welcome)

B

Installing IBM WebSphere Application Server 4.0 AEs

In this appendix we describe how to install IBM WebSphere Application Server 4.0 Advanced Edition Single Server (AEs).

Things to do before installation

Prior to installing AEs, the following checks and tasks need to be completed on the WebSphere Server machine:

1. Check hardware and software prerequisites.
2. Create groups and users.
3. Check that IP ports are unused.
4. Stop the Web server processes.

Hardware and software prerequisites

AEs has the following hardware and software requirements:

- Hardware:
  - 180 MB diskspace (minimum) for AEs
  - 50 MB diskspace (minimum) for IBM HTTP Server
  - 135 MB diskspace (minimum) for TEMP directory
  - 500Mhz Pentium III or above
  - 384 MB RAM minimum, 512 MB recommended
  - Ethernet or token ring card
  - CD-ROM drive
  - Network connectivity to the Internet
- Software:
  - Microsoft Windows 2000 Server, SP 1 or 2 *or* Microsoft Windows NT Server 4.0 SP 6a
  - IBM HTTP Server 1.3.19

    (IBM HTTP Server is included with AEs and will be installed if not already present.)

Create groups and users

To create the required groups and users, perform the following steps:

1. If you have not already done so, create a Windows 2000 user under which the WebSphere service will be run, as follows:
   - Locally defined (not a member of a Windows domain)
   - Member of administrators group.

   You can create local users and assign group memberships by clicking **Control Panel—>Administrative Tools—>Computer Management —>System Tools —>Local Users and Groups.**
2. Assign the following rights to this user:

  - Act as part of the Operating System
  - log on as a service

  You can assign user rights by clicking **Control Panel—> Administrative Tools —>Local Security Policy—>Local Policies—>User Rights Assignment**.

Tip: We suggest creating the user wsadmin.

Check that IP ports are unused

To check that the required ports are not in use, perform the following steps:

1. Check that there are no existing active services that use the following IP ports on the server:
   - 900 (bootstrap port)
   - 9000 (location service daemon)
   - 9080 (default application server).

   Use the following command for this task: `C:\> netstat -an`

Stop the Web server processes

The IBM HTTP Server process must be stopped while AEs is being installed. The installation changes the httpd.conf configuration file as part of the Web server plug-in component installation.

- Issue the command: `C:\> net stop "IBM HTTP Server"` or stop the service under **Control Panel—>Administrative Tools—>Services**.

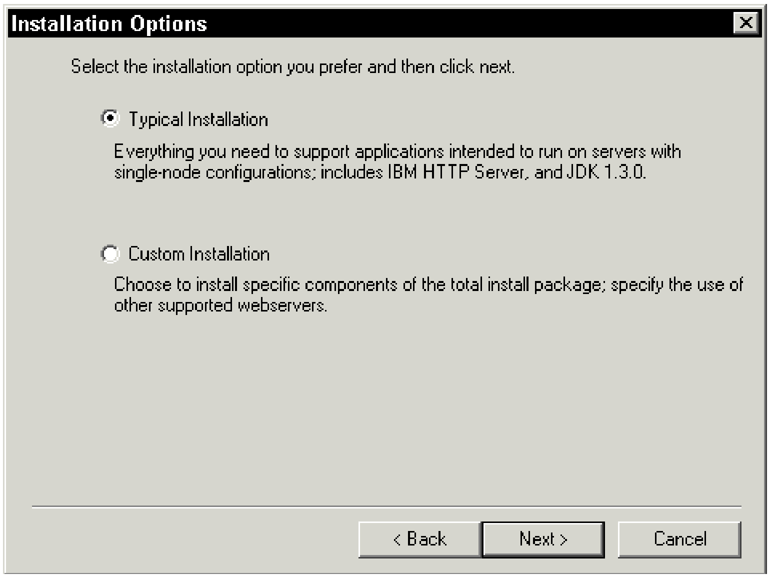
Install WebSphere Application Server

To install AEs using the GUI installer interface, complete the following steps on the WebSphere server machine:

Tip: The WebSphere installer (setup.exe) also provides a non-GUI, scripted or silent mode of operation. See product documentation for details.

1. Log on with a user ID that has administrator rights to the local server domain.
2. Insert the AEs CD.
3. Start the AEs installation by double-clicking **Setup** from the root of the CD.

