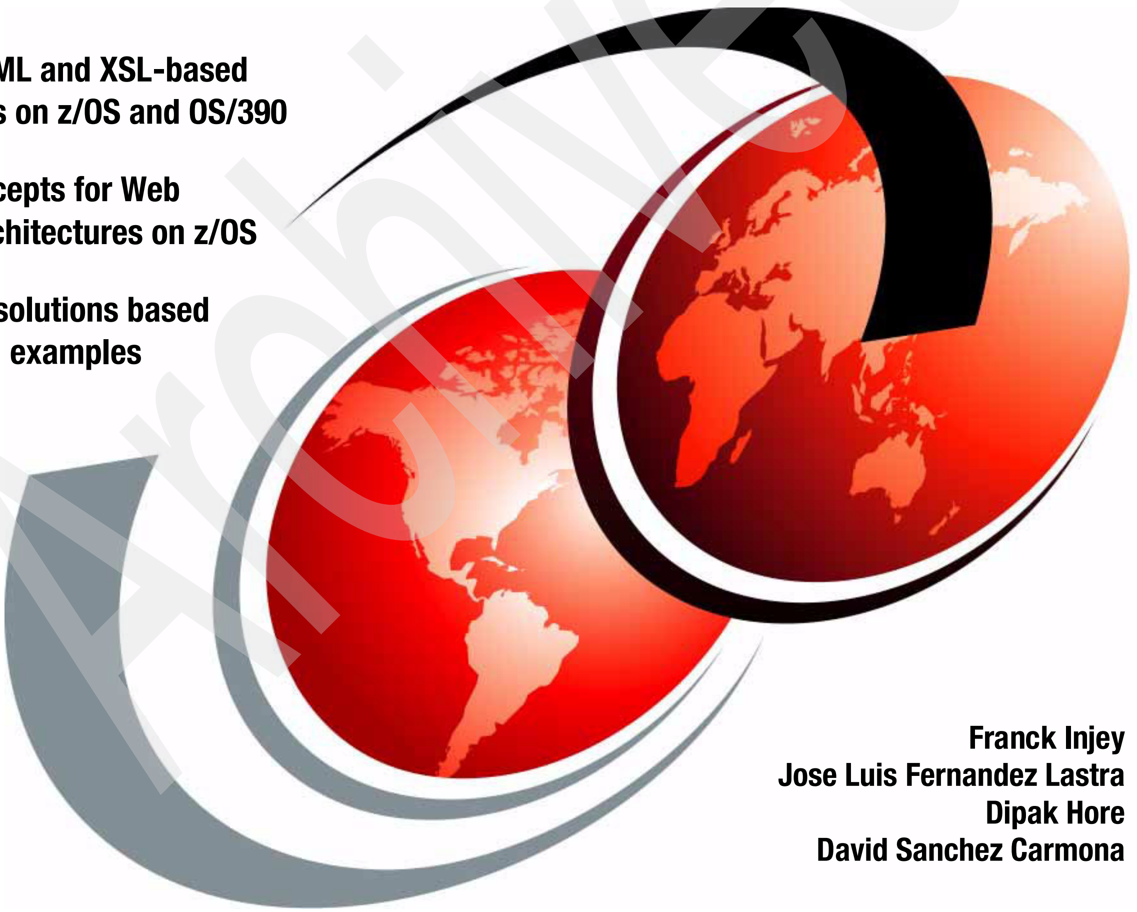
IBM

XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture

Leverage XML and XSL-based applications on z/OS and OS/390

Design concepts for Web services architectures on z/OS

Implement solutions based on practical examples



Franck Injey
Jose Luis Fernandez Lastra
Dipak Hore
David Sanchez Carmona

Redbooks

ibm.com/redbooks

International Technical Support Organization

XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture

May 2003

SG24-6826-00

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

First Edition (May 2003)

This edition applies to IBM XML Toolkit for z/OS and OS/390 V1.4, program number 5655-J51 for use with:

z/OS Version 1 Release 3 program number 5694-A01 or OS/390 Version 2 Release 10 program number 5647-A01.

WebSphere Application Server V4.0.1 for z/OS and OS/390 at Service Level 4, program number 5655-F31

Contents

Notices

Trademarks

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

AIX®
CICS®
DB2 Universal Database™
DB2®
Domino™
DRDA®
Encina®
Enterprise Systems Architecture/390®
ESCON®
FICON™
ibm.com®
IBM®
IMS™
Language Environment®
Lotus®
MQSeries®
MVS™
Net.Data®
Notes®
OS/2®
OS/390®
OS/400®
Parallel Sysplex®
PR/SM™
RACF®
Redbooks (logo) ™
Redbooks™
S/370™
S/390®
System/360™
eserver ™
VisualAge®
WebSphere®
z/OS®
zSeries®

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

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX and Pentium 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.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

This IBM Redbook describes the use of XML on IBM servers running z/OS® or OS/390®, and how it can be extended to modernize legacy applications. It provides both a high-level discussion of service-oriented architecture along with practical, detailed information about XML.

In addition to an overview of XML concepts, the first part of the book provides detailed instructions for installing the XML Toolkit for z/OS and OS/390 V1.4 and running the sample programs bundled with it. It describes how to use various tools that are part of the services development environment, details the support for XML in Enterprise COBOL, and provides an overview of the IBM WebSphere Application Server. This material is of interest mainly to system programmers and application programmers.

The second part of the book is geared more to the needs of application developers and architects. It provides a comprehensive introduction to service-oriented architecture (SOA) and Web services, and describes in detail some service-based topologies for both legacy systems and new applications. Finally, this book presents some important design concepts to enable the reader to build robust SOA-based solutions rapidly. This includes an introduction to the IBM Patterns for e-business, as well as XML-based message design, and the principles of design by contract and service design.

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, Poughkeepsie Center.

Franck Injey is a Project Leader at the International Technical Support Organization, Poughkeepsie. He has 25 years experience working on S/390® hardware and system performance. Before joining the ITSO, Franck was a Consulting IT Architect in France.

Jose Luis Fernandez Lastra is a z/OS and OS/390 Instructor in IBM Learning Services Spain. He has 4 years experience in system programming. He holds a Masters degree in Electronic Physics from UC (Universidad de Cantabria). His areas of experience include system application development and Parallel Sysplex®.

Dipak Hore is a Senior Consultant with Westpac Banking Corp.,Australia. He holds a Masters degree in Mathematics from Delhi University, India. He has more than 25 years experience in IT. His area of expertise are XML, component-based development, patterns, application architecture and e-business applications. His current focus is on use of XML in modernization of legacy systems.

David Sanchez Carmona is a z/OS and OS/390 instructor in IBM Learning Services Spain. He has 8 years of experience in the MVS™ field. He holds a Masters degree in Computing Science from UPM (Universidad Politecnica de Madrid). His areas of expertise include storage, z/OS UNIX System Services, Linux in zSeries®, Web Server and WebSphere® Application Server in z/OS.

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

Rich Conway, Tamas Vilaghy, Holger Wunderlich, Alison Chandler
International Technical Support Organization, Poughkeepsie Center

Ueli Wahli
International Technical Support Organization, San Jose Center

David Booz, Mark Dingis, Kim Johnson, Ivan Joslin, Teddy Torres
WebSphere Application Server for z/OS and OS/390, IBM Poughkeepsie

William G. Carey
zSeries System Software Design, IBM Poughkeepsie

Michael D. Connor
Enterprise Tooling, IBM Santa Teresa Laboratory

Chris Larsson
zSeries System Software Design, IBM Poughkeepsie

Gary Mazo
WebSphere zSeries Tools development, IBM Santa Teresa Laboratory

Nick Tindall
Application Development, IBM Santa Teresa Laboratory

Robin Tanenbaum
WebSphere Design and Performance Analysis, IBM Poughkeepsie

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. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

XML on z/OS and OS/390

In this part we provide a brief overview of XML concepts, and describe the XML Toolkit for z/OS and OS/390 and the distributed sample programs. We also provide an overview of XML and COBOL.

1

XML concepts

This chapter introduces basic XML concepts like DTDs, namespaces, and XML schemas.

1.1 XML introduction

The idea of universal data formats is not new. Programmers have been trying to find ways to exchange information between different computer programs for a long time. Standard Generalized Markup Language (SGML) was developed to achieve this. SGML can be used to *mark up data*, that is, to add metadata in a way that allows data to be *self-describing*. SGML is meta-language.

The markup process involves using *tags* to identify pieces of information in a document. Tags are names (strings of characters) surrounded by arrow brackets (< and >). Every piece of data that is encoded will have a start tag and an end tag, for example, <town> patiya</town>. The start and end tags make it easy for software to process the encoded information, as it clearly delineates where certain pieces of information start and where they end.

SGML does not prescribe any particular markup; instead, it defines how any markup language can be formally specified.

The most popular SGML application is HTML (Hypertext Markup Language), the markup language that rules the Web. The HTML specification is owned by W3C. However, different browser vendors introduced a number of incompatible tags to HTML, which are outside the scope of the original HTML specifications. These tags create problems for developers when they author Web pages because they must consider what browser will display the pages. And, although HTML has been very successful for displaying information on browsers, it was not found to be useful in describing the data that it represents, meaning it did not have the metadata capability that is essential for a self-describing data document.

Furthermore, SGML is quite inefficient and cumbersome when it is used to encode complex data structure. Hence, there arose a need to develop a more lightweight markup language, so W3C developed the specification for XML (eXtensible Markup Language). XML is similar to SGML in that it preserves the notion of *general markup*. There are very few optional features, and most SGML features that were deemed difficult to implement have been dropped.

1.1.1 Document-centric versus data-centric XML

There are two broad application areas of XML technologies. The first relates to document-centric applications, and the second to data-centric applications. The document-centric application outputs are primarily meant for human consumption. Some examples of such documents are legal briefs, manuals, product catalogs, and so forth. The key element of these documents is semi-structured marked-up text.

Data-centric XML is used to mark up highly structured information such as data structures in a programming language, relational data from databases, financial transactions and the like. Data-centric XML is typically generated by machines and is meant for machine consumption. XML's ability to nest and repeat markup makes it a perfect choice for representing these types of data. With the introduction of XML Schema, we are now able to add data type attributes to the tags, which makes data-centric XML a very powerful mechanism to represent enterprise data, especially for data exchange and e-business.

For the purpose of this book, whenever we refer to XML, it is understood to mean data-centric XML only.

1.1.2 XML definitions

XML is a system-independent standard for the representation of data. XML is not just some new version of HTML; it is different from HTML. Like HTML, XML has tags, and in these tags it encloses data. The difference is that HTML uses its tags to display the enclosed text, and these tags are standard and fixed.

In XML you can create the tags you want, with only a small number of restrictions, and these tags will be used by a program (parser) to process the data enclosed between them.

Example 1-1 shows a simple XML document.

Example 1-1 An XML document

```
<?xml version="1.0"?>
<!DOCTYPE JavaXML:EmployeeList SYSTEM "DTD\JavaXML.dtd">
<JavaXML:employeeList xmlns:JavaXML="http://www.ibm.com">
   <JavaXML:Employee action="add">
      <JavaXML:firstName>David</JavaXML:firstName>
      <JavaXML:secondName>Sanchez Carmona</JavaXML:secondName>
      <JavaXML:age>20</JavaXML:age>
   </JavaXML:Employee>
   <JavaXML:Employee action="delete">
      <JavaXML:firstName>Jose Luis</JavaXML:firstName>
      <JavaXML:secondName>Fernandez Lastra</JavaXML:secondName>
   </JavaXML:Employee>
</JavaXML:employeeList>
```

A client with his Web browser could fill out a form, entering the names of the employees he wants to add or delete. The data could then be sent to a Web application that could process the XML document and extract the data, generating the necessary updates, for example, on a DB2® table.

As this example illustrates, the rules are very few: each tag must have an enclosing tag, and not much more. The tags are invented tags, which means that they are free-form.

Text is system-independent, and since XML is very flexible and is based only on text, it is used as the main way to transport data between different environments.

Often, XML documents are automatically generated by tools, and in many situations we need these XML documents to follow rules we create. We use other documents, containing XML data definitions in which we specify our restrictions, to accomplish this.

The most widely used rules language is Document Type Definition (DTD), described in 1.2, “Document type definition” on page 8.

XML Schema is another rules language that aims to provide more complex semantic rules. It also introduces new semantic capabilities, such as support for namespaces and type-checking within an XML document. XML Schema is described in 1.4, “XML Schema” on page 13.

1.1.3 Document validity and well-formedness

XML is reminiscent of HTML since they are both derived from SGML, which was defined in 1986. But unlike HTML, XML tags identify the data, rather than specifying how to display it. Where an HTML tag says something like “display this data in bold font” (<b>...</b>), an XML tag acts like a field name in your program. It puts a label on a piece of data that identifies it (for example: <message>...</message>). This is the first of a number of differences between the languages.

XML documents can be *well-formed*, or they can be *well-formed* and *valid*. These are two very important rules that do not exist for HTML documents. These iron-clad rules contrast with the more free-style nature of a lot of the concepts in XML. The rules can be defined briefly as follows:

- A well-formed document carries out the basic design rules for XML documents.
- A valid document respects the rules written in its DTD.

A document might be well-formed but still not be valid. Example 1-2 shows a DTD while Example 1-3 shows a sample XML document.

Example 1-2 DTD

```
<!ELEMENT BANCO (TARJETA+)>
<!ELEMENT TARJETA (Nom, Cod_Cuenta)>
<!ELEMENT Nom (#PCDATA)>
<!ELEMENT Cod_Cuenta ((#PCDATA)>
```

Example 1-3 XML document

```
<BANCO>
   <TARJETA>
      <NOM>Silvia</NOM>
      <Cod_Cuenta>2562789452</Cod_Cuenta>
   </TARJETA>
</BANCO>
```

The document shown in Example 1-3 is well-formed, but it is not valid according to the sample DTD shown in Example 1-2 because the <NOM> tag is not defined in the associated DTD (tags are case sensitive).

These examples illustrate the difference between well-formedness and validity:

- Documents that adhere to rules described in the associated DTD or XML Schema are valid.
- Documents that carry out the syntactical rules for XML documents are well-formed. These rules have to do with attribute names, which should be unique within an element, and attribute values, which must not contain the character <, and so on.

All of the constraints are defined in the XML 1.0 recommendation. For more information refer to the following Web site:

http://www.w3.org/XML

Determining whether a particular document is in compliance with these rules is a two step process. Well-formedness insures that XML parsers will be able to read the document, validity determines whether an XML document adheres to a DTD or schema. An XML application will check for and reject documents that are not well-formed before checking whether they comply with validity constraints (VCs).

Tip: After a system is tested, validity checking can be turned off to improve performance.

1.2 Document type definition

A document type definition, or DTD, specifies the kinds of tags that can be included in your XML document, the valid arrangements of those tags, and the structure of the XML document. The DTD defines the type of elements, attributes, and entities allowed in the documents, and may also specify some limitations to their arrangement. You can use the DTD to make sure you don't create an invalid XML structure since the DTD defines how elements relate to one another within the document's tree structure. You can also use it to define which attributes can be used to define an element and which ones are not allowed.

In other words, a DTD defines our own language for a specific application.

The DTD can be either stored in a separate file or embedded within the same XML file. If it is stored in a separate file it may be shared with other documents.

XML documents referencing a DTD will contain a <!DOCTYPE> declaration, which either contains the entire DTD declaration if this is the case of an internal DTD, or specifies the location of an external DTD. Example 1-4 shows an external DTD in a file named `DTD-Agenda.dtd`.

Example 1-4 An external DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT AGENDA (PERSONA+)>
<!ELEMENT PERSONA (EMPRESA, CARGO, NOMBRE, TELEFONO1+,TELEFONO2*, EXT?)>
<!ELEMENT EMPRESA (#PCDATA)>
<!ELEMENT CARGO (#PCDATA)>
<!ELEMENT NOMBRE (#PCDATA)>
<!ELEMENT TELEFONO1 (#PCDATA)>
<!ELEMENT TELEFONO2 (#PCDATA)>
<!ELEMENT EXT (#PCDATA)>
```

Example 1-5 is an XML document that refers to this external DTD.

Example 1-5 Reference to an external DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AGENDA SYSTEM "DTD_Agenda.dtd">
<AGENDA>
   <PERSONA>
      <EMPRESA>Matutano</EMPRESA>
      <CARGO>Gerente</CARGO>
      <NOMBRE>Pepe Montilla</NOMBRE>
      <TELEFONO1>912563652</TELEFONO1>
      <TELEFONO2>658968574</TELEFONO2>
      <EXT>256</EXT>
   </PERSONA>
</AGENDA>
```

We can also define an internal DTD in a XML document so that both of them are in the same file. Example 1-6 shows this case. Notice that in any case, internal or external DTD, the <!DOCTYPE> declaration indicates what the root element is.

Example 1-6 An internal DTD

```
<!DOCTYPE RAIZ [
<!ELEMENT ...... >
<!ELEMENT ...... >
<!ELEMENT ...... >
<!ELEMENT ...... >
]>
<RAIZ>
.
.
.
</RAIZ>
```

An XML document is not required to have a DTD. DTDs provide parsers with clear instructions on what to check for when they are determining the validity of an XML document. DTDs or other mechanisms, like XML schemas, contribute to the goal of ensuring that the application can easily determine whether the XML document adheres to a given set of rules, beyond the well-formedness rules defined in the XML standard. DTDs are also used by tools to create XML documents.