4. In the Choose Setup Language window, select your national language from the drop-down menu (English is selected by default) and click **OK**.
5. In the WebSphere Application Server Attention window, read the warnings and then click **Next** to continue.
6. In the **I**nstallation Options window shown in Figure 12-10, select **Typical Installation**, and then click **Next**. If you have some special requirements, you may want to select **Custom Installation** instead. This will give you the option to deselect certain options, for example, the IBM HTTP Server and the JDK.


Installation Options

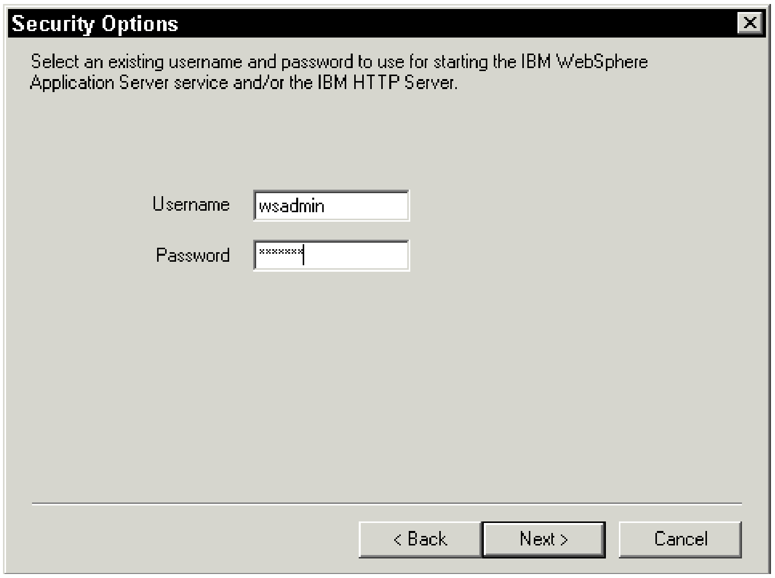
Select the installation option you prefer and then click next.

[✓ Typical Installation]
Everything you need to support applications intended to run on servers with single-node configurations; includes IBM HTTP Server, and JDK 1.3.0.

Custom Installation
Choose to install specific components of the total install package; specify the use of other supported webservers.

Figure 12-10 Installation options

7. In the Security Options window (shown in Figure 12-11) enter the user name and password of the Windows account under which AEs is to run as a service. This is the account created during the WebSphere pre-installation tasks. Click **Next** to continue.


Security Options

Select an existing username and password to use for starting the IBM WebSphere Application Server service and/or the IBM HTTP Server.

Username: wsadmin

Password: *******

Figure 12-11 Security options

8. In the Product Directory window, select the destination directory, then click **Next**.
9. In the Select Program Folder window, accept the default and click **Next**.

10.. In the Install Options Selected window, note the selected options and click **Next** to start the installation.

11. When the Setup Complete window is shown, click **Finish**.

12.. In the Restarting Windows window, select to restart the computer to complete the installation.

Verifying the installation

Perform the following tasks to verify that the installation was successful:

1. Start the WebSphere administrative server processes.
2. Start WebSphere default server.

Start the WebSphere administrative server processes: The WebSphere administrative server needs to be started in order to test the installation:

1. Start the server by selecting **Start application server** from the WebSphere menu or by entering `C:\WebSphere\AppServer\bin\startServer.bat` on a command line.

2. The startup of WebSphere administrative server was successful if the last line of the <WAS_HOME>\logs\default_server_stdout.log file is similar to the following:

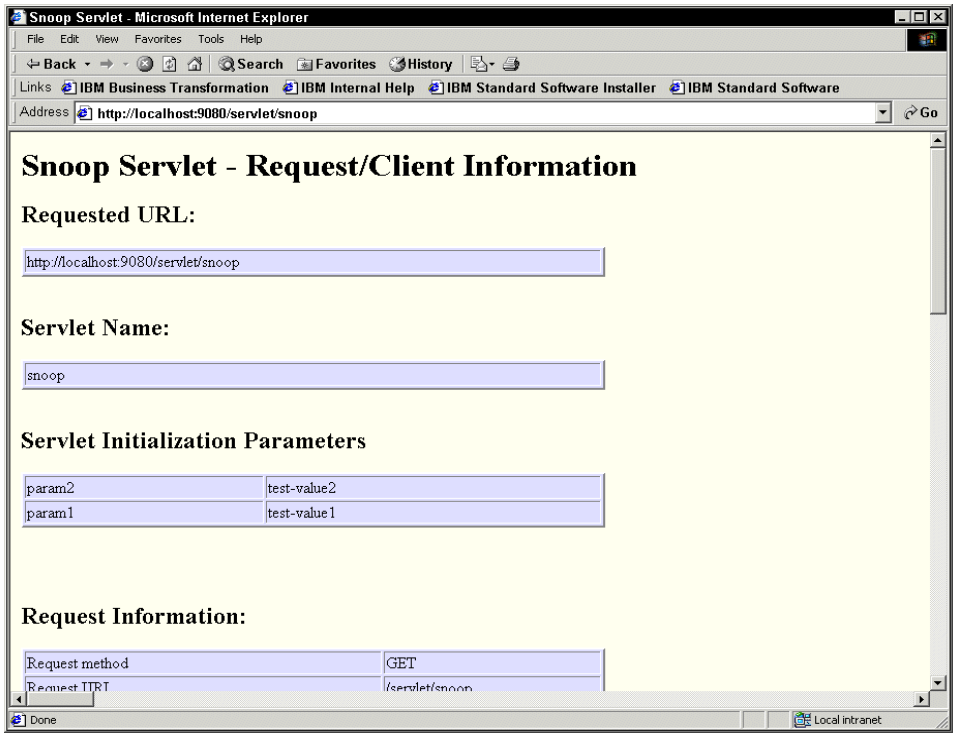
   ```
   [02.04.05 15:48:56:309 PST] 5dc79b14 Server A WSVR0023I: Server Default
   Server open for e-business
   ```

3. Verify that the default server Web container has been properly installed and configured by accessing its servlets through the Web *server embedded* within the WebSphere V4.0 Web container:

   a. Using a Web browser, request the following URL:

      ```
      http://localhost:9080/servlet/snoop
      ```

      A window similar to the one shown in Figure 12-12 should be displayed in your browser.


Snoop Servlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links IBM Business Transformation IBM Internal Help IBM Standard Software Installer IBM Standard Software
Address http://localhost:9080/servlet/snoop
Go
Done
Local intranet

Snoop Servlet - Request/Client Information

Requested URL:

http://localhost:9080/servlet/snoop

Servlet Name:

snoop

Servlet Initialization Parameters

param2	test-value2
param1	test-value1

Request Information:

Request method	GET
Request URI	/servlet/snoop

Figure 12-12 Snoop servlet accessed through embedded Web server

WebSphere Application Server AEs has now been successfully installed on your workstation.

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG246586`

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, SG24-6586.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

File name	*Description*
SG246586.zip	Zipped code samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	5 MB minimum
Operating System:	Windows
Processor:	Pentium II or higher
Memory:	256 MB above

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Each sub directory has a code to use the chapters:

workspace	Web services sample in Chapter 9
workspaceEJB	EJB sample in Chapter 10
workspaceLW	SQL to/from XML and DataSource sample in Chapter 11

Each directory is an image of workspace. You can look using explorer or import each project using project import function.

Abbreviations and acronyms

AAT	application assembly tool
ACL	access control list
API	application programming interface
BLOB	binary large object
BMP	bean-managed persistence
CCF	Common Connector Framework
CICS	Customer Information Control System
CMP	container-managed persistence
CORBA	Component Object Request Broker Architecture
DBMS	database management system
DCOM	Distributed Component Object Model
DDL	data definition language
DLL	dynamic link library
DML	data manipulation language
DOM	document object model
DTD	document type description
EAB	Enterprise Access Builder
EAI	Enterprise Application Registration
EAR	Enterprise archive
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
FTP	File Transfer Protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAF	Java Activation Framework
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSDK	Java Servlet Development Kit
JSP	Java server page
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MFS	message format services
MVC	model-view-controller
OLT	object level trace
OMG	Object Management Group
OO	object-oriented

OTS	object transaction service
RAD	rapid application development
RDBMS	relational database management system
RMI	Remote Method Invocation
SAX	Simple API for XML
SCCI	source control control interface
SCM	software configuration management
SCMS	source code management systems
SDK	Software Development Kit
SMR	Service Mapping Registry
SOAP	Simple Object Access Protocol (a.k.a. Service Oriented Architecture Protocol)
SPB	Stored Procedure Builder
SQL	structured query language
SRP	Service Registry Proxy
SSL	secure socket layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UCM	Unified Change Management
UDB	Universal Database
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
UOW	unit-of-work
URL	uniform resource locator
VCE	visual composition editor
VXML	voice extensible markup language
WAR	Web application archive
WAS	WebSphere Application Server
WML	Wireless Markup Language
WS	Web service
WSBCC	WebSphere Business Components Composer
WSDL	Web Service Description Language
WSTK	Web Service Development Kit
WTE	WebSphere Test Environment
WWW	World Wide Web
XMI	XML metadata interchange
XML	eXtensible Markup Language
XSD	XML Schema definition

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 316.