Having the logical definition of an XML file stored separately allows for the resulting DTD to be shared across organizations, industries, or the Web. When building XML applications, it is probably a good idea to look for existing DTDs that might suit your purpose. As XML becomes more popular, more commercially and industrially oriented applications will likely appear, and standards will emerge. For more information on the latest emerging XML standards refer to the following Web site:

http://www.oasis-open.org

DTD contents overview

The purpose of a DTD is to define the valid building blocks of an XML document. It defines the document structure with a list of acceptable elements. Seen from a DTD point of view, all XML documents (and HTML documents) are made up of the following simple building blocks:

- **Elements**

  In a DTD, XML elements are declared with a DTD element declaration. The syntax of this declaration is:

  ```
  <!ELEMENT element-name(allowed element contents)>
  ```

- **Attributes**

  Attributes are additional information about an element. The syntax of this declaration is:

  ```
  <!ATTLIST element-name attribute-name value-type attribute-type "default">
  ```

- **Entities**

  Entities are defined abbreviations so that when the document is analyzed any reference to the entity is replace with the character string represented. An example of this declaration, together with its appearance in the document, is:

```
<!DOCTYPE text
<!ENTITY ovni "Objeto Volante No Identificado" >
]>
<texto><titulo>Un día en la vida de un &ovni; </titulo></texto>
```

- **Parameter entities**

  Parameter entities are used within the DTD itself. Their declaration differs by the inclusion of the % character. An example of this declaration is as follows:

```
<!ENTITY % commonAtts “ID ID #REQUIRED MAKE CDATA #IMPLIED MODEL CDATA
#IMPLIED”>
<!ELEMENT CAR (#PCDATA)>
<!ATTLIST CAR %commontAtts>
<!ELEMENT COMPUTER (#PCDATA)>
<!ATTLIST COMPUTER %commonAtts>
```

- **Notations**

  Notations are used to refer to data from an outside (non-XML) source. They provide a basic means by which non-textual information can be handled within a document. An example of the use of this declaration is as follows:

```
<!DOCTYPE person [
<!NOTATION jpeg SYSTEM “jpeg.exe”>
<!NOTATION gif SYSTEM “gif.exe”>
<!ELEMENT person (#PCDATA)>
<!ATTLIST person picformat NOTATION (jpeg | gif) #REQUIRED>
]>
<person picformat=”jpeg”>Kelly Brown</person>
```

- **COMMENTS**

  It is possible to insert any comment as shown in the following:

```
<!-- any XML comment -->
```

DTD limitations

Some people think that DTD syntax is dreadful and hard to enlarge. It is possible that a parser might find “hello” when it is expected to find numeric data in a valid document.

DTDs do have a few limitations, for example:

- A DTD makes it possible to validate the structure of relatively simple XML documents, but that's as far as it goes. A DTD can't restrict the content of elements, and it can't specify complex relationships.
- In a DTD, you only get to specify the structure of the <heading> element one time. There is no context-sensitivity.
- A DTD specification is not hierarchical. For a mailing address that contains several “parsed character data” (PCDATA) elements, for instance, the DTD might look something like that shown in Example 1-7. As you can see, the specifications are linear. That fact forces you to come up with new names for similar elements in different settings.

Example 1-7 The need for namespaces

```
<!ELEMENT mailAddress (name, address, zipcode)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT zipcode (#PCDATA)>
```

- Another problem with the non-hierarchical nature of DTD specifications is that it is not clear what comments are meant to explain.
- Finally, a DTD uses syntax which is substantially different from XML, so it can't be processed with a standard XML parser. That means you can't read a DTD into a DOM, for example, modify it, and then write it back out again.

1.3 Namespaces

Before talking about XML Schema, we must first introduce *Namespaces*. Namespaces are used when there is a need to have different elements with different attributes but with the same name. Depending of the context, a tag is related to an element or to another one. Example 1-8 illustrates this situation.

Example 1-8 The need for namespaces

```
<widget type="gadget">
   <head size="medium"/>
      <info>
         <head>
          <title>Description of gadget</title>
         </head>
         <body>
          <h1>Gadget</h1>
         </body>
      </info>
</widget>
```

It is obvious there is a problem with the meaning of `<head>`. It depends on the context. This situation complicates things for processors and might even cause ambiguities. We need some mechanism to distinguish between the two, and apply the correct semantic description to the correct tag. The root of the problem is one common name space.

There is a simple solution to this problem: Namespaces. Namespaces are a simple and straightforward way to distinguish names used in XML documents. If you can specify the related DTD when an element is being validated, the problem is solved.

Example 1-9 Namespaces

```
<?xml version="1.0" ?>
<library-entry xmlns:authr="http://sc58ts.itso.ibm.com/Jose/2002/author.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>JuanJose</authr:firstname>
      <authr:lastname>Hernandez</authr:lastname>
      <authr:title>Mr</authr:title>
    </authr:author>
  </bk:book>
</library-entry>
```

As you can see in Example 1-9, the `<title>` tag is used twice, but in a different context: once within the `<author>` element and once within the `<book>` element. Note the use of the `xmlns` keyword in the namespace declaration. Interestingly, the XML recommendation does not specify whether a namespace declaration should point to a valid Uniform Resource Identifier (URI), only that it should be unique and persistent.

In the previous example, in order to illustrate the relationship of each element to a given namespace, we specified 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 overridden by another prefix. The extent to which a namespace prefix applies to elements in a document is defined as the namespace scope.

Example 1-10 is equivalent to Example 1-9, but only the necessary namespace prefixes have been used.

Example 1-10 Namespaces

```
<?xml version="1.0" ?>
<library-entry xmlns:authr="http://sc58ts.itso.ibm.com/Jose/2002/author.dtd"
xmlns:bk="books.dtd">
  <bk:book>
    <title>XML Sample</title>
    <pages>210</pages>
    <isbn>1-868640-34-2</isbn>
    <authr:author>
      <firstname>JuanJose</firstname>
      <lastname>Hernandez</lastname>
      <title>Mr</title>
    </authr:author>
  </bk:book>
</library-entry>
```

Information on namespaces can be found at the following Web site:

http://www.w3.org/TR/REC-xml-names

1.4 XML Schema

The W3C XML Schema Definition Language is an XML language for describing and constraining the content of XML documents. A *Schema* is similar to a DTD in that it defines which elements an XML document can contain, how they are organized, and which attributes and attribute types elements can be assigned. Therefore it is a method to check the validity of well-formed XML documents. The main advantages of Schemas over DTDs are:

- Schemas use XML syntax.
- It is possible to specify data types.
- Schemas are extensible.

DTD and XML Schema

In "DTD limitations" on page 11 we identified some of the limitations and problems of DTDs. In addition to those limitations, we can add the following: limitations and add some new ones:

- There are no constraints on character data. If character data is allowed, any character data is allowed.
- The attribute value models are too simple.
- There is no support for namespaces.
- There is no support for schema evolution, extension, or inheritance of declarations.
- It is difficult to write, maintain, and read large DTDs, and to define families of related schemas.
- It is only possible to set defaults for attributes, not for elements.

Therefore, there is a need for a way to specify more complex semantic rules and provide all those things that DTDs cannot do, like type-checking within an XML document. XML Schema, a W3C Recommendation as of May 2001, aims to provide such functionality; it also introduces new semantic capabilities, such as support for namespaces and type-checking.

XML Schema example

It is difficult to give a general outline of the elements of a schema due to the number of elements that can be used according to the W3C XML Schema Definition Language. The purpose of this language is to provide an inventory of XML markup constructs with which to write schemas. Example 1-11 is a simple document which describes the information about a book.

Example 1-11   A book description

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">
   <title>
   Don Quijote de la Mancha
   </title>
   <author>De Cervantes Saavedra, Miguel</author>
   <character>
      <name>Sancho Panza</name>
      <friend-of>El Quijote</friend-of>
      <since>1547-10-04</since>
      <qualification> escudero </qualification>
   </character>
   <character>
      <name>ElbaBeuno</name>
      <since>1547-08-22</since>
      <qualification>Amor Platonico de Don Quijote</qualification>
   </character>
</book>
```

Since the XML Schema is a language, there are several choices to build a possible schema that covers the XML document. Example 1-12 is a possible and very simple design.

Example 1-12 XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="friend-of" type="xs:string" minOccurs="0"
                 maxOccurs="unbounded"/>
              <xs:element name="since" type="xs:date"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

It is clear that Example 1-11 is an XML document since it begins with the XML document declaration. The schema element opens our schema holding the definition of the target namespace. Then we define an element named `book`. This is the root element in the XML document. We decided it is a complex type since it has attributes and non-text children. With `sequence` we begin to declare the children elements of the root element `book`. W3C XML Schema lets us define the type of data, as well as the number of possible occurrences of an element. For more information on possible values for these types, refer to the specification documents from W3C.

XML Schema options

XML Schema Language offers possibilities and alternatives beyond what is shown in Example 1-12. We could develop another schema based on a flat catalog of all the elements available in the instance document and, for each of them, lists of child elements and attributes. Thus we would have two choices: defining elements and attributes as they are needed, or creating them first and referencing them. The first option has a real disadvantage: the schema could became very difficult to read and maintain when documents are complex.

W3C XML Schema allows us to define data types and use these types to define our attributes and elements. It also allows the definition of groups of elements and attributes. In addition, there are several ways to arrange relationships between elements.

Documentation for XML Schemas can be defined by the `xs:documentation` element, and processing instructions for applications can be include with the `xs:appinfo` element. More details are on the Web at:

http://www.w3.org/TR/NOTE-xml-schema-req

1.5 XSL - Extensible Stylesheet Language

Up to this point we have been concerned with XML—its syntax, how XML is used to mark up information according to our own vocabularies, how a program can check the validity of an XML document, and so forth. In other words, we have described how XML can ensure that an application running on any particular platform receives valid data. This is how we ensure that a program will be able to process this data.

However, since XML only describes document syntax, the program will not know how to format this data without specific instructions about style.

The solution is XSL transformations. The Extensible Stylesheet Language (XSL) specification describes powerful tools to accomplish the required transformation of XML data. XSL consists of:

- The XSL Transformations (XSLT) language for transformation
- Formatting Objects (FO), a vocabulary for describing the layout of documents
- XSLT uses the XML Path Language (XPath), a separate specification that describes a means of addressing XML documents and defining simple queries.

XSLT offers a powerful means of transforming XML documents into other forms, producing XML, HTML, and other formats. It is capable of sorting, selecting, numbering, and has many other features for transforming XML. It operates by reading a style sheet, which consists of one or more templates, then matching the templates as it visits the nodes of the XML document. The templates can be based on names and patterns.

XSLT is increasingly being used to transform XML data into another form, sometimes different XML (for example, filtering out certain data, SQL statements, plain text, and so on), or any other format. Thus, any XML document may be shown in different formats, such as HTML, PDF, RTF, VRML, Postscript, and so forth.

XPath

XPath is a string syntax for building addresses to the information found in an XML document. We use this language to specify the locations of document structures or data found in an XML document when processing that information using XSLT. XPath allows us from any location to address any other location or content. In other words, XPath is a tool used in XSLT to select certain information to be formatted.

XPath patterns

Some XPath patterns are shown in Table 1-1. These are just a few examples to give you an idea what kind of things can be selected.

Table 1-1 XPath

Symbol	**Meaning**
/	Refer to immediate child
//	Refer to any child in the node
.	Refer to actual context
*	Refer to all elements in the actual node
@	Refer to an attribute
@*	Refer to all attributes in the actual node

XPath models an XML document as a tree of nodes, as follows:

- Root nodes
- Element nodes
- Attribute nodes
- Text nodes
- Namespace nodes
- Processing instruction nodes
- 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

Example 1-13 Xpath

```
<?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>
```

Considering Example 1-13, we could make paths like:

- /book/copies

  Selects all `copies` element children of `book` elements.

- /book//title

  Selects all `title` elements in the tree, although `title` elements are not immediate children.

- /book/@ID

  Selects all `ID` attributes beyond `book` elements.

But as we mentioned previously, it is also possible to select elements based on other criteria, such as:

- /library/*/book[title eq "good story"]

  Selects all `book` elements beyond `library` element, but only if the `title` element matches with good story.

XSLT

The question (and programming challenge!) is how to access and display the information contained in an XML file. After all, data is useless unless you can use it. This is where XSLT comes into the picture.

A comparison can be made between the relationship of CSS and HTML and the relationship of XSLT and XML. Indeed, XSLT is usually referred to as the *stylesheet language* of XML; however XML and XSLT are far more sophisticated technologies than HTML and CSS.

XSLT is a high-level declarative language. It is also a transforming and formatting language. It behaves in the following way:

- The pertinent data is extracted from an XML source document and transformed into a new data structure that reflects the desired output. The XSLT markup is commonly called a *stylesheet*. A parser is used to convert the XML document into a tree structure composed of various types of nodes. The transformation is accomplished with XSLT by using pattern matching and templates. Patterns are matched against the source tree structure, and templates are used to create a result tree.
- Next, the new data structure is formatted, for example in HTML or as text, and finally the data is ready for display.

Figure 1-1 shows the source tree from the XML document shown in Example 1-13.

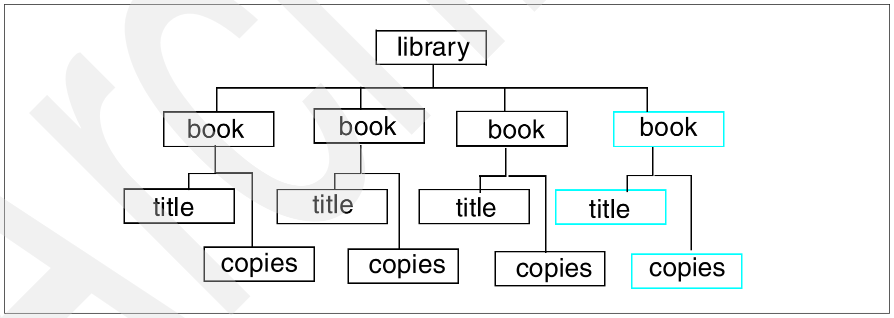


Figure 1-1 DOM tree

The result tree after an XSL transformation could be an XHTML document, as shown in Figure 1-2.

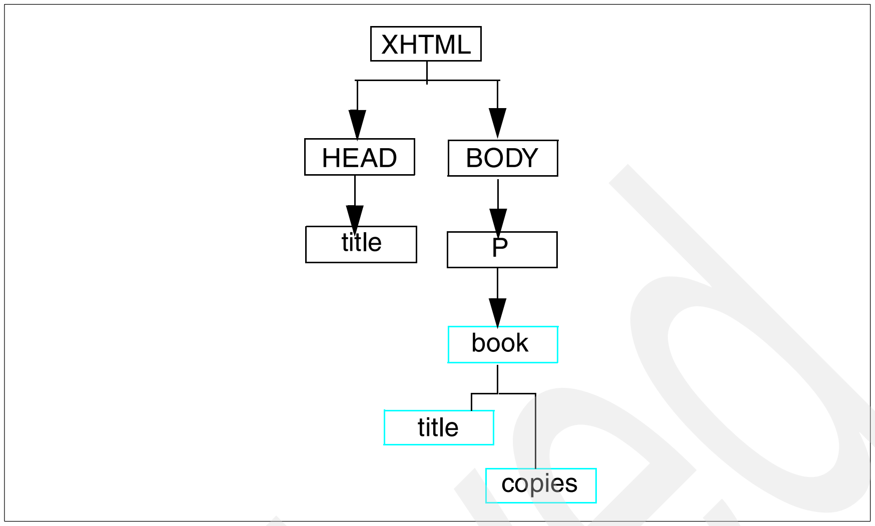


Figure 1-2 DOM tree after XSL transformation

Based on how we instruct the XSLT processor to access the source of the data being transformed, the processor will incrementally build the result by adding the filled-in templates. We write our stylesheets, or “transformation specifications,” primarily with declarative constructs, though we can employ procedural techniques if and when needed. We assert the desired behavior of the XSLT processor based on conditions found in our source.

Note that XSLT only manipulates the source tree and that the original XML document is left unchanged.

The most important aspect of XSLT is that it allows you to perform extremely complex manipulations on the selected tree nodes by affecting both content and appearance. Indeed, the final output may bear absolutely no resemblance to the source document. This ability to manipulate the nodes is where XSLT far surpasses CSS.

The World Wide Web Consortium (W3C) has set the recommended standards for XSLT Version 1.0. The W3C proposed recommendation for XSL is available at the following URL:

http://www.w3.org/TR/xslXHTML

1.6 XHTML

The history of XHTML is very simple: it is derived directly from HTML version 4.01 and is designed to be used with XML. Indeed, XHTML is part of a whole new suite of “X” technologies, with acronyms such as XML, XPATH, XSL, and XSLT, that are destined to have a profound effect on the Internet.

People often think XML is an extension of HTML, but we have already dispelled this notion. XHTML is the real extension of HTML.

There are a few fundamental differences between HTML and XHTML that will significantly affect how you code with XHTML. While HTML is a loose and forgiving language, XHTML will quickly remind you of a strict English teacher who demands firm adherence to the rules of grammar.

Fortunately, the syntax and coding rules are very straightforward, easy to implement, and they make sense. The real purpose of these rules is to allow a seamless integration of XHTML with XML and other related X technologies. The rules are summarized as follows:

- All attributes, events, and tags must be written in lower case.
- All elements must be closed.
- The value assigned to an attribute must be enclosed in quotes.
- No attribute may be minimized.
- All elements must be properly nested.
- XHTML documents must be well-formed.
- There must be a DOCTYPE declaration.

Notice that this last rule implies that there must be a DTD to validate the XHTML document. HTML has become an XML document.

1.6.1 XHTML document types

XHTML 1.0 specifies three XML document types that correspond to three DTDs: Strict, Transitional, and Frameset. The most common is XHTML transitional. The `DOCTYPE` declaration at the beginning of the XHTML document specifies which type is being used.

XHTML 1.0 Strict

Use this when you want really clean markup, free of presentational clutter. Use this together with Cascading Style Sheets. Example 1-14 shows a strict DTD.

Example 1-14 Strict DTD

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1.0 Transitional

Use this when you need to take advantage of HTML's presentational features and when you want to support browsers that don't understand Cascading Style Sheets. Example 1-15 shows a transitional DTD.

Example 1-15 Transitional DTD

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset

Use this when you want to use HTML Frames to partition the browser window into two or more frames. Example 1-16 shows a frameset DTD.

Example 1-16 Frameset DTD

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

1.6.2 Xlink

XLink is a powerful and compact specification for the use of links in XML documents.

Every developer is familiar with the linking capabilities of the Web today. However, as the use of XML grows, we quickly realize that simple tags like the following ones are not going to be enough in a near future.

```
<a href="elem_lessons.html">Freud</a information about X > are not going to be
enough for many of our needs.
```

XML Linking Language (XLink) allows elements to be inserted into XML documents to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links.

XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to:

- Assert linking relationships among more than two resources
- Associate metadata with a link
- Express links that reside in a location separate from the linked resources

Even though XLink has not been implemented in any of the major commercial browsers yet, its impact will be crucial for the XML applications of the near future. Its extensible and easy-to-learn design should prove an advantage as the new generation of XML applications develop.

For more information about Xlink, refer to the specification document from W3C:

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

Xpointer

XML Pointer Language (Xpointer) specifies a language that builds upon the XML Path Language (XPath), to support addressing into the internal structures of XML documents. In particular, it provides for specific references to elements, character strings, selections, and other parts of XML documents—whether or not they bear an explicit ID attribute—using traversals of a document's structure and choice of parts based on their properties, such as element types, attribute values, character content, and relative position, containment, and order. Xpointer defines the meaning of the “selector” or “fragment identifier” portion of URIs that locate resources of MIME media types text/xml and application/xml.

In Xpointer, one defines the addressing expression to link XML documents using XPath. For more information about XPointer, refer to the specification documents from W3C:

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

1.7 XSL, XSLT, Xpath, and XHTML examples

Let's first look at some example stylesheets using two implementations of XSLT 1.0 and XPath 1.0

Consider the XML file shown in Example 1-17. It is a very simple file we are going to use as the source of information for our XSLT transformation.

Example 1-17 hello.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<greeting>Hello world.</greeting>
```

Note that the stylesheet association processing instruction in line 2 refers to a stylesheet with the name `hello.xsl` of type XSL. Recall that an XSLT processor is not obliged to respect the stylesheet association preference, so let us first use a standalone XSLT processor with the stylesheet hellohtm.xsl, shown in Example 1-18.

Example 1-18 hellohtm.xsll

```
<?xml version="1.0"?><!--hellohtm.xsl-->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">
<head><title>Greeting</title></head>
<body><p>Words of greeting:<br/>
<b><i><u><xsl:value-of select="greeting"/></u></i></b>
</p></body>
</html>
```

This file looks like a simple XHTML file: an XML file using the HTML vocabulary. Indeed, it is just that, but we are allowed to inject into the instance XSLT instructions using the prefix for the XSLT vocabulary declared in line 3. We can use any XML file as an XSLT stylesheet provided it declares the XSLT vocabulary within and indicates the version of XSLT being used. Any prefix can be used for XSLT instructions, though convention often sees XSL: as the prefix value.

The `xsl:value-of` instruction uses an XPath expression in the `select=` attribute to calculate a string value from our source information. XPath views the source hierarchy using parent/child relationships. The XSLT processor's initial focus is the root of the document, which is considered the parent of the document element. Our XPath expression value `"greeting"` selects the child named `"greeting"` from the current focus, thus returning the value of the document element named `"greeting"` from the instance.

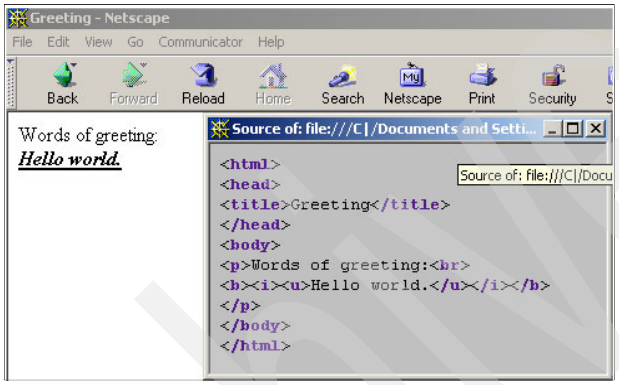
We invoke the XSLT processor to point to which is the XML source file, which is the XSL stylesheet, and where to leave the result. Example 1-19 shows the result file.

Example 1-19 Output from XSLT processor

```
<html>
<head>
```

```
<title>Greeting</title>
</head>
<body>
<p>Words of greeting:<br>
<b><i><u>Hello world.</u></i></b>
</p>
</body>
</html>
```

As you can see in Figure 1-3, this is an HTML file, so any browser could interpret it and generate a display.


Greeting - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Source of: file:///C|/Documents and Setti...
Source of: file:///C|/Docu

Words of greeting:
Hello world.

```
<html>
<head>
<title>Greeting</title>
</head>
<body>
<p>Words of greeting:<br>
<b><i><u>Hello world.</u></i></b>
</p>
</body>
</html>
```

Figure 1-3 Hello world

1.8 Real-life uses of XML

XML is already used in a wide range of fields and industries, including science and medicine, publishing, broadcasting, communications, and financial services. On the W3C Web site there is information about the standards and recommendations for several. At the end of this section we provide several links to information about the standards.

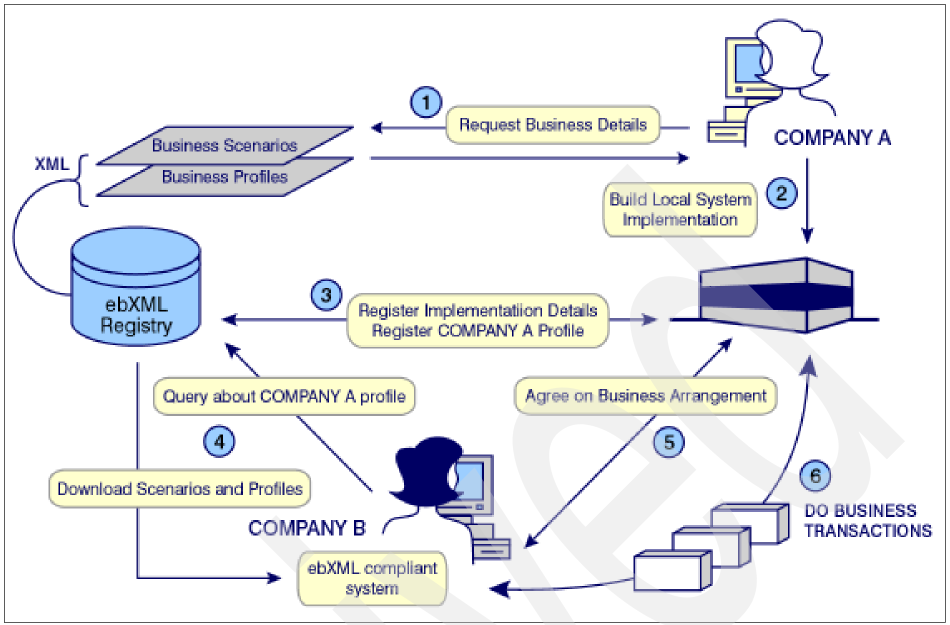
ebXML

ebXML is a set of specifications that together enable a modular electronic business framework. The vision of ebXML is to enable a global electronic marketplace where enterprises of any size and in any geographical location can meet and conduct business with each other through the exchange of XML-based messages. Or in other words, ebXML hopes to succeed Electronic Data Interchange, more often known by its abbreviation, EDI.

ebXML adds new acronyms and other special terms that should be mentioned to help you understand the whole “vision” of ebXML interactions.

- When a business wants to start an ebXML relationship with another business, it queries a Registry (information in XML form) to locate a suitable partner and to find information about requirements for dealing with that partner.
- Business Processes: Activities that a business can engage in (and for which it would generally want one or more partners).
- Collaboration Protocol Profile (CPP): A profile filed with a Registry by a business wishing to engage in ebXML transactions. The CPP specifies some Business Processes of the business, as well as some Business Service Interfaces it supports.
- Core Library: A set of standard “parts” that may be used in larger ebXML elements. For example, Core Processes may be referenced by Business Processes. The Core Library is contributed by the ebXML initiative itself, while larger elements may be contributed by specific industries or businesses.
- Collaboration Protocol Agreement (CPA): In essence, a contract between two or more businesses that can be derived automatically from the CPPs of the respective companies. If a CPP says “I can do X,” a CPA says “We will do X together.”

Figure 1-4, based on the ebXML Technical Architecture Specification, illustrates what ebXML means for business.



XML
Business Scenarios
Business Profiles
1
Request Business Details
COMPANY A
2
Build Local System Implementation
ebXML Registry
3
Register Implementation Details
Register COMPANY A Profile
Query about COMPANY A profile
Agree on Business Arrangement
4
5
6
Download Scenarios and Profiles
COMPANY B
DO BUSINESS TRANSACTIONS
ebXML compliant system

Figure 1-4 ebXML

Company A will first review the contents of an ebXML Registry, especially the Core Library which can be downloaded or viewed there. The Core Library (and maybe other registered Business Processes) will allow Company A to determine the requirements for their own implementation of ebXML (and whether ebXML is appropriate for their business needs). Based on a review of the information available from an ebXML Registry, Company A can build or buy an ebXML implementation suitable for its anticipated ebXML transactions. The hope of the ebXML initiative is that vendors will support all of the elements of ebXML. At such time, an “ebXML system” might be little more than a prepackaged desktop application. Or maybe, more realistically, the ebXML system will at least be as manageable as a commercial database system (which still needs a DBA).

Either way, the next step is for Company A to create and register a CPP with the Registry. The CPP will contain the information necessary for a potential partner to determine the business roles in which Company A is interested, and the type of protocols it is willing to engage in for these roles.

Finally, the two companies begin actual transactions. ebXML will have helped in agreeing to, monitoring, and verifying these real-world activities.

For more information about ebXML, refer to the following Web sites:

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

http://www.ebxml.org

References

- **Schema standards**

  XML Schema

  http://www.w3.org/XML/Schema

  TREX (Tree Regular Expressions for XML)

  http://www.thaiopensource.com/trex/

  SOX (Schema for Object-oriented XML)

  http://www.w3.org/TR/NOTE-SOX

  Schematron (Schema for Object-oriented XML)

  http://www.ascc.net/xml/resource/schematron/schematron.html

- **Linking and presentation standards**

  Xlink, Xpointer

  http://www.w3.org/XML/Linking

  XHTML

  http://www.w3.org/TR/xhtml1

- **Knowledge standards**

  RDF (Resource Description Framework)

  http://www.w3.org/TR/REC-rdf-syntax

  RDF Schema

  http://www.w3.org/TR/rdf-schema

  XTM (XML Topic Maps)

  http://www.topicmaps.org/xtm/index.html

- **Standards that build on XML**

  SMIL (Synchronized Multimedia Integration Language)

  http://www.w3.org/TR/REC-smil

  MathML (Mathematical Markup Language)

  http://www.w3.org/TR/REC-MathML

1.8.1 XML parsers

An XML parser is a software module that is used to read XML documents and provide application programs with access to their content and structure. Several XML parsers are now available that support many languages, such as Java, C/C++, Enterprise COBOL, and so forth. Some of them are validating parsers, while others are non-validating. Validating parsers typically allow validation to be performed or bypassed under application control. When reading an XML document, a validating parser checks the validity constraints and the well-formedness constraints defined in the XML recommendation.

The current APIs for accessing XML documents, either serially or in random access mode are, respectively, SAX and DOM.

SAX

The Simple API for XML (SAX) defines an API for an event-based parser. Being event-based means that the parser reads an XML document from beginning to end, and each time it recognizes a syntax construction, it notifies the application that is running it. The SAX parser notifies the application by calling methods from the ContentHandler interface. For example, when the parser detects the start element, it calls the startElement method; when it comes to character data, it calls the characters method; when it comes to the end element it calls the endElement method.

SAX is a public domain API developed cooperatively by the members of the XML-DEV mailing list. It provides an event-driven interface for the purpose of parsing XML documents.

An event-driven interface provides a mechanism for “callback” notifications to an application’s code as the underlying parser recognizes XML syntactic constructions in the document.

DOM

The Document Object Model (DOM) is a set of interfaces defined by the W3C DOM Working Group. It describes facilities for a programmatic representation of a parsed XML (or HTML) document. The DOM Level 2 specification defines these interfaces using Interface Definition Language (IDL) in a language-independent fashion, and also includes a Java Language binding.

The Java API for XML processing specification includes by reference both the abstract semantics described for the DOM Level 2 Core Recommendation interfaces and the associated Java Language binding. It does not include the optional extensions defined by the DOM working group.

The API package included by the Java API for XML processing specification is:

```
http://www.w3.org/DOM/
```

JAXP

The Java API for XML Processing (JAXP) makes it easy to process XML data using applications written in the Java programming language. JAXP leverages the parser standards SAX (Simple API for XML Parsing) and DOM (Document Object Model) so that you can choose to parse your data as a stream of events or to build a tree-structured representation of it. The latest versions of JAXP also support the XSLT (XML Stylesheet Language Transformations) standard, giving you control over the presentation of the data and enabling you to convert the data to other XML documents or to other formats, such as HTML. JAXP also provides namespace support, allowing you to work with schemas that might otherwise have naming conflicts.

Designed to be flexible, JAXP allows you to use any XML-compliant parser from within your application. It does this with what is called a “plugability layer,” which allows you to plug in an implementation of the SAX or DOM APIs. The plugability layer also allows you to plug in an XSL processor, which lets you transform your XML data in a variety of ways, including the way it is displayed.

The latest version of JAXP is JAXP 1.2, which adds support for XML Schema.

2

XML Toolkit for z/OS and OS/390

In this chapter we provide information about the XML Toolkit for z/OS and OS/390 V1R4. We describe the product and its components. We also describe a non-SMP/E installation and the environments where the toolkit can be executed.

Note: As this publication goes to print, the XML Toolkit for z/OS and OS/390 Version 1 Release 5 has been released. This new release provides updates to the XML Java and C++ parsers, based on the Apache Software Foundation's Xerces2 Java Parser 2.2.1 and Xerces-C++ Version 2.1.0. This version also includes updates to the XSLT processor based on the Apache Software Foundation's Xalan Java Version 2.4.1.

For more information, see the XML Toolkit Web page at:

http://www-1.ibm.com/servers/eserver/zseries/software/xml/

2.1 XML toolkit components

The XML toolkit includes the following components:

- XML Parser for z/OS and OS/390, C++ Edition

  This corresponds to the Apache Software Foundation's Xerces-C 1.6.0.
- XML Parser for z/OS and OS/390, Java Edition

  This corresponds to the Apache Software Foundation's Xerces2 Java Parser 2.0.1.

The XML Toolkit for z/OS and OS/390 V1R4 provides the following enhancements to the previous releases:

- Full Schema support is now offered for the C++ parser.
- Within the Java release, a rewrite of the code base has been done, providing a cleaner, more modular, and easier to maintain design. External APIs were not affected by this change.
- Support for the XSLT Processor in MVS native mode has been added.

Table 2-1 shows the interfaces and specifications for XML parsers in the XML toolkit for z/OS and OS/390.

Table 2-1 XML toolkit for z/OS and OS/390

Interfaces & specifications	C++ Edition parser	Java Edition parser
DOM 1.0	Completely supported	Completely supported
DOM 2.0	Completely supported	Completely supported
DOM 3.0	-----------	Subset of experimental APIs
SAX 1.0	Completely supported	Completely supported
SAX 2.0	Completely supported	Completely supported
IDOM	Completely supported	Completely supported
XML 1.0	Completely supported	Completely supported
Namespaces	Completely supported	Completely supported
Schema	Completely supported	Completely supported
JAXP 1.0	-----------	-----------
JAXP 1.1	-----------	Completely supported

At the time of writing this redbook, DOM 3.0 has not reached W3C recommendation status. Since it is subject to change, this API is not supported. Consequently, fixes may not be made available for this experimental API.

Table 2-2 shows the interface and specifications for XSL processors in the XML toolkit.

Table 2-2 XSL processors in the XML toolkit

Interfaces and specifications	C++ Edition processor	Java Edition processor
XSLT Transformation	Completely supported	Completely supported
XPATH 1.0	Completely supported	Completely supported
TRaX	-----------	Completely supported

Sample applications of XML parsers and XSL processors have also been provided to demonstrate the features of the both editions of the toolkit. The results of running these samples are documented in the next chapter.

2.2 Operating environments

2.2.1 XML Toolkit for z/OS and OS/390, Java Edition

CICS®

Operation with CICS Transaction Server can be achieve by exploiting the capability to run a Java Virtual Machine (JVM) within the CICS Transaction Server V2.2 or above. CICS application programs written in Java are able to run in a CICS address space under the control of a JVM. The internal architecture of CICS TS 2.2 allows specified user tasks to run under their own task control block (TCB). The new TCBs under which tasks, optionally, can run are known as open TCBs. These form the basis for CICS becoming an open transaction environment in the longer term.

Interoperability with CICS TS 2.2 is supported via CICS Transaction Gateway 4.02, which transmits application requests to CICS.

WebSphere Application Server

The XML parser and XSL processor for z/OS and OS/390, Java edition can be used with applications running in WebSphere Application Server for z/OS or OS/390. Use of the IBM Common Connector Framework, which is supported by WebSphere, also provides interoperability with IBM's enterprise transaction server processing platforms, like CICS; it converts parsed data into a

COMMAREA, and communicates with CICS. WebSphere Application Server also provides IMS™ connectors. This is discussed in more detail in the later chapters of this book.

IMS Transaction Manager

As we mentioned previously, a Java application can invoke the OS/390 XML parser or the XSL processor, Java Edition and then use IMS Connector to send a message to IMS.

XML documents can be used to drive existing IMS transactions. The Java parser runs in the IMS Transaction Manager V7 environment. Transformation to and from the IMS message format is the responsibility of the application. New IMS applications can also use XML. Written in Java, they can receive XML input, parse the document and process the data directly.

Batch

Batch programs that invoke the XML parser or the XSL processor Java edition must run in a JVM.

2.2.2 XML Toolkit for z/OS and OS/390, C++ Edition

The XML parser and XSL processor for z/OS and OS/390, C++ edition can be invoked by any program that can access C++ code running under the OS/390 Language Environment®, except in a CICS environment. It can also be invoked from a C or C++ DB2 application or stored procedure.

CICS Transaction Server

The C++ parsers in CICS are not supported. The initial beneficiaries of the changes brought in CICS TS 2.2 are Java application programs. These Java programs have direct access to CICS resources via the new JCICS classes, and run under the OS/390 JVM, at JDK 1.3.1 or higher, attached by CICS under a new TCB (OTE). CICS C++ programs run under a single TCB, the quasi-reentrant TCB, and are not allowed to access files that are not controlled by CICS TS.

IMS Transaction Manager

The XML C++ parser and XSL processor APIs are supported in IMS.

Batch and UNIX System Services

The XML C++ parser and XSL processor are supported in batch and in UNIX System Services.

2.2.3 XML Toolkit V1R4 requirements

The software prerequisite for running the XML toolkit is one of the following:

- OS/390 V2R8 with Language Environment and OS/390 UNIX System Services active.
- z/OS V1R1 with Language Environment and OS/390 UNIX System Service active.

This is mandatory for a successful installation; in addition, there are some functional requirements that must be met at run time for this product to work. These requirements are identified in Table 2-3.