- *Web Services Wizardry with WebSphere Studio Application Developer,* SG24-6292-00
- *Self-Study Guide: WebSphere Studio Application Developer and Web Services ,* SG24-6407-00
- *WebSphere Studio Application Developer Programming Guide,* SG24-6585-00
- *DB2 UDB e-business Guide,* SG24-6539-00

Other resources

These publications are also relevant as further information sources:

- *Java and XSLT, O'Reilly,* 0-596-00143-6

Referenced Web sites

These Web sites are also relevant as further information sources:

- Developing XML Web services with WebSphere Studio Application Developer:

  `http://www.research.ibm.com/journal/sj/412/lau.pdf`
- IBM WebSphere Developer Technical Journal:

  `http://www7b.boulder.ibm.com/wsdd/techjournal/`
- XML Bible, Second Edition:

  `http://www.ibiblio.org/xml/books/bible2/chapters/ch20.html#d1e519`

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

A

Access Beans 245
Administrative Tools 306
AEs 77, 305
AlphaWorks 13
Animated GIF Designer 73
Apache Tomcat 78
Application Server 71
Apply XSL 96
Attribute 182
Audio Image Video (AIV) Extender 126

C

ClearCase LT 76
CMP 239
Cocoon 49
Compiling XSL 191
Concurrent Versions System 76
Conditional (XQuery) 43
Container Managed Persistence 175
Content Model 182
Copy file transfer mechanism 292
CSS 21
Customer Relation Management 6

D

DAD 131, 171, 196
DAD Extension 199
DAD extension 98
DADX 164, 171
DADX Group 200
datasource 278
DB2 driver location 290
DB2 XML Extender 101, 126, 164, 195
DB2 XML extender 98
DDL to XML Schema wizard 122–123
Developer Resource Portal 72
Diskspace 306
DOM 19, 48, 57, 67, 165, 175, 218, 228, 250
DOM Level2 62
DRP 72
DSA 38
DTD 18, 74, 84, 115
DTD Editor 83
DTD Repository 130
DTD to XSD 146
DTD/XSD from XML 149
DXX 200
dxxGenXMLClob 204
DxxInvoker 200

E

ebXML 6
Edit Join Conditions 197
EJB 71, 238
EJB to RDB Mapping 241
Element 42, 110
EncryptedData 36
Enterprise 72
Enterprise application archive 76
Enterprise Architecture Integration 6
Enterprise Developer 72
Enterprise Generation Language 72
Enterprise Java Beans 172
Entity EJB 239
Event-based parsing 19
Extensible Stylesheet Language 11, 15

F

Filtering 44
FLWR 43
Foreign key as links 109
FTP file transfer mechanism 292

G

Generate DADX 202
Generate DTD 194
Generate query template file 117
Generate XML 184
Generate XML Schema 122
Generating XSL from XHTML 188
Genrrate XSL 186
Global Element 181
GML 9

group.properties 201

H
HTML and mapping approach 184
HTML from XSD 151
HTTPRequest 265

I
IBM DB2 XML Extender 75
IBM WebSphere Studio Application Developer Early Availability Version 5 300
Install 302
Installation Options 308
Installing EAR 285
Integration Edtion 72
Introspect 223

J
J2EE 71, 180, 216, 303
J2EE 1.2 Complient 220
Java API for XML Processing 65
Java Architecture for XML Binding 49
Java beans from DTD/XSD 152
Java Proxy 212
Java proxy 205
Java Server Pages 9
JavaScript 73
JAXP 48, 174, 190, 212, 230, 267
JDBC 56
JDBC data source 247
JNDI 176
JSP 71

M
Mapping DTDIDs to XSD Namespaces 208
Mapping XML to HTML 187

N
Namespace 88
namespace prefix 24
Namespaces 23
Navigator 103
Net Search Extender 126