Table 2-3 XML Toolkit co-requisites

Product name	**Function**
IBM Developer Kit for OS/390, Java 2 Technology Edition V1R1 with PTF UQ61198 or higher	XML parser for z/OS and OS/390, Java edition and XSLT processor for z/OS and OS/390, Java edition
Java SDK 1.3.1	XML parser for z/OS and OS/390, Java Edition

The toolkit is compatible with IBM and non-IBM hardware and software platforms. Key uses include:

- Categorizing and tagging data for exchange in disparate environments.
- Transforming “ad hoc” unstructured data to XML records, enabling you to search, cross-reference, and share records.

2.3 XML Toolkit V1R4 installation and configuration

In the following sections, we describe how to install and configure the XML Toolkit.

2.3.1 Obtaining the toolkit

Note: These instructions *do not* replace the XML Toolkit for z/OS and OS/390 Program Directory. Refer to that document and the XMLSMPE.README.txt file supplied with the download to accomplish the installation.

The XML Toolkit for z/OS and OS/390 can be distributed on a product tape if you wish, but it can also be downloaded from the Web. If you choose this second option, follow the instructions provided in the Program Directory available at the following URL:

http://www.ibm.com/servers/eserver/zseries/software/xml/pdf/v1r4.pdf

A number of sample jobs are provided to help you install the toolkit into its own SMP/E environment. But before running these jobs you must define all DDDEF entries in the appropriate zones. Instead of running the samples, you could use the SMP/E Dialogs.

The SMP/E GIMUNZIP function is required to process the downloaded package. For a description of this function refer to the SMP/E Web page at:

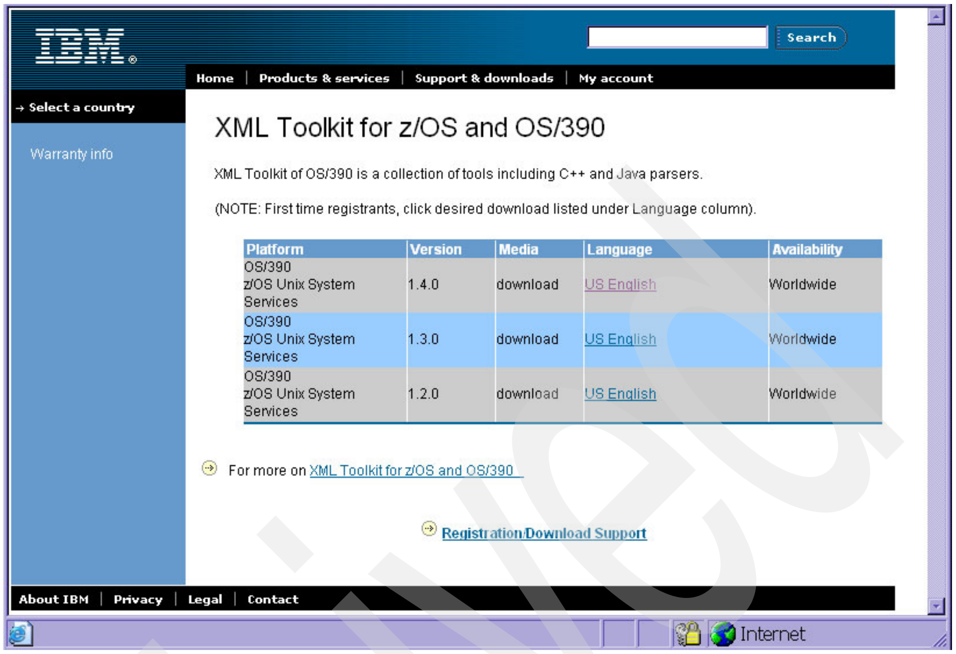
http://www.ibm.com/servers/eserver/zseries/zos/smpe/

Before downloading the toolkit, allocate an R/W HFS directory into which to stage the download package using the **mkdir** command. You can also choose the *tmp* file of the Hierarchical File System. In this case you are not interested on keeping the zip files from the download.

You are now ready to start the download process. The package is available from the following Web site:

http://www.ibm.com/servers/eserver/zseries/software/xml/download/

You must fill out a registration form prior to beginning the download.


Search
Home | Products & services | Support & downloads | My account
→ Select a country
Warranty info

XML Toolkit for z/OS and OS/390

XML Toolkit of OS/390 is a collection of tools including C++ and Java parsers.

(NOTE: First time registrants, click desired download listed under Language column).

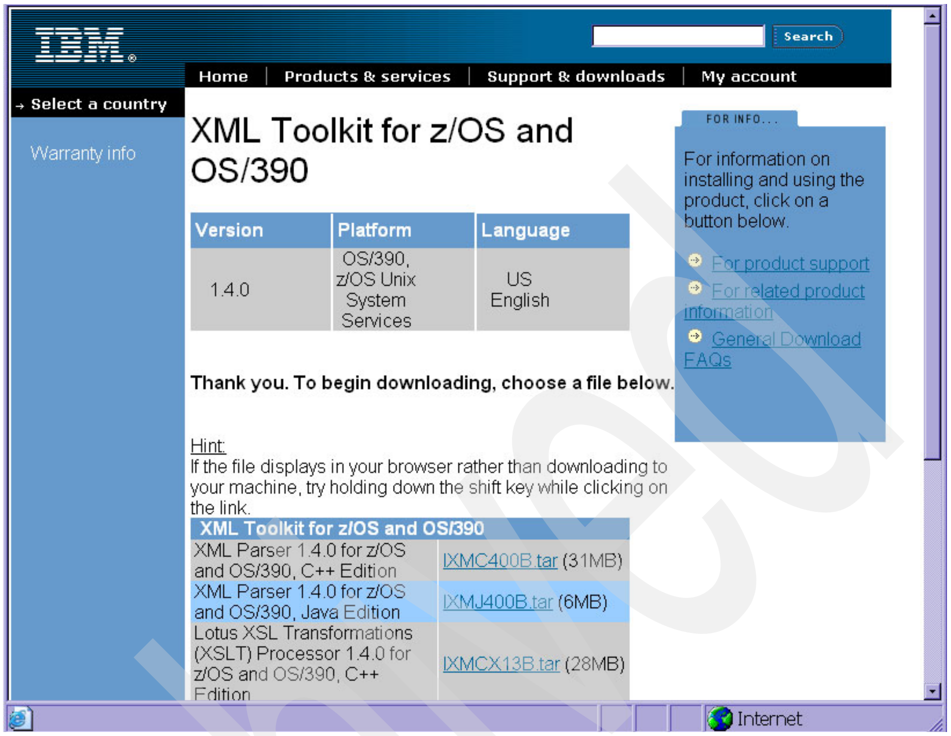
Platform	Version	Media	Language	Availability
OS/390 z/OS Unix System Services	1.4.0	download	US English	Worldwide
OS/390 z/OS Unix System Services	1.3.0	download	US English	Worldwide
OS/390 z/OS Unix System Services	1.2.0	download	US English	Worldwide

For more on XML Toolkit for z/OS and OS/390

Registration/Download Support

Figure 2-1 Toolkit download Web page

Once you have filled out the form with your ID, the files can be downloaded.



XML Toolkit for z/OS and OS/390

Version	Platform	Language
1.4.0	OS/390, z/OS Unix System Services	US English

Thank you. To begin downloading, choose a file below.

Hint:
If the file displays in your browser rather than downloading to your machine, try holding down the shift key while clicking on the link.

XML Toolkit for z/OS and OS/390	
XML Parser 1.4.0 for z/OS and OS/390, C++ Edition	IXMC400B.tar (31MB)
XML Parser 1.4.0 for z/OS and OS/390, Java Edition	IXMJ400B.tar (6MB)
Lotus XSL Transformations (XSLT) Processor 1.4.0 for z/OS and OS/390, C++ Edition	IXMCX13B.tar (28MB)

FOR INFO...

For information on installing and using the product, click on a button below.

For product support
For related product information
General Download FAQs

Figure 2-2 XML Toolkit for z/OS and OS/390 download page

The following table lists the installation files.

Table 2-4 Toolkit V1R4 installation files

XML Toolkit for z/OS and OS/390	SMP/E installable version	Files	Instructions
SMP/E Installable Toolkit (includes all components)	Yes	Toolkit.pax.Z	XMLSMPE.README.txt
XML Parser, C++ Edition only	No	IXMC400B.tar	link
XML Parser, Java Edition only	No	IXMJ400B.tar	link
Lotus® XSLT Processor, C++ Edition only	No	IXMCX13B.tar	link
Lotus XSLT Processor, Java Edition only	No	IXMJX23B.tar	link

Now you are ready to execute the SMP/E dialogs or the JCLs provided to install the toolkit.

As indicated in Table 2-4, when SMP/E installation is invoked the installation of both the XML parser and XSL processor is done. When a non-SMP/E installation is chosen, the user may select the installation of each individual component.

2.3.2 SMP/E Installation

The SMP/E installation process is fully described in the *Program Directory for XML Toolkit for z/OS and OS/390 Version 1 Release 4, Modification Level 0*, GI10-0665-03. The document is available on the Web, at:

`http://www-1.ibm.com/servers/eserver/zseries/software/xml/pdf/v1r4.pdf`

2.3.3 Non-SMP/E Installation

The following sections describe the non-SMP/E installations for each individual component.

XML Toolkit for z/OS and OS/390, C++ Edition

Table 2-5 lists the files downloaded to the workstation and the directories in which they are placed when you issue the **`tar`** command.

Table 2-5 C++ Edition installation files

Component	File name	Directory path
XML Parser	IXMC400B.tar	/usr/lpp/ixm/IBM/xml4c-4_0
XSLT Processor	IXMCX13B.tar	/usr/lpp/ixm/IBM/LotusXSL-C_1_3

V1R4 XML Parser, C++ Edition

Use the following steps to install the XML parser, C++ edition.

1. Download the IXMC400B.tar file to a temporary directory on your workstation.
2. From the TSO command environment invoke the OS/390 shell with the command:

   `OMVS`

3. Check whether the directory /usr/lpp/ixm/IBM exists. If not, create it using the following commands:

   `cd /usr/lpp/`

   `mkdir ixm` (if this directory does not exit)

cd ixm

mkdir IBM (if this directory does not exit)

cd IBM

4. Using FTP, upload the file IXMC400B.tar to /usr/lpp/ixm/IBM in binary format.

   At the Shell prompt `USERID:/usr/lpp/ixm/IBM $` you can issue the **ls** command to check that IXMC400B.tar is in this directory after FTP. You might choose another directory, or the tmp or some other temporary directory if you do not want to keep the tar file after installation.

5. Make sure you are in the directory specified in Step 3, then extract files from the tar archive by entering the command:

   tar -xvozf IXMC400B.tar

   The extract will create an xml4c-4_0 directory with subdirectories for samples, XML documents, and documentation in HTML format. Figure 2-3 shows part of the OS/390 shell screen, where we have entered the **ls** command to see the files created in the xml4c-4_0 directory.

```
JOSE:/usr/lpp/ixm/IBM $ ls
LotusXSL-C_1_3  LotusXSL-J_2_3  xml4c-4_0       xml4j-4_0
JOSE:/usr/lpp/ixm/IBM $ cd xml4c-4_0
JOSE:/usr/lpp/ixm/IBM/xml4c-4_0 $ ls
LICENSE.txt    bin            doc            lib            samples
XLicense.html  credits.txt    include        license.html   version.incl
JOSE:/usr/lpp/ixm/IBM/xml4c-4_0 $
 ===>
                                                                   INPUT
```

Figure 2-3 Files in xml4c-4_0 directory

6. Issue the following command and then edit your .profile file with the **oedit** command and add this line (or command):

   export LIBPATH=/usr/lpp/usr/ixm/IBM/xml4c-4_0

7. Go to ISPF panel 3.4 and allocate 3 datasets with the followings attributes. The high-level qualifier might be different from your userid if RACF® grants, for example SYS1. For this exercise we continue with this qualifier.

   – **SYS1.AIXMMOD1** with 45 cylinders on 3390, Record format:U, Directory blocks:3, Record length: 0, Block size: 32760 and Data set type: PDS

   – **SYS1.SIXMMOD1** with 45 cylinders on 3390, Record format:U, Directory blocks:3, Record length: 0, Block size: 32760 and Data set type: PDS

This is used for the library files required to run the XML Parser for z/OS and OS/390, C++ Edition.

– **SYS1.SIXMEXP** with 2 cylinders on 3390, Record format: FB, Directory blocks:3, Record length: 0, Block size: 12960 and Data set type: PDS

  This dataset is used by the binder to resolve references to functions and variables.

8. Execute the following TSO commands:

   `ogetx '/usr/lpp/ixm/IBM/xml4c-4_0/lib/IBM' 'SYS1.AIXMMOD1'`

   (You can ignore the messages that the EXP and JCLIN subdirectories were not selected.)

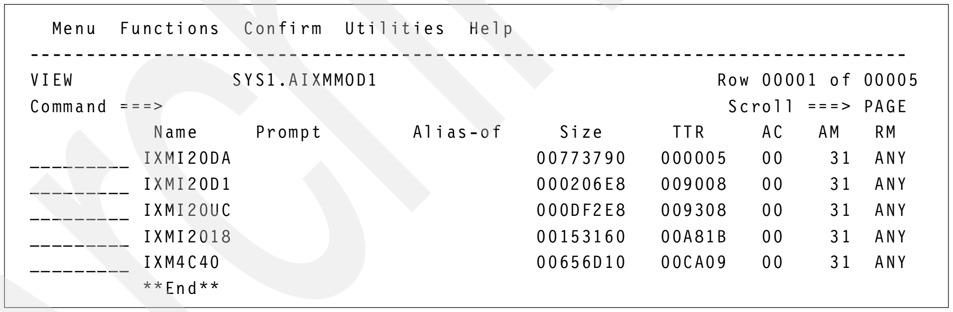
   `ogetx '/usr/lpp/ixm/IBM/xml4c-4_0/lib/IBM/EXP' 'SYS1.SIXMEXP'`

   `ogetx '/usr/lpp/ixm/IBM/xml4c-4_0/lib/IBM/JCLIN/HXML140.XML.JCLIN' 'SYS1.HXML140.XML.JCLIN'`

   Edit the file SYS1.HXML140.XML.JCLIN. This job will link-edit the load modules in the SYS1.AIXMMOD1 dataset into SYS1.SIXMMOD1.

   Watch out! Update the high-level qualifier on the dataset in the AIXMMOD1 and SIXMMOD1 DD statements if you are not using SYS1 as your HLQ.

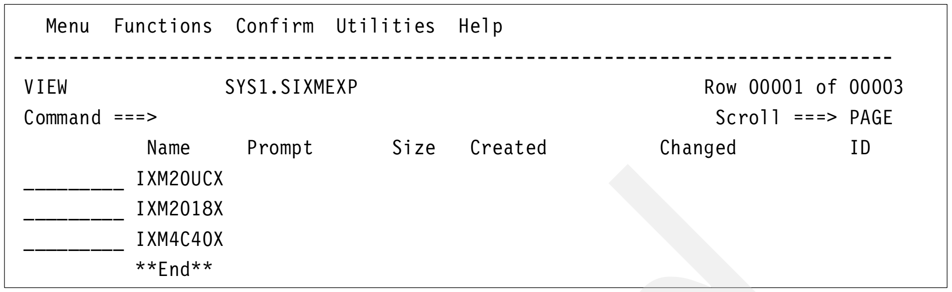
   SYS1.AIXMMOD1 now contains the files shown in Figure 2-4.



```
  Menu  Functions  Confirm  Utilities  Help
------------------------------------------------------------------------------
VIEW             SYS1.AIXMMOD1                             Row 00001 of 00005
Command ===>                                                Scroll ===> PAGE
           Name     Prompt        Alias-of     Size      TTR     AC   AM   RM
_________ IXMI20DA                           00773790   000005   00    31  ANY
_________ IXMI20D1                           000206E8   009008   00    31  ANY
_________ IXMI20UC                           000DF2E8   009308   00    31  ANY
_________ IXMI2018                           00153160   00A81B   00    31  ANY
_________ IXM4C40                            00656D10   00CA09   00    31  ANY
          **End**
```

Figure 2-4 SYS1.AIXMMOD1

SYS1.SIXMEXP now contains the files shown in Figure 2-5.



```
   Menu  Functions  Confirm  Utilities  Help
-------------------------------------------------------------------------------
 VIEW             SYS1.SIXMEXP                                Row 00001 of 00003
 Command ===>                                                  Scroll ===> PAGE
            Name     Prompt       Size   Created          Changed          ID
 _________ IXM2OUCX
 _________ IXM2O18X
 _________ IXM4C40X
           **End**
```

Figure 2-5 SYS1.SIXMEXP

9. Submit the JCL to link-edit the modules into SYS1.SIXMMOD1.

V1R4 XSL Processor, C++ Edition

Some of the steps to perform this task are very similar to the ones used to install the XML parser, C++ edition. Do not repeat unnecessary steps to avoid receiving error messages.

1. Download the IXMCX13B.tar file to a temporary directory on your workstation.
2. From the TSO command environment invoke the OS/390 shell with the command:

   ```
   OMVS
   ```

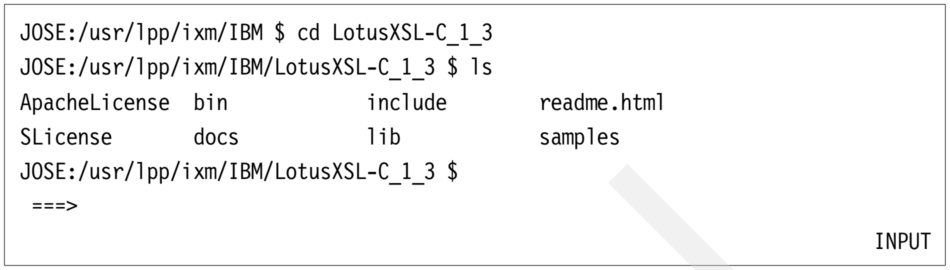
3. Go to the /usr/lpp/ixm/IBM directory. If this directory does not exist, create it using the following commands:

   ```
   cd /usr/lpp/
   mkdir ixm
   cd ixm
   mkdir IBM
   cd IBM
   ```

4. Upload the file IXMCX13B.tar to /usr/lpp/ixm/IBM in binary format (using FTP). After this, at the shell prompt `USERID:/usr/lpp/ixm/IBM $`, issue the **`ls`** command to check that IXMCX13B.tar is in this directory.
5. Make sure you are in the directory defined in Step 2, then extract files from the tar archive by entering the command:

   ```
   tar -xvozf IXMCX13B.tar
   ```

   The extract will create a LotusXSL-C_1_3 directory with subdirectories for samples, XML documents, and documentation in HTML format. Figure 2-6 shows part of the OS/390 shell screen where we have entered the **`ls`** command to see the files created in the LotusXSL-C_1_3 directory.



```
JOSE:/usr/lpp/ixm/IBM $ cd LotusXSL-C_1_3
JOSE:/usr/lpp/ixm/IBM/LotusXSL-C_1_3 $ ls
ApacheLicense  bin            include        readme.html
SLicense       docs           lib            samples
JOSE:/usr/lpp/ixm/IBM/LotusXSL-C_1_3 $
 ===>
                                                                  INPUT
```

Figure 2-6

6. Issue the following command and then edit your .profile file with the **`oedit`** command and add this line (or command):

   `export LIBPATH=/usr/lpp/ixm/IBM/LotusXSL-C_1_3`

7. If you have already allocated the SYS1.AIXMMOD1, SYS1.SIXMMOD and SYS1.SIXMEXP datasets, there is no need to do any more definitions. (If you have not installed the V1R4 XML parser C++ edition yet, then you must allocate the three datasets as described in Step 7 of that installation procedure.)

8. Execute the following TSO commands:

   ```
   ogetx '/usr/lpp/ixm/IBM/LotusXSL-C_1_3/lib/IBM/IXMLC13'
   'SYS1.AIXMMOD1'
   ```

   You can ignore the messages that the EXP and JCLIN subdirectories were not selected.

   Now you can check that module IXMLC13 has been added to the SYS1.AIXMMOD1 dataset, so this dataset has six members now.

   ```
   ogetx '/usr/lpp/ixm/IBM/LotusXSL-C_1_3/lib/IBM/EXP/IXMLC13X'
   'SYS1.SIXMEXP'
   ```

   You can also check that member IXMLC13X has been added to the SYS1.SIXMEXP dataset, so this dataset has four members now.

   ```
   ogetx
   '/usr/lpp/ixm/IBM/LotusXSL-C_1_3/lib/IBM/JCLIN/HXML140.XSLT.JCLIN'
   'SYS1.HXML140.XSLT.JCLIN'
   ```

   Edit the file SYS1.HXML140.XSLT.JCLIN. The job will link-edit the load modules from the SYS1.AIXMMOD1 dataset into SYS1.SIXMMOD1.

   Do not forget to update the high-level qualifier on the dataset names in the AIXMMOD1 and SIXMMOD1 DD statements if you are not using SYS1 as your high-level qualifier.

9. Submit the JCL to link-edit the modules into SYS1.SIXMMOD1.

XML Toolkit for z/OS and OS/390, Java Edition

Table 2-6 lists the tar files downloaded to the workstation and the directories in which the files are placed in when you run the **tar** command.

Table 2-6 Toolkit C++ Edition tar files

Component	File name	Directory path
XML Parser	IXMJ400B.tar	/usr/lpp/ixm/IBM/xml4j-4_0
XSLT Processor	IXMJX23B.tar	/usr/lpp/ixm/IBM/LotusXSL-J_2_3

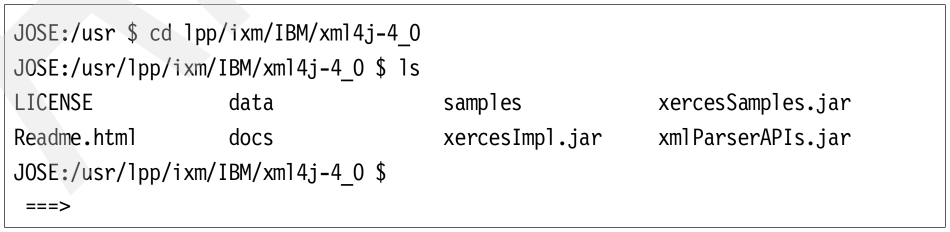
V1R4 XML Parser, Java Edition

To carry out a non-SMP/E installation of the V1R4 XML Parser, Java Edition, follow these steps:

1. Download the file to a temporary directory on your workstation.
2. Upload the IXMJ400B.tar file to z/OS UNIX system services or OS/390 UNIX system services, or to a temporary directory (*tmp*). This upload must be in BINARY format.
3. Issue the following commands. Watch out! The following **mkdir** commands may prompt with the message: `EDC5117I File exists`.

   ```
   cd /usr/lpp
   mkdir ixm
   cd ixm
   mkdir IBM
   cd IBM
   tar -xvozf IXMJ400B.tar
   ```

4. Enter the **ls** command to see the new directory /usr/lpp/ixm/IBM/xml4j-4_0. This is the result of the **tar** command. This directory now contains new subdirectories for samples, XML documents, and documentation in HTML format, as shown in Figure 2-7.



```
JOSE:/usr $ cd lpp/ixm/IBM/xml4j-4_0
JOSE:/usr/lpp/ixm/IBM/xml4j-4_0 $ ls
LICENSE          data             samples          xercesSamples.jar
Readme.html      docs             xercesImpl.jar   xmlParserAPIs.jar
JOSE:/usr/lpp/ixm/IBM/xml4j-4_0 $
 ===>
```

Figure 2-7 /usr/lpp/ixm/IBM/xml4j-4_0

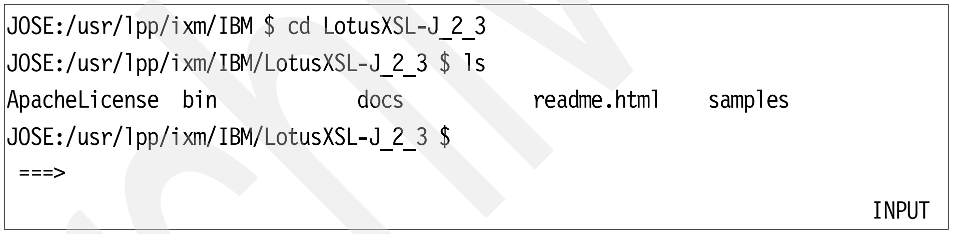
V1R4 XSL Processor, Java Edition

To carry out a non-SMP/E installation of the V1R4 XSL Processor, Java Edition:

1. Download the IXMJX23B.tar file to a temporary directory on your workstation.
2. Upload this IXMJX23B.tar file to z/OS UNIX System Services to a temporary directory (*tmp*). This upload must be in BINARY format. If you want to keep this tar file, choose a directory other than tmp.
3. Issue the following commands. Watch out! The following **`mkdir`** commands may prompt with the message: `EDC5117I File exists`.

   ```
   cd /usr/lpp
   mkdir ixm
   cd ixm
   mkdir IBM
   cd IBM
   tar -xvozf IXMJX23B.tar
   ```

4. Enter the **`ls`** command to see the new directory that is the result of the **`tar`** command: /usr/lpp/ixm/IBM/LotusXSL-J_2_3. This directory now contains new subdirectories for samples, XML documents, and documentation in HTML format, as shown in Figure 2-8.



```
JOSE:/usr/lpp/ixm/IBM $ cd LotusXSL-J_2_3
JOSE:/usr/lpp/ixm/IBM/LotusXSL-J_2_3 $ ls
ApacheLicense  bin          docs          readme.html    samples
JOSE:/usr/lpp/ixm/IBM/LotusXSL-J_2_3 $
 ===>
                                                                   INPUT
```

Figure 2-8 /usr/lpp/ixm/IBM/LotusXSL-J_2_3

2.4 Runtime considerations

2.4.1 XML parsers and code page conversion

Programs executing on z/OS and OS/390 and having something to do with XML deserve special mention because of the code page considerations on z/OS and OS/390.

The character encoding scheme used in z/OS and OS/390 is EBCDIC. There are hundreds of different encoding systems for assigning each character a binary number, a combination of bits. Some of these encoding schemes are based on a single byte per character (SBCS) and others use two bytes per character (DBCS). All this encoding have been appearing as the computer science world has gone forward for the last 20 years. EBCDIC was used on the successful System/360™ and survived for many years despite the almost universal adoption of ASCII elsewhere. EBCDIC and ASCII are both SBCSs.

In this section we discuss some characteristics of the encoding schemes that are related to z/OS and OS/390 performance considerations.

- ASCII, the American Standard Code for Information Interchange, is used in almost all present-day computers. It is a character set that uses 7-bit units to convey some control codes, space, numbers, most basic punctuation, and unaccented letters a-z and A-Z. It is the most important character set out there, despite its limitation. ASCII provides only 128 numeric values, and 33 of those are reserved for special functions.
- EBCDIC, the Extended Binary-Coded Decimal Interchange Code, is an encoding format using 8-bit units. It uses more or less the same characters as ASCII. It has non-contiguous letter sequences, some ASCII characters do not exist in EBCDIC (for example, square brackets), and EBCDIC has some characters that are not in ASCII. As a consequence, the translation between ASCII and EBCDIC was never officially completely defined. Many different encoding schemes were created over time to fulfill the needs of different Western languages. These are the different “code pages.” In z/OS and OS/390, the default is code page Cp1047.
- ISO 8859-1. This standard is part of the ISO 8859 family of standards. It defines a character repertoire identified as “Latin alphabet No. 1,” commonly called “ISO Latin 1,” as well as a character code for it. It is actually a direct superset of US-ASCII (the repertoire contains the ASCII repertoire as a subset and the code numbers for those characters that are the same as in ASCII). The standard also specifies an encoding, which is similar to that of ASCII: each code number is presented simply as one octet. ISO 8859-1 contains various accented characters and other letters needed for writing the languages of Western Europe, and some special characters.

This list of character encoding schemes could go on and on since there are hundreds of standards, as well as variations and modern versions of some. These encoding systems also conflict with one another. That is, two encodings can use the same number (binary number) for two *different* characters, or use different numbers for the *same* character.

- **Unicode** provides a unique character set for every character, no matter what the platform, no matter what the program, no matter what the language. Unicode is a character encoding that provides more character properties and algorithm descriptions, while ISO standards define collections, subsets, and so on. Unicode has been designed for the best interoperability with both ASCII and ISO 8859-1, the most widely used character sets, to make it easier for Unicode to be used in applications and protocols.

The various Unicode encoding forms are optimized for a variety of different uses:

– UTF-16, the default encoding form, maps a character code point to either one or two 16-bit integers.
– UTF-8 is a byte-based encoding that offers backwards compatibility with ASCII-based, byte-oriented APIs and protocols. A character is stored with 1, 2, 3, or 4 bytes.
– UTF-32 is the simplest but most memory-intensive encoding form; it uses one 32-bit integer per Unicode character.
– SCSU is an encoding scheme that provides a simple compression of Unicode text. It is designed only for input and output, not for internal use.

When a Java XML Parser has to parse an XML document that is not already encoded in Unicode, the document must be converted from its encoding schema to Unicode. The XML parser will convert the document to Unicode before parsing. When the XML document is not encoded in UTF-8 or UTF-16, you must specify the correct encoding in the XML processing instructions by including an encoding statement like the following:

```
<?xml version="1.0" encoding="ebcdic-cp-us"?>
```

In this case the encoding of the XML document is neither UTF-8 or UTF-16, so the explicit specification via the `encoding` attribute is done on the XML processing instruction.

From the point of view of a C++ XML parser, there is also support for internationalization. XML4C (the base of XML Parser, C++ Edition) integrates the Xerces-C parser with IBM's International Components for Unicode (ICU) and extends the number of encodings supported to over 150.

ICU enables you to write language-independent C and C++ code that is used on separate, localized resources to get language-specific results. ICU supports many features, including language-sensitive text, dates, time, numbers, currency, and so on. ICU provides conversion basis: A converter is used to transform text from one encoding type to another. In the case of Unicode, ICU transforms text from one encoding codepage to Unicode and back. An encoding is a mapping from a given character set definition to the actual bits used to represent the data.

The XML Parser for z/OS and OS/390, C++ Edition V1R4 has been tested with ICU version 2.0.2. The required ICU level is packaged with the corresponding parser and Xalan level. For the XML Toolkit for z/OS and OS/390 V1R4, all of the supported encodings are listed in the file shipped with ICU called "`convrtrs.txt`" that you can find in the HFS.

On z/OS or OS/390, avoiding conversion prior to calling the XML parser is the best approach, whenever possible. If you do a conversion beforehand, you may be doing the conversion twice and wasting CPU. Furthermore, performing a conversion prior to calling the parser might create a discrepancy between the resulting data encoding and the encoding attribute in the processing instruction of the document, provoking the parser to produce an unpredictable result.

2.4.2 Multiple parser initialization

A considerable amount of CPU time is used for parser initialization. You should, therefore, avoid instantiating a new parser every time you parse. Instead, create the parser once, and reuse the parser instance. From the point of view of Java programs, this is a programmer's task. It is a very good idea to maintain a pool of parsers that can serve multiple users simultaneously. But the implementation of object pooling is a Java program itself.

2.4.3 XML document recommendations

Validation is a process that should be avoided when possible. CPU time increases significantly when validation is done. If you decide to turn validation off, XML documents must not include the DOCTYPE clause because this version of the parser always incurs the cost of reading the DTD if this clause is included in the document, even when not validating.

If you use DTD validation, try not to assign default values for attributes. The process is a much slower one.

We have presented in detail the advantages of schema validation versus DTD validation. Schemas provide extensive capability for validation. However, DTD validation is more efficient than schemas when performing equivalent validating tasks.

3

XML Toolkit samples

In this chapter we discuss the samples provided with the XML Toolkit for z/OS and OS/390 V1R4. We provide short descriptions of the samples and identify the configuration steps necessary to run them.

The following samples are discussed:

- Java samples:
  DOM, SAX, and Socket samples
- C/C++ XML parser samples, including running them in z/OS or OS/390 UNIX System Services environments as well as in traditional z/OS or OS/390 environments
- C/C++ XSLT processor samples, including running the samples in z/OS or OS/390 UNIX System Services as well as in traditional z/OS or OS/390 environments

3.1 Java samples

In this section we identify the Java samples provided with the XML Toolkit for z/OS and OS/390, and we demonstrate the features of the XML Parser for z/OS and OS/390, using SAX and DOM APIs.

To find the samples examine the path in which you have installed the Toolkit, commonly /usr/lpp/ixm.Locate the directory IBM/xml4j-4_0/samples; this is where you will find the Java samples provided with the Toolkit.

In this document, we assume that the installation directory is /usr/lpp/ixm. If this is not the case for your installation, adjust accordingly.

The samples that come with the toolkit are the following:

- DOM Samples from package 'dom'

Counter	Displays the time and count of elements, attributes, spaces, and characters in an XML file
GetElementsByTagName	Parses an XML document searching for specific elements by name
Writer	Parses an XML document and prints it
ASBuilder	Preparses XML schema documents and validates instance documents against preparsed schema grammars

- SAX Samples from package 'sax'

Counter	Displays the time and count of elements, attributes, spaces, and characters in an XML file.
DocumentTracer	Provides a trace of SAX2 events for files parsed
Writer	Parses an XML document and prints it

- Socket Samples from package 'socket'

DelayedInput	Delays the input to the SAX parser to simulate reading data from a socket.
KeepSocketOpen	Wraps both the input and output stream on both ends of a socket

We are going to examine some representative cases of these samples. All Java-related contents are in directory IBM/xml4j-4_0. For the purpose of this discussion, we assume that you installed the samples in the same directory.

3.1.1 Setting up the samples

Perform the following steps to set up the appropriate environment configuration:

1. Go to the shell prompt in TSO OMVS, and edit your personal profile file:

   ```
   oedit $HOME/.profile
   ```

   $HOME is the variable containing your user home path, for example, /u/david.

2. Add the following lines:

   ```
   export XERCESJROOT=/usr/lpp/ixm/IBM/xml4j-4_0
   ```

   Use the full path to jar files where all needed classes for XML parsing are located.

   ```
   CLASSPATH=.:$XERCESJROOT/xercesImpl.jar:$CLASSPATH
   CLASSPATH=$XERCESJROOT/xmlParserAPIs.jar:$CLASSPATH
   CLASSPATH=$XERCESJROOT/xercesSamples.jar:$CLASSPATH
   export CLASSPATH
   ```

   It is a very good practice to organize your *.profile* file into sections, so you can dedicate a section to Java, its CLASSPATH, and related contents.

3. Exit and re-enter the shell, so that updates on your *.profile* file take effect. You can also do this with the **/bin/sh -L** command.

At this point, you have all the necessary classes on your CLASSPATH, so you can invoke them in a native way from the shell prompt. The xerces parser classes are in xercesImpl.jar; the APIs implemented by the parser are in xmlParserAPIs.jar. To determine if your definitions are correct, you can check the XML4J version in use with following:

- From shell prompt, type:

  ```
  java org.apache.xerces.impl.Version
  ```

- If it is correct, JVM can find this class using your CLASSPATH. It loads the class and returns the message:

  ```
  XML4J 4.0.2
  ```

  This indicates the XML4J level that is in use.

We need some more settings to use Java XSLT Processor for z/OS and OS/390. We need to add xalan.jar classes, the processor class files xml-apis.jar containing the standard APIs implemented by the parser, and bsf.jar, required to test Xalan-Java extensions. So, if you want to use a Java XSLT Processor, enter the following commands:

```
export XALANJROOT=/usr/lpp/ixm/IBM/LotusXSL-J_2_3
export CLASSPATH=$CLASSPATH:$XALANJROOT/bin/bsf.jar
export CLASSPATH=$CLASSPATH:$XALANJROOT/bin/xml-apis.jar
export CLASSPATH=$CLASSPATH:$XALANJROOT/bin/xalan.jar
```

Output for some samples is not displayable because the samples use UTF-8 format, and UNIX System Services for z/OS or OS/390 doesn't support it. In those cases, you can redirect output to a file, and then you can display this file. Redirections in UNIX can be done in the form: **`command1 > output.txt`** for standard output, or **`command1 2> errors.txt`** for standard error output.

3.1.2 DOM samples

ASBuilder

The ASBuilder sample pre-parses XML schema documents and validates instance documents against pre-parsed schema grammars. It can be used to check if your new schemas are well formed, and to check if your new XML documents are valid for these schemas. You can think about it as a DOM error handler skeleton more than a DOM parser: you can use it as a basis for your new error handlers, and it shows you the features you need to activate to do XML validating with schemas.