P
Plug-in Development Environment 78
Primary key as attributes 106
produceDOMDocument 228, 265
produceDOMSource method 259

Q
Quantifiers 44
Query Template 117
Querying Relational Data 45
QueryProperty 270

R
RDB to XML Mapping 196
RDB_node mapping 137, 140
Recurse though foreign key 109
Recurse through foreign keys 104
Redbooks Web site 316
  Contact us xiii
Remote Server 295
Remote Server Instance 288

S
SAML 40
SAX 19, 48, 57, 67
SAX2 58
Scalable Vector Graphics 7
Schema 263
Schema Files 237
Security Options 308
Servlet 71, 175
setup.exe 307
SGML 9
SHA-1 38
Show table columns as 103, 118
Show table columns as 'Elements' 104
Signature 40
Simple Object Access Protocol 13, 76
Simple Project 179
SOAP 164, 172
SQL mapping 137
SQL to XML wizards 102
SQL to/from XML 258
SqlProperties 264
SQLResult 112, 262, 266
SQLToXML 270
Standard Generalized Markup Language 9
Start application server 309
Stylesheet Compilation 68

SVG 50

T
Testing the XSL 189
Text Extender 126
Transformer 192
Transforms 38
TrAX 48
Tree-based parsing 18

U
UDF 200
UDT 200
Uniform Resource Identifier 24
Universal Description Discovery and Integration 13, 76
URI 24
Use default WebSphere deployment directory 290
User-Defined Function 127
User-Defined Type 126

V
Validating DTD 85
Validating schema 88
Validating the XML file 83
VBScript 73
VisualAge for Java 70
Voice XML 7

W
WAS_V4_XALAN 190
Web application archive 76
Web Art Designer 73
Web Project 180
Web Services 13, 71, 98, 165, 177–178, 193, 208
Web Services Description Language 13, 76, 166
Web Services Object Runtime Framework 171
Web Tier 221, 237
WebSphere administrative server 309
WebSphere Application Server Attention 308
WebSphere Studio Asset Analyzer 72
WebSphere Test Environment 253
Workspace 302
WSAA 72

X
XACML 40
Xalan 57
Xerces 19, 57
XHTML 188
XKMS 40
XLink 25
XML 1.0 specifications 10
XML Base 28
XML Collection 134
XML Collection Operations 203
XML Column 133
XML Digital 34
XML digital signature 37
XML Document 265
XML Editor 81–82
XML encryption 36
XML from DTD 147
XML from XSD 148
XML Namespace 33, 61
XML parser 57
XML Parser for Java 19
XML Perspective 80
XML Pointer 29
XML processor 18
XML Query Language 40
XML Schema 19, 110, 175
XML Schema Editor 84–85, 180
XML security 40
XML to SQL Wizard 258
XML to SQL wizard 119
XML/XSL from Java beans 153
XMLCLOBFromFile 127
XMLToSQL 263, 267
XMLToSQL library 259
XPath 13, 33, 92
XPointer
  path 30
  Range 31
  String 32
XSD Editor 85
XSD syntax 181
XSD to DTD 146
XSL Debugger 96
XSL Editor 89–90
xsl variable 269
XSL/XML /from HTML 156
XSL-FO 21, 50
XSLServlet 217
XSLT 20, 47, 56, 75, 167, 178, 216
XSLT Processor 259

XSLTC 48, 65

IBM

Redbooks

The XML Files: Development of XML/XSL Applications Using WebSphere Studio Version 5

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

The XML Files:

Development of XML/XSL Applications Using WebSphere Studio Version 5

Introduces WebSphere Studio Application Developer Version 5 XML tooling

A comprehensive guide to XML support of WebSphere Family

Start-to-finish application case studies

Extensible Markup Language (XML) has very quickly gathered a large number of industry supporters. Therefore, a significant number of XML-based conferences, books, Web sites, and training classes have sprung up. The book was written for those interested in designing and developing Web applications using XML and related technologies (XSL, XSLT). Project managers, architects, and developers will find this book particularly useful.

We start with an overview of XML technology. Then we explain how to apply XML technology with IBM WebSphere. Finally, we show a sample application written using the technologies described. Source code and installation steps are also available.

This IBM Redbook applies to IBM WebSphere Studio Application Developer V5.0.

SG24-6586-00 ISBN 0738426873