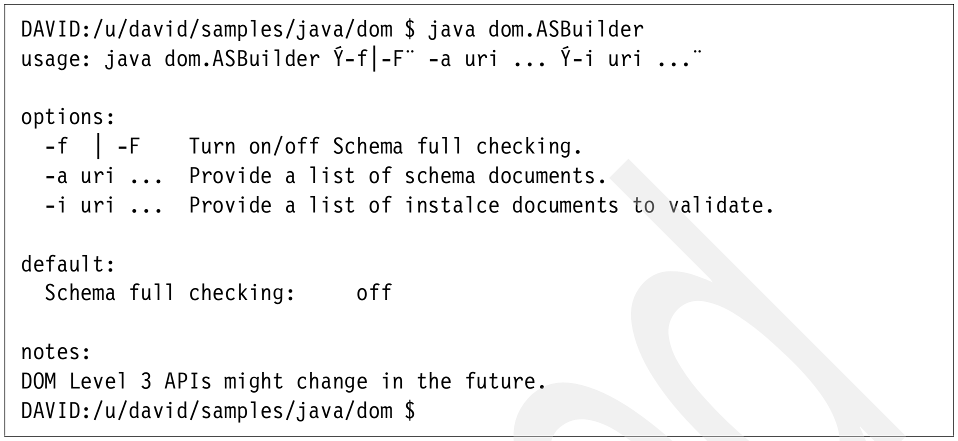
You will probably want to update the original file at some point. So, to preserve it, we have copied the toolkit sample from its original directory to a new one created for this purpose. We did this with the following sequence of commands:

```
mkdir /u/david/samples (where david was our userid)
mkdir /u/david/samples/java
mkdir /u/david/samples/java/dom
cp /usr/lpp/ixm/IBM/xml4j-4_0/samples/dom/ASBuilder.java
        /u/david/samples/java/dom/ASBuilder.java
```

We can now work with our new /u/david/samples/java/dom/ASBuilder.java file, making changes to the source code while still preserving the original version. The best way to do the updates is to rename the Java class because in your CLASSPATH you have an ASBuilder class already, and it is a good idea to preserve it. Anyway, here we are going to work with the original version of the ASBuilder class. We can test whether our CLASSPATH variable is correct and has this class invoking it without any parameter. We do this by entering the following command:

```
java dom.ASBuilder
```

If we see ASBuilder source, we can find a package declaration that points to dom, so this is the reason we have to enter dom.ASBuilder. The result is shown in Figure 3-1.



```
DAVID:/u/david/samples/java/dom $ java dom.ASBuilder
usage: java dom.ASBuilder Ý-f|-F¨ -a uri ... Ý-i uri ...¨

options:
  -f  | -F    Turn on/off Schema full checking.
  -a uri ...  Provide a list of schema documents.
  -i uri ...  Provide a list of instalce documents to validate.

default:
  Schema full checking:     off

notes:
DOM Level 3 APIs might change in the future.
DAVID:/u/david/samples/java/dom $
```

Figure 3-1 dom.ASBuilder

With parameter *-f* you can activate a feature called `http://apache.org/xml/features/validation/schema-full-checking`; this is the special URI for the feature. When turned on, this feature enables Schema grammar checking for additional errors that are time-consuming or memory-intensive. It does not affect the checking level performed on document instances that use Schema grammars. Parameter *-F* turns off this checking level. By default this feature is off since it affects performance.

This class waits for at least two parameters. If you write fewer than two parameters, the program shows you the previous screen again. We are going to make a file with a sample XML Schema to test this DOM parser, so we can use it as the second parameter. To do this, we create a different directory where we create a new file:

```
mkdir /u/david/samples/java/schemas
oedit /u/david/samples/java/schemas/zOSXML.xsd
```

in this file we have entered the contents shown in Example 3-1.

Example 3-1 our sample XML Schema

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

<schema targetNamespace="http://www.redbooks.ibm.com/"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:zOSXML="http://www.redbooks.ibm.com/" elementFormDefault="qualified”>

    <element name="Book" type="zOSXML:BookType" />

    <complexType name="BookType">
```

```
        <sequence>
            <element name="Title" type="string"/>
            <element name="Contents" type="zOSXML:ContentsType" />
            <element name="Copyright" type="string" />
        </sequence>
    </complexType>

    <complexType name="ContentsType">
        <sequence>
            <element name="Chapter" maxOccurs="unbounded">
                <complexType>
                    <sequence>
                        <element name="Heading" type="string" minOccurs=”0” />
                        <element name="Topic" maxOccurs="unbounded">
                            <complexType>
                                <simpleContent>
                                    <extension base="string">
                                        <attribute name="subSections" type=”integer” />
                                    </extension>
                                </simpleContent>
                            </complexType>
                        </element>
                    </sequence>
                    <attribute name="focus" default="Java">
                        <simpleType>
                            <restriction base="string">
                                <enumeration value="Java"></enumeration>
                                <enumeration value="XML"></enumeration>
                            </restriction>
                        </simpleType>
                    </attribute>
                </complexType>
            </element>
            <element name="SectionBreak" minOccurs="0" maxOccurs="unbounded”>
            </element>
        </sequence>
    </complexType>
</schema>
```

This is a schema to validate XML documents describing a list of redbooks with XML-related contents. First, we validate this schema to determine if it is correct by issuing the command:

```
java dom.ASBuilder -a /u/david/samples/java/schemas/JavaXML.xsd
```

It is very useful to pre-parse our schemas before we put them in a production environment. If you browse the ASBuilder.java file, you are going to find the section shown in Example 3-2 in the main method. For simplicity, we have omitted non-relevant parts of the code from this example.

Example 3-2 ASBuilder.java

```
public static void main(String argv[]) {

     // too fee parameters
     if (argv.length < 2) {
         printUsage();
         System.exit(1);
     }
     // get DOM implementation
     DOMImplementationAS domImpl =
(DOMImplementationAS)DOMImplementationImpl.getDOMImplementation();
     // create a new parser, and set the error handler
     DOMASBuilder parser = domImpl.createDOMASBuilder();
     parser.setErrorHandler(new ASBuilder());

       ..................................
ASModel asmodel = null;
for (i = 0; i < asfiles.size(); i++) {
    asmodel = parser.parseASURI((String)asfiles.elementAt(i)
    parser.setAbstractSchema(asmodel);
}
```

In this code, *parser* is a variable of type DOMASBuilder, declared at the beginning of this method. Its declaration is a little complex, because it uses the createDOMASBuilder() method of object DOMImplementationAS to obtain an instance of this parser.

We use this type of declaration many times in parsers, because it gives a level of transparency that prevents us as developers from having to update our code if DOMASBuilder changes its implementation (that is, we change the provider).

With getDOMImplementation() we get the current implementation in use for DOM parser (Apache implementation) and we store it in the *domImpl* variable. Next we use this variable (class object) to invoke its method createDOMASBuilder(); this creates a new parser instance. Once we have *parser* pointing to our new DOM parser instance, we call its setErrorHandler() method, giving our own ASBuilder() class as parameter, so our class is going to be a DOM parser error handler. You can see the following line at the beginning of the class:

```
public class ASBuilder implements DOMErrorHandler { ......
```

This line indicates that the class is going to implement generic DOMErrorHandler, which was our purpose.

In *asfiles* we have a vector with URIs entered as parameters. In the previous example we are parsing these URIs. We are using as our schema the first URI (with parameter -a), it is also going to be the first element in the vector.

If all the previous steps are correct, you are not going to have any answer after this command, because if you check ASBuilder source code, when you only enter the URI for the schema, it parses this schema (parser.setAbstractSchema(asmodel)), and if there are no errors, the error handler (method handleError() in this java class) is not going to be invoked.

If you make a mistake on the source file of the schema, you can see the output handled by method handleError(). (Try to change one character, to eliminate an element, or the like, and run the sample again.) This method is where we control different types of errors: Warning, Error and FatalError. Here all error types stop the process, but commonly Warning errors only would show a message and the process continues. Anyway, remember that this is only an error handler skeleton, and that the final decision about the way of handling errors is up to you.

Now we validate an XML instance with the schema shown in Example 3-1. For this purpose, we created the XML document shown in Example 3-3, compliant with our schema.

Example 3-3 contents.xml

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

<!-- Java and XML -->
<zOSXML:Book xmlns:zOSXML="http://www.redbooks.ibm.com/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.redbooks.ibm.com/
      /u/david/samples/java/schemas/zOSXML.xsd”>

 <zOSXML:Title>Java and XML</zOSXML:Title>
 <zOSXML:Contents>

  <zOSXML:Chapter focus="XML">
   <zOSXML:Heading>Introduction</zOSXML:Heading>
   <zOSXML:Topic subSections="7">What Is It?</zOSXML:Topic>
   <zOSXML:Topic subSections="3">How Do I Use It?</zOSXML:Topic>
   <zOSXML:Topic subSections="4">Why Should I Use It?</zOSXML:Topic>
   <zOSXML:Topic subSections="0">What's Next?</zOSXML:Topic>
  </zOSXML:Chapter>

  <zOSXML:Chapter focus="XML">
   <zOSXML:Heading>Creating XML</zOSXML:Heading>
   <zOSXML:Topic subSections="0">An XML Document</zOSXML:Topic>
   <zOSXML:Topic subSections="2">The Header</zOSXML:Topic>
   <zOSXML:Topic subSections="6">The Content</zOSXML:Topic>
   <zOSXML:Topic subSections="1">What's Next?</zOSXML:Topic>
  </zOSXML:Chapter>

  <zOSXML:Chapter focus="Java">
   <zOSXML:Heading>Parsing XML</zOSXML:Heading>
   <zOSXML:Topic subSections="3">Getting Prepared</zOSXML:Topic>
   <zOSXML:Topic subSections="3">SAX Readers</zOSXML:Topic>
   <zOSXML:Topic subSections="9">Content Handlers</zOSXML:Topic>
   <zOSXML:Topic subSections="4">Error Handlers</zOSXML:Topic>
   <zOSXML:Topic subSections="0">
     A Better Way to Load a Parser
   </zOSXML:Topic>
   <zOSXML:Topic subSections="4">"Gotcha!"</zOSXML:Topic>
   <zOSXML:Topic subSections="0">What's Next?</zOSXML:Topic>
  </zOSXML:Chapter>

  <zOSXML:SectionBreak/>

 </zOSXML:Contents>

 <zOSXML:Copyright>IBM</zOSXML:Copyright>

</zOSXML:Book>
```

We called the document contents.xml, and we stored it in directory /u/david/samples/java/xml (a new directory we created). Now, to check if our XML document is valid and conforms to our defined schema, we enter the following command:

```
cd /u/david/samples/java/dom
java dom.ASBuilder -a ../schemas/zOSXML.xsd -i ../xml/contents.xml
```

We get no answer, so our document is valid. Now we edit our contents.xml file, and we eliminate line **`<zOSXML:Copyright>IBM</zOSXML:Copyright>`** from it. We enter the previous command once again, to check the validity of our document.

The response we get is:

```
[Error: contents.xml:57:15: cvc-complex-type.2.4.b: The content of element
'zOSXML:Book' is not complete. It must match
'((("http://www.redbooks.ibm.com/":Title),("http://www.redbooks.ibm.com/":C
ontents)),("http://www.redbooks.ibm.com/":Copyright))'.
```

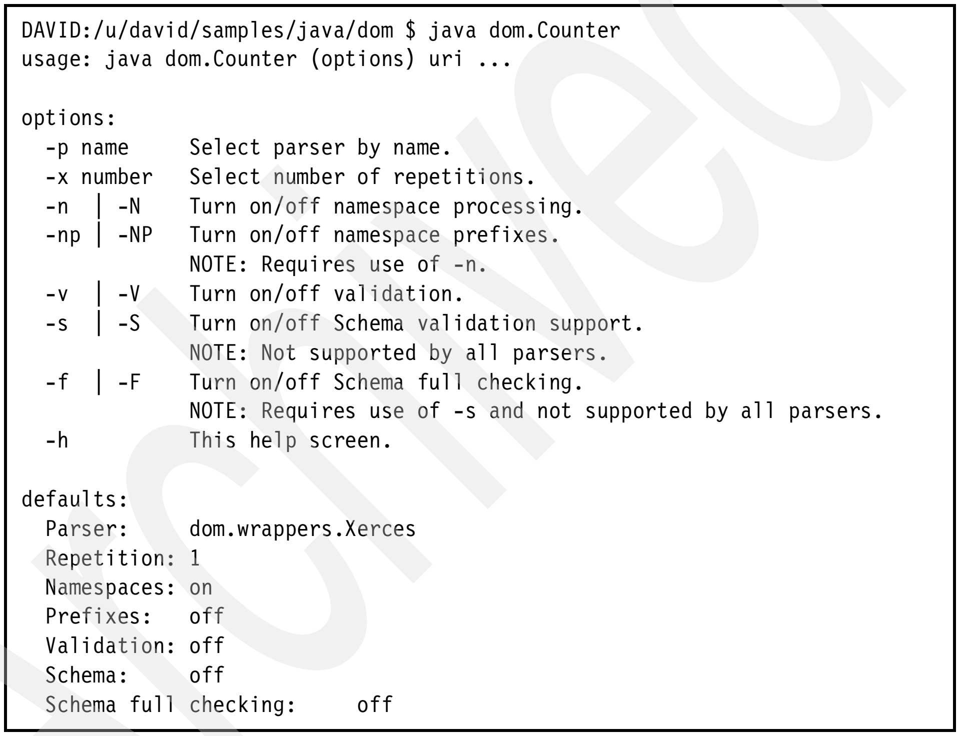
This is because we have defined in our schema that we need at least one *Copyright* element on our XML document.

Counter

In this sample, we traverse the DOM tree to show the time and count of elements, attributes, ignorable whitespaces, and characters in the XML document. We follow a process similar to that described for the previous sample, so first we copy the original file /usr/lpp/ixm/IBM/xml4j-4_0/samples/dom/Counter to the directory /u/david/samples/java/dom. To check that the Counter class is found, we invoke it without any parameter:

```
java dom.Counter
```

If everything is correct, the screen shown in Figure 3-2 is returned.



```
DAVID:/u/david/samples/java/dom $ java dom.Counter
usage: java dom.Counter (options) uri ...

options:
  -p name     Select parser by name.
  -x number   Select number of repetitions.
  -n  | -N    Turn on/off namespace processing.
  -np | -NP   Turn on/off namespace prefixes.
              NOTE: Requires use of -n.
  -v  | -V    Turn on/off validation.
  -s  | -S    Turn on/off Schema validation support.
              NOTE: Not supported by all parsers.
  -f  | -F    Turn on/off Schema full checking.
              NOTE: Requires use of -s and not supported by all parsers.
  -h          This help screen.

defaults:
  Parser:     dom.wrappers.Xerces
  Repetition: 1
  Namespaces: on
  Prefixes:   off
  Validation: off
  Schema:     off
  Schema full checking:     off
```

Figure 3-2   dom.Counter

With parameter *-p* you can select the parser implementation you are going to use. The following line is in the main method of the Counter.java source file:

```
parser = (ParserWrapper)Class.forName(parserName).newInstance();
```

Class is the generic class from package java.lang, which is very useful in the dynamic loading of classes. It has a method called forName(); what it does is load the class specified as parameter to this method. For example, if you enter *dom.wrapper.Xerces* as parameter, this method is going to use xerces DOM Implementation. This is the value that the parameter has by default.

ParserWrapper is an abstract interface that represents the DOM parser implementation (it only defines methods interfaces), so what we're saying in the identified line is that Class is casted to the ParserWrapper class, so it has all the interfaces necessary for a DOM parser, and that the final implementation of this abstract class is the one specified in parameter *parserName* (by default, *dom.wrapper.Xerces*). So in the future, if you change your DOM implementation, you could use parameter -p to specify the new one.

Parameter *-x* lets you specify the number of times you want to repeat the parsing process, to have more values to obtain the final average. For example, if you say *-x 3*, the parsing process is repeated three times. It calculates the time spent on each parsing, and the result is accumulated in a total variable. At the end, this variable is divided by the number of repetitions (three in our example), which is the average time spent on parsing.

With option *-n* or *-N*, you can activate or deactivate namespace processing. This is done with a feature you can activate with the following line in your Java code:

```
parser.setFeature("http://xml.org/sax/features/namespaces", true);
```

If it is *true*, this feature specifies that prefixes on elements in the XML document will be replaced by their corresponding URIs, as you code in *xmlns* associations at the beginning of the document. Be very careful with this setting: if you set it to true, your grammar (schema) has to go through namespace processing. By default this feature is true.

Parameter *-np* is not currently implemented; you will not find it anywhere in the code.

Parameter *-v* or *-V* is used to activate or deactivate the following feature:

```
parser.setFeature("http://xml.org/sax/features/validation", true);
```

When you activate it, the XML document must specify a grammar. By default validation is against a DTD document, but you can use a schema as well by activating the next feature.

Use option *-s* or *-S* to activate or deactivate schema validation support. This feature is as follows:

```
parser.setFeature("http://apache.org/xml/features/validation/schema",
true);
```

With this feature on, you can use your schema as a validation grammar.

Schema full checking, which may be expensive in terms of time and memory, can be activated with option *-f*, or deactivated with *-F.* The feature is defined as follows:

```
parser.setFeature("http://apache.org/xml/features/validation/schema-full-ch
ecking", true)
```

With this option, particle-unique attribution constraint checking and particle derivation restriction checking are controlled, but these are the only new tests you can obtain.

Now you want to run a full sample, using the XML document shown in example 3-3 on page 56 and the schema in Example 3-1 on page 53. Enter the following command:

```
java dom.Counter -x 6 -v -s ../xml/contents.xml
```

The answer you should get is:

```
../xml/contents.xml: 5230/6=871;15;1 ms (26 elems, 21 attrs, 0 spaces, 386
chars)
```

The next test is to update your zOSXML.xsd file, writing *Look* instead of *Book*, so your grammar is going to wait for a *Look* element at the beginning of the document. Obviously, parse is going to fail, because your xml document starts with *Book*. In UNIX, errors are written as standard output, so enter the following command:

```
java dom.Counter -x 6 -v -s ../xml/contents.xml 2> /u/david/docs/output.txt
```

After running this command, the output you should see (standard output) is:

```
../xml/contents.xml: 5887/6=981;15;1 ms (26 elems, 21 attrs, 0 spaces, 386
chars)
```

But there are error messages on /u/david/docs/output.txt, so browse it with the **obrowse** command, and you will see the following text:

```
[Error] contents.xml:6:96: cvc-elt.1: Cannot find the declaration of
element 'zOSXML:Heading'.
```

In the lab, this was only the first of many messages we got. It was telling us that it was looking for the *zOSXML:Heading* element at the beginning of our XML document, and it could not find it.

3.1.3 SAX samples

DocumentTracer

This program makes a trace of the XML document; it is a good starting point to make your own parser because you can use it as a skeleton. Again, the recommendation is to copy the original source file /usr/lpp/ixm/IBM/xml4j-4_0/samples/sax/DocumentTracer to /u/david/samples/java/sax, after creating a new directory sax for these samples.

First, test whether the class is accessible by entering the command:

```
java sax.DocumentTracer
```

The result is shown in Figure 3-3.

```
DAVID:/u/david/samples/java/dom $ java sax.DocumentTracer
usage: java sax.DocumentTracer (options) uri ...

options:
  -p name  Select parser by name.
  -n | -N  Turn on/off namespace processing.
  -v | -V  Turn on/off validation.
  -s | -S  Turn on/off Schema validation support.
           NOTE: Not supported by all parsers.
  -f | -F  Turn on/off Schema full checking.
           NOTE: Requires use of -s and not supported by all parsers.
  -h       This help screen.

defaults:
  Parser:     org.apache.xerces.parsers.SAXParser
  Namespaces: on
  Validation: off
  Schema:     off
  Schema full checking:     off
```

Figure 3-3 java sax.DocumentTracer

Use parameter *-p name* to specify the SAX parser implementation in use. The default value is org.apache.xerces.parsers.SAXParser. It is the most popular implementation but you could choose another. In this case, the implementation is loaded with the following instructions:

```
parser = XMLReaderFactory.createXMLReader(parserName);
     .....................................
parser sax1Parser = ParserFactory.makeParser(parserName);
parser = new ParserAdapter(sax1Parser);
     .....................................
```

`XMLReaderFactory` is a class in the org.xml.sax.helpers package that defines a method `createXMLReader()` to instantiate this implementation.

`XMLReaderFactory` contains static methods to create an XML reader based on the system property `org.xml.sax.driver` as the full name for the XML Reader Java class. The *parserName* is a variable containing the name of the implementation, `org.apache.xerces.parsers.SAXParser` is the default value, and `XMLReaderFactory` is going to try to instantiate it.

`ParserFactory` is a deprecated class which dynamically loads SAX 1.0 parsers. This class is designed to work with SAX 1.0 implementations. The reason to have it here is that this part of the code is going to be executed only if we get an exception instantiating with XMLReaderFactory (SAX 2), then we can test if the problem is that our parser is a SAX 1.0 parser. In this case we show a warning advising about this (some features are not available in a SAX 1 parser), and the process continues instantiating with *ParserAdapter* (the class that involves the SAX 1 parser). So the last two lines are executed only if SAX 2 instantiating fails.

The four sets of parameters -n and -N, -v and -V, -s and -S, -f and -F, have the same meaning that we discussed for the DOM samples "Counter" on page 58:

-n and *-N* turn namespace processing on and off
-v and *-V* turn the validation feature on and off
-s and *-S* turn schema validation on and off
-f and *-F* turn schema full checking on and off

One execution of this sample is the following:

```
java sax.DocumentTracer -v -s ../xml/contents.xml > /u/david/docs/output.txt
```

If we do an **obrowse** to file output.txt, we see the result shown in Example 3-4. We only show here a part of the output, because it is very long.

Example 3-4 java sax.DocumentTracer output

```
setDocumentLocator(locator=org.apache.xerces.parsers.AbstractSAXParser$LocatorProxy@81ad6bc)
startDocument()
 comment(text=" Java and XML ")
 startPrefixMapping(prefix="zOSXML",uri="http://www.redbooks.ibm.com/")
 startPrefixMapping(prefix="xsi",uri="http://www.w3.org/2001/XMLSchema-instance")

startElement(uri="http://www.redbooks.ibm.com/",localName="Book",qname="zOSXML:Book",attributes={{uri="h
ttp://www.w3.org/2001/XMLSchema-instance",localName="schemaLocation",qname="xsi:schemaLocation",type="CD
ATA",value="http://www.redbooks.ibm.com/ /u/david/samples/java/schemas/zOSXML.xsd"}})
  characters(text="\n\n ")
  startElement(uri="http://www.redbooks.ibm.com/",localName="Title",qname="zOSXML:Title",attributes={})
   characters(text="Java and XML")
  endElement(uri="http://www.redbooks.ibm.com/",localName="Title",qname="zOSXML:Title")
  characters(text="\n ")
```

```
startElement(uri="http://www.redbooks.ibm.com/",localName="Contents",qname="zOSXML:Contents",attributes=
{})
   characters(text="\n\n  ")

startElement(uri="http://www.redbooks.ibm.com/",localName="Chapter",qname="zOSXML:Chapter",attributes={{
uri="",localName="focus",qname="focus",type="CDATA",value="XML"}})
    characters(text="\n   ")

startElement(uri="http://www.redbooks.ibm.com/",localName="Heading",qname="zOSXML:Heading",attributes={}
)
     characters(text="Introduction")
    endElement(uri="http://www.redbooks.ibm.com/",localName="Heading",qname="zOSXML:Heading")
    characters(text="\n   ")
       ..............................................................
```

As you see, it is a trace of the SAX 2 API's flow, so you can use this parser as a skeleton to make your own parser. The only thing you have to do is replace the code for the APIs you don't need with your own code.

3.1.4 Socket samples

These samples show the possibility of making an XML parser that uses a socket stream for the transmission of XML documents. The problem is that because XML documents have no explicit end-of-document, you have to close the socket end point in your code. The sample shows you how to solve this problem.

KeepSocketOpen

This sample tries to make a parser receive multiple XML documents by the same open socket. This presents a problem because the parser could be receiving many root elements (from the many documents it receives), and this is forbidden in XML.

The sample is also going to use this socket to receive different types of data; the problem with this is that it buffers the input stream in specific block sizes, and because of this it could read beyond the end of the data.

First we test that this class is found by invoking it:

```
java socket.KeepOpenSocket
```

The answer must be:

```
usage: java socket.KeepSocketOpen file(s)
```

This program creates two threads: a server and a client. These two threads communicate between themselves by a specific port in the localhost address.

The client contacts the server, and the server responds by sending the client XML documents specified as parameters.

For testing purposes, create an XML document called content2.xml. This document is a copy of the one in Example 3-3 on page 56. In this second document, change all *zOSXML* references to *zOS2* strings, so you can distinguish between the documents. Do not refer to an external DTD document because the client will not be able to access it.

Once you have prepared your two documents, run the following command:

```
java socket.KeepSocketOpen ../xml/contents.xml ../xml/content2.xml
```

The response should be like that shown in Figure 3-4.

```
DAVID:/u/david/samples/java/dom $ java socket.KeepSocketOpen
../xml/contents.xml
 ../xml/content2.xml
Server: Created.
Client: Created.
Server: Running.
Client: Running.
Server: Client connected.
Server: Opening file "../xml/contents.xml"
Server: Wrote 2211 byte(s) total.
Server: Opening file "../xml/content2.xml"
Server: Wrote 2073 byte(s) total.
Server: Exiting.
Client: Read 2211 byte(s) total.
Client: Read 2211 byte(s) total.
Client: 154 ms (26 elems, 21 attrs, 0 spaces, 386 chars)
Client: Read 2073 byte(s) total.
Client: Read 2073 byte(s) total.
Client: 67 ms (26 elems, 21 attrs, 0 spaces, 386 chars)
Client: Exiting.
```

Figure 3-4 socket.KeepSocketOpen output

In the main() section of the source file you will see the following declaration:

```
// constants
final int port = 6789;
           .......................................
if (argv.length == 0) {
    System.out.println("usage: java socket.KeepSocketOpen file(s
    System.exit(1);
}
```

```
   // create server and client
   Server server = new Server(port, argv);
   Client client = new Client("localhost", port);
```

Here, we declare *6789* as the port to use, and create an instance for Server and another for Client. The server wraps each document in a `socket.io.WrappedOutputStream`. This prevents problems with the length of the message, and the client reads from a `socket.io.WrappedInputStream`. The server has a loop for the number of bytes resulting from the files you give it as parameters, and the client parses them as they come to it.

3.2 C/C++ samples

In this section we introduce the setup of C/C++ samples, and we describe how to run a representative sample using XSLT processing.

The samples that come with the toolkit are the following:

SAXCount	Counts the elements, attributes, spaces, and characters in an XML file
SAX2Count	Same as SAXCount, except uses SAX 2.0
SAXPrint	Parses an XML file and prints it out
SAX2Print	Same as SAXPrint, except uses SAX 2.0
DOMCount	Counts the elements, attributes, spaces, and characters in an XML file
IDOMCount	Counts the elements, attributes, spaces and characters in an XML file (uses experimental IDOM API)
DOMPrint	Parses an XML file and prints it out
IDOMPrint	Parses an XML file and prints it out (uses experimental IDOM API)
MemParse	Parses XML in a memory buffer, outputting the number of elements and attributes.
Redirect	Redirects the input stream for external entities
PParse	Demonstrates progressive parsing
StdInParse	Demonstrates streaming XML data from standard input
EnumVal	Shows how to enumerate the markup declarations in a DTD validator
SEnumVal	Shows how to enumerate the markup declarations in a Schema validator

CreateDOMDocument Creates a DOM tree in memory from scratch

3.2.1 Setting up the samples

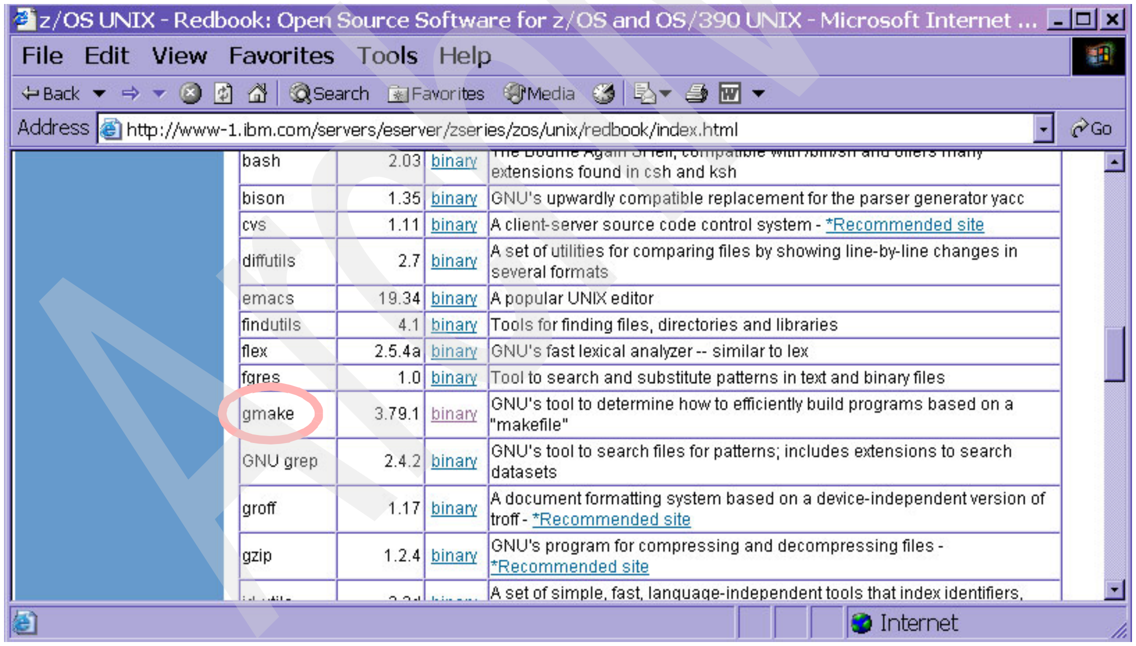
Setting up the C++ XML samples for z/OS or OS/390 UNIX System Services

To run Toolkit C/C++ parser samples on UNIX System Services for z/OS or OS/390, you need to do following:

- Ensure that the SCEERUN Language Environment run-time library can be found in the correct searching sequence z/OS or OS/390 follows to find modules. In your system this library is called *<hlq>.SCEERUN* (check the first hlq in your installation); add it to the *LNKLSTxx* member in use.
- You need a GNU gmake utility to build parser samples. From a browser, go to following URL:

  `http://www.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html`

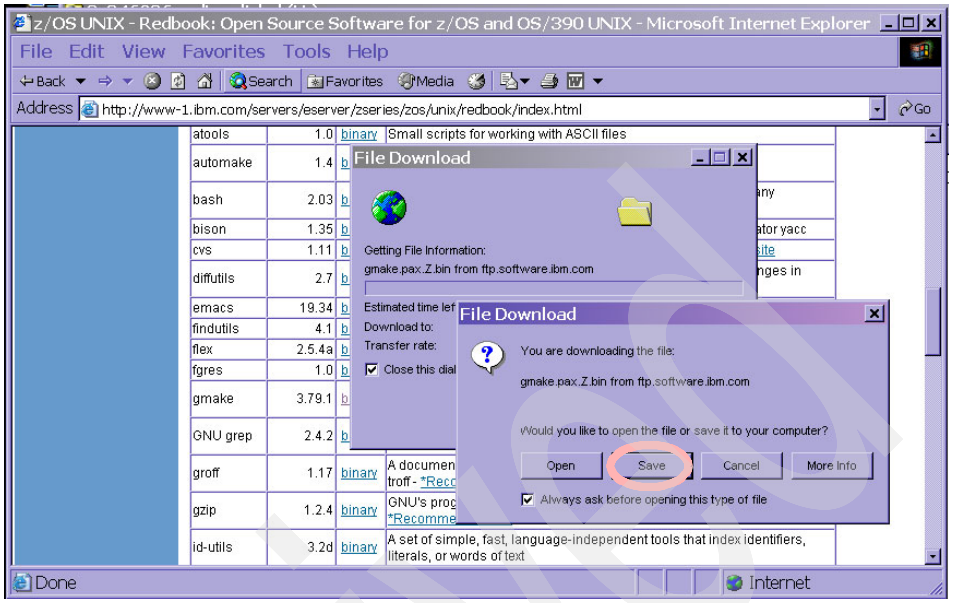
  Find the **gmake** link and click it. See Figure 3-5.


z/OS UNIX - Redbook: Open Source Software for z/OS and OS/390 UNIX - Microsoft Internet ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html Go
Internet

bash	2.03	binary	The Bourne Again Shell, compatible with /bin/sh and offers many extensions found in csh and ksh
bison	1.35	binary	GNU's upwardly compatible replacement for the parser generator yacc
cvs	1.11	binary	A client-server source code control system - *Recommended site
diffutils	2.7	binary	A set of utilities for comparing files by showing line-by-line changes in several formats
emacs	19.34	binary	A popular UNIX editor
findutils	4.1	binary	Tools for finding files, directories and libraries
flex	2.5.4a	binary	GNU's fast lexical analyzer -- similar to lex
fgres	1.0	binary	Tool to search and substitute patterns in text and binary files
gmake	3.79.1	binary	GNU's tool to determine how to efficiently build programs based on a "makefile"
GNU grep	2.4.2	binary	GNU's tool to search files for patterns; includes extensions to search datasets
groff	1.17	binary	A document formatting system based on a device-independent version of troff - *Recommended site
gzip	1.2.4	binary	GNU's program for compressing and decompressing files - *Recommended site
idutils	3.2.1	binary	A set of simple, fast, language-independent tools that index identifiers,

Figure 3-5 Obtain gmake tool

Click the **binary** link. This link will open a message window in which you have to select "Save this file to disk."


z/OS UNIX - Redbook: Open Source Software for z/OS and OS/390 UNIX - Microsoft Internet Explorer

Address http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html

atools	1.0	binary	Small scripts for working with ASCII files
automake	1.4	b	
bash	2.03	b	ny
bison	1.35	b	ator yacc
cvs	1.11	b	site
diffutils	2.7	b	nges in
emacs	19.34	b	
findutils	4.1	b	
flex	2.5.4a	b	
fgres	1.0	b	
gmake	3.79.1	b	
GNU grep	2.4.2	b	
groff	1.17	binary	A documen troff - *Recc
gzip	1.2.4	binary	GNU's prog *Recomme
id-utils	3.2d	binary	A set of simple, fast, language-independent tools that index identifiers, literals, or words of text

File Download

Getting File Information:
gmake.pax.Z.bin from ftp.software.ibm.com

Estimated time lef
Download to:
Transfer rate:
Close this dial

File Download

You are downloading the file:

gmake.pax.Z.bin from ftp.software.ibm.com

Would you like to open the file or save it to your computer?

Open | Save | Cancel | More Info

Always ask before opening this type of file

Done | Internet

Figure 3-6 Download gmake tool

Select a directory on your PC into which the compressed file will be downloaded. For example, you could use `C:\temp`. The name of the file is of type `gmake-bin.pax.Z`.

Once you have downloaded the file to your PC, open a z/OS or OS/390 UNIX System Services shell session. Then, create a directory called */usr/local*, or another name of your choice.

```
mkdir /usr/local
```

This directory should be on a different HFS, so it will not be replaced on subsequent maintenance or service fixes. You can create this new HFS with the ISHELL:

a. From TSO enter command **`TSO ISHELL`** to get into ISHELL application.

b. From ISHELL, go to pop-up menu bar and select **`File System - New`**. Enter name of your HFS dataset, primary and secondary space assignment, and if it is necessary, specify storage class, management class, and data class (check the SMS requirements for your installation).

```
File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
|___________________________________________________________________________|
|                           Make a File System                           |
|                                                                        |
| File system name  . . . 'OMVS.SC58.GMAKE'___________________________  |
| Primary cylinders . . . 4_______                                       |
| Secondary cylinders . . 1_______                                       |
| Storage class . . . . . . ________                                       |
| Management class  . . . . ________                                       |
| Data class  . . . . . . . ________                                       |
| Volume  . . . . . . . .  _______                                       |
| Unit  . . . . . . . . .  _______                                       |
|                                                                        |
|___________________________________________________________________________|
```

Figure 3-7 Create a file system

The mount point of this new HFS should be */usr/local* (or the directory you created). You can mount your new HFS from ISHELL. Select **File System** -> **Mount**.

```
Mount a File System

Mount point:
                                                         More:     +
   /usr/local
   ________________________________________________________________

File system name . . 'OMVS.SC58.GMAKE'___________________________
File system type . . HFS_____  New owner  . . . . . ________
Owning system  . . . ________  Character Set ID . . _____

Select additional mount options:
_  Read-only file system          _  Do not automove file system
_  Ignore SETUID and SETGID       _  Automove unmount file system
_  Bypass security                _  Text conversion enabled

Mount parameter:
____________________________________________________________________
```

Figure 3-8 Mount a file system

Remember to update your *BPXPRMxx* UNIX parameter member in SYS1.PARMLIB. For example, add:

```
MOUNT FILESYSTEM('OMVS.&SYSNAME..GMAKE')
             MOUNTPOINT('/usr/local')
```

```
                TYPE(HFS)  MODE(RDWR)
```

Now you have a new directory */usr/local* from which it hangs an independent HFS (OMVS.&SYSNAME..GMAKE).

c. Next, we are going to do a binary file transfer from C:\temp\gmake.bin.pax.Z file on your workstation to the new /usr/local directory on z/OS. From a Windows command prompt, enter the following commands:

```
ftp> cd /usr/local
ftp> lcd C:\temp
ftp> bin
ftp> put gmake.bin.pax.Z
```

And, from the z/OS shell prompt, enter the following command:

```
pax -rzf gmake.bin.pax.Z
```

After executing this command, you have following subdirectories in directory /usr/local:

```
./bin
./info
./share
./man
```

- From the shell prompt, enter the commands:

```
export PATH=$PATH:/usr/local/bin
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-4_0/samples
unset _CXX_CXXSUFFIX
export CXX=c++
export CXXFLAGS=-2
```

- Then, enter the commands to configure the samples environment:

```
cd $XERCESCROOT/samples
configure
```

Wait until it finishes. Check output messages for errors. From this point, you have created Makefile files for C/C++ parser samples.

- Once you have finished the configuration, you have to enter the following commands. Be careful with permission bits; if any file is not accessible, compilation is going to fail:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_CCMODE=1
export _CXX_CVERSION="0x220a0000"
```

(You have to enter this command only if you are using a C/C++ compiler higher than V2R10.)

```
gmake
```

- Wait until all compilations have finished. It is possible to have four warning messages, and we had a problem with permission bits and owner on directories and files. Check all these points before you look for other reasons. To run parser samples, you need some more adjustments:

  ```
  export LIBPATH=$XERCESCROOT/lib:$LIBPATH
  export ICU_DATA=$XERCESCROOT/lib
  ```

  You could add these lines to your .profile file. These lines are needed to bind with libxerces-c1_6_0.dll, libxerces-c1_6_0.x, libicuuc.20.2.dll, libicudt20e.dll, and libicudt20e_390.dll.

- You are now ready to run the parser samples. The SAXCount program is equivalent to the program with the same name we saw in the Java section. To run it, enter the following:

  ```
  cd $XERCESCROOT/bin
  SAXCount $XERCESCROOT/samples/data/personal.xml
  ```

  personal.xml is an XML document we used for testing. The response you receive should be similar to Figure 3-9.

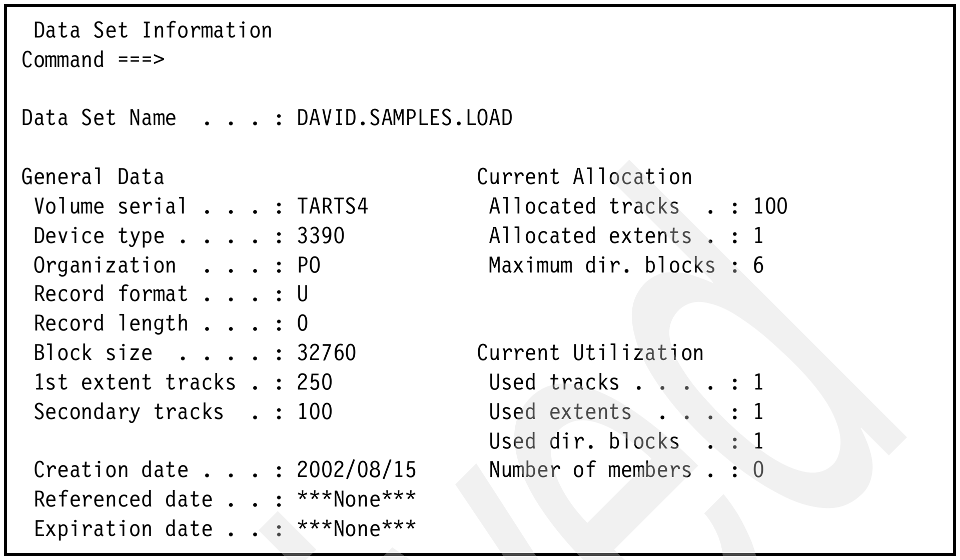
```
DAVID:/usr/lpp/ixm/IBM/xml4c-4_0/bin $ SAXCount
$XERCESCROOT/samples/data/person
al.xml
/usr/lpp/ixm/IBM/xml4c-4_0/samples/data/personal.xml: 55 ms (37 elems, 12
attrs, 134 spaces, 134 chars)
```

Figure 3-9 SAXCount output

Setting up C/C++ XML samples for z/OS or OS/390

In addition to the setup described in the previous section, perform the following steps to be able to run z/OS or OS/390 samples:

1. Allocate a PDS dataset to contain all necessary modules. The dataset we allocated had the attributes identified in Figure 3-10.



```
 Data Set Information
Command ===>

Data Set Name  . . . : DAVID.SAMPLES.LOAD

General Data                          Current Allocation
 Volume serial . . . : TARTS4          Allocated tracks  . : 100
 Device type . . . . : 3390            Allocated extents . : 1
 Organization  . . . : PO              Maximum dir. blocks : 6
 Record format . . . : U
 Record length . . . : 0
 Block size  . . . . : 32760          Current Utilization
 1st extent tracks . : 250             Used tracks . . . . : 1
 Secondary tracks  . : 100             Used extents  . . . : 1
                                       Used dir. blocks  . : 1
 Creation date . . . : 2002/08/15      Number of members . : 0
 Referenced date . . : ***None***
 Expiration date . . : ***None***
```

Figure 3-10 Partitioned dataset attributes

2. Open a shell session (TSO OMVS) and enter the following commands:

   ```
   export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-4_0
   ```

   Remember that you can add this line to your .profile, so you do not need to enter this command again.

   Adjust the high level qualifiers, and file names, depending on the options selected in your installation of the Toolkit.

   ```
   export LOADMOD=DAVID.SAMPLES.LOAD
   export LOADEXP=IXM.SIXMEXP
   export OS390BATCH=1
   unset _CXX_CXXSUFFIX
   export CXX=c++
   export CXXFLAGS=-2
   cd $XERCESCROOT/samples
   configure
   ```

   These are the Makefiles needed to compile the samples. Wait until the process ends, and check for error messages. If everything is correct, you will see the messages shown in Example 3-5.

Example 3-5 Makefile messages

```
    creating SAXCount/Makefile
    creating SAX2Count/Makefile
    creating DOMCount/Makefile
    creating IDOMCount/Makefile
```

```
    creating SAXPrint/Makefile
    creating SAX2Print/Makefile
    creating DOMPrint/Makefile
    creating IDOMPrint/Makefile
    creating MemParse/Makefile
    creating Redirect/Makefile
    creating PParse/Makefile
    creating StdInParse/Makefile
    creating EnumVal/Makefile
    creating SEnumVal/Makefile
    creating CreateDOMDocument/Makefile
```

3. Create sample modules using the gmake utility:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_XSUFFIX_HOST=SIXMEXP
export _CXX_CCMODE=1
export _CXX_CVERSION="0x220a0000"
```

   (You have to enter this command only if you are using a C/C++ compiler higher than V2R10.)

```
gmake
```

   You can expect a return code 4.

4. To run parser samples in a z/OS or OS/390 environment, you can put the library *SIXMMOD1* with the modules you generated in the installation in LNKLST, or you can put it in STEPLIB in the JCL. We used the following JCL:

```
//TOOLKIT1 JOB (POK,999),DAVID,MSGLEVEL=(1,1),MSGCLASS=X,
//  CLASS=A,NOTIFY=&SYSUID
//*
//*********************************************************************
//RUNIVP EXEC PGM=SAXCOUNT
//STEPLIB DD DSN=FRANCK.SIXMMOD1,DISP=SHR
//        DD DSN=DAVID.SAMPLES.LOAD,DISP=SHR
```

   The output from execution is shown in Figure 3-11.

```
IEF376I  JOB/TOOLKIT1/STOP  2002227.1546 CPU 0MIN 00.03SEC SRB 0MIN 00.02S
Usage:
    SAXCount Ýoptions¨ <XML file | List file>
This program invokes the SAX Parser, and then prints the
number of elements, attributes, spaces and characters found
in each XML file, using SAX API.
Options:
  -l Indicate the input file is a List File that has a list of xml fi
           Default to off (Input file is an XML file).
  -v=xxx    Validation scheme Ýalways | never | auto*¨.
  -n        Enable namespace processing. Defaults to off.
  -s        Enable schema processing. Defaults to off.
  -f        Enable full schema constraint checking. Defaults to off.
  -? Show this help.
  * = Default if not provided explicitly.
```

Figure 3-11 SAXCOUNT output

In the previous example, we could pass the document to parse as a parameter. The JCL would be as follows:

```
//TOOLKIT1 JOB (POK,999),DAVID,MSGLEVEL=(1,1),MSGCLASS=X,
//  CLASS=A,NOTIFY=&SYSUID
//*
//*********************************************************************
//RUNIVP EXEC PGM=SAXCOUNT,
//     PARM='//usr/lpp/ixm/IBM/xml4j-4_0/data/personal.xml'
//STEPLIB DD DSN=FRANCK.SIXMMOD1,DISP=SHR
//        DD DSN=DAVID.SAMPLES.LOAD,DISP=SHR
```

Note that in the path of the document we passed in the parameter line, we included an extra slash at the beginning; if you write an absolute path, the beginning of the line is "//". On the other hand, if you want to activate POSIX processing, you can write a parameter as follows:

`PARM='POSIX(ON)//usr/lpp/ixm/IBM/xml4j-4_0/data/personal.xml'`

You can check the result of the previous JCL execution in the job output:

```
/usr/lpp/ixm/IBM/xml4j-4_0/data/personal.xml: 59 ms (37 elems, 18 attrs, 140 
spaces, 128 chars)
```

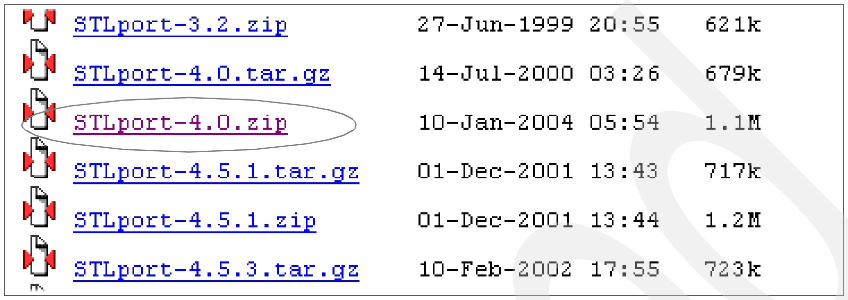
Setting up the C/C++ XSLT samples for z/OS or OS/390 UNIX System Services

- To use XSLT processor in C/C++ you need STLport-4.0 (Standard Template Library). To install this product, perform the following steps:

a. Go to the Web browser, and open URL:

`http://www.stlport.com/archive/`

b. Search for a link to version 4.0 of the product and click **STLport-4.0.tar.gz,** (see Figure 3-12).



```
STLport-3.2.zip          27-Jun-1999 20:55   621k
STLport-4.0.tar.gz       14-Jul-2000 03:26   679k
STLport-4.0.zip          10-Jan-2004 05:54   1.1M
STLport-4.5.1.tar.gz     01-Dec-2001 13:43   717k
STLport-4.5.1.zip        01-Dec-2001 13:44   1.2M
STLport-4.5.3.tar.gz     10-Feb-2002 17:55   723k
```

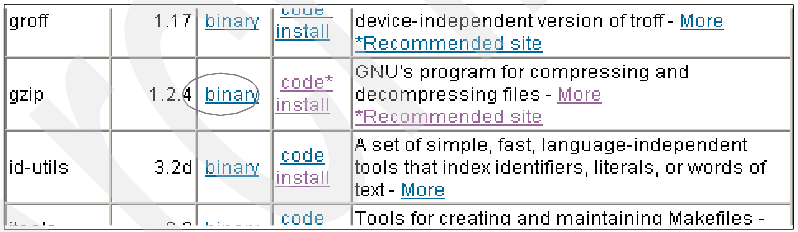
Figure 3-12 Locate STLport 4.0

c. Select "Save this file to disk" and select a folder in which to store the downloaded file. For example, select `C:\temp`.

▸ You also need another tool, GNU zip, to decompress files in format gz. To obtain this tool, point your browser to:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html`

Locate the `gzip` link and select the `binary` option.



groff	1.17	binary	code install	device-independent version of troff - More *Recommended site
gzip	1.2.4	binary	code* install	GNU's program for compressing and decompressing files - More *Recommended site
id-utils	3.2d	binary	code install	A set of simple, fast, language-independent tools that index identifiers, literals, or words of text - More
			code	Tools for creating and maintaining Makefiles -

Figure 3-13 gzip download

When prompted to save the file to disk, click **OK**. Select the folder in which to save the file. In our case we selected the same folder: C:\temp. Now, in this directory you also have a file called gzip.bin.pax.Z.

▸ FTP to the z/OS host the files you just downloaded to your PC.

```
C:\ftp 9.12.2.22
Connected to 9.12.2.22.
220-FTPD1 IBM FTP CS V1R2 at wtsc58oe, 21:43:26 on 2002-08-15.
220 Connection will close if idle for more than 30 minutes.
User (9.12.2.22:(none)): david
331 Send password please.
Password:
230 DAVID is logged on. Working directory is "/u/david".
ftp> cd /usr/local
250 HFS directory /usr/local is the current working directory
ftp> lcd C:\temp
Local directory now C:\temp
ftp> bin
200 Representation type is Image
ftp> put STLport-4.0.tar.gz
125 Storing dataset /usr/local/STLport-4.0.tar.gz
250 Transfer completed successfully.
ftp: 695790 bytes sent in 0.62Seconds 1120.43Kbytes/sec.
ftp> put gzip.bin.pax.Z
125 Storing dataset /usr/local/gzip.bin.pax
250 Transfer completed successfully.
ftp: 129024 bytes sent in 0.13Seconds 992.49Kbytes/sec.
```

Figure 3-14 ftp upload gzip to z/OS

- Open a shell session (TSO OMVS). Go to `/usr/local` and enter the following commands:

  ```
  cd /usr/local
  pax -rf gzip.bin.pax.Z
  gzip -d STLport-4.0.tar.gz
  tar -xvf STLport-4.0.tar
  ```

  Wait until decompression finishes. If you have problems, check your write permission bits.

- At this point, you have a subdirectory called STLport-4.0, but it is in ASCII, so your compilations are going to fail. To solve this problem, issue the commands:

  ```
  pax -o from=ISO8859-1,to=IBM-1047 -wf compout STLport-4.0
  rm -Rf STLport-4.0
  pax -rf compout
  ```

  These steps re-compress your ASCII directory into a new file, use **`pax`** to convert from ASCII to EBCDIC, and then decompress the file using **`pax`**.

- Once you have the STL libraries installed, enter the commands:

  ```
  export XALANCROOT=/usr/lpp/ixm/IBM/LotusXSL-C_1_3
  ```

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-4_0
export STLPORTROOT=/usr/local/STLport-4.0
export XALANCOUT=/u/david/samples/C/XSLT
```

Directory /u/david/samples/C/XSLT is a new directory created for the output of the compilations.

```
export _CXX_CXXSUFFIX=cpp
export _CXX_CCMODE=1
export _CXX_CVERSION="0x220a0000"
```

(You have to enter this command only if you are using C/C++ compiler higher than V2R10)

```
cd $XALANCROOT/samples
/usr/local/bin/gmake
```

Remember: this is the directory where we installed the `gmake` utility.

These compilations may end with a return code 4, giving you some warning messages. In the lab we obtained messages such as the following:

```
"/usr/local/STLport-4.0/stlport/stl/_construct.h", line 53.74: CBC1252(W)
The destructor for "NodeSorter::VectorEntry" does not exist. The call is
ignored.
```

Another problem we encountered was running out of virtual storage, which happened while we were running the TraceListen sample. We solved this problem by increasing the region size for the TSO userid.

Setting-up the C/C++ XSLT samples for z/OS or OS/390

Once you have completed the setup in the UNIX environment, you can generate the modules in a PDS to invoke them in the z/OS environment.

1. First of all, check that in the Toolkit installation process you have installed library <hlq>.SIXMEXP with members:

   – `IXMLC13X`
   – `IXM4C40X`
   – `IXM20UCX`
   – `IXM2018X`

   You are going to use the same dataset, DAVID.SAMPLES.LOAD, already used in the previous section.

2. Enter the following commands:

```
export XALANCROOT=/usr/lpp/ixm/IBM/LotusXSL-C_1_3
export LOADMOD=DAVID.SAMPLES.LOAD
export LOADEXP=<hlq>.SIXMEXP
export OS390BATCH=1
unset _CXX_CXXSUFFIX
export CXX=c++
export CXXFLAGS=-2
```

(or -g if building a debug version)

```
export _CXX_CXXSUFFIX=cpp
export _CXX_XSUFFIX_HOST=SIXMEXP
export _CXX_CCMODE=1
export _CXX_CVERSION="0x220a0000"
gmake
```

3. The built samples are now in the DAVID.SAMPLES.LOAD dataset.

3.2.2 Running the C++ XSLT sample on z/OS or OS/390 UNIX System Services

We have chosen the SimpleTransform sample to test the XSLT processor. This sample uses a foo.xsl stylesheet to transform a foo.xml document, and the output is stored on foo.out. For this exercise, we made two input documents. You could use sample documents stored in $XALANCROOT/samples/SimpleTransform, but we preferred to create new documents for our test.

First we made the XML document, calling it *foo.xml*. This document contains a list of products with their prices, and is shown in Example 3-6.

Example 3-6 foo.xml

```
<?xml version="1.0" encoding="ibm-1047"?>
<Pricelist>
<Entry>
<ProductID>BLU-051</ProductID>
<UPC>780811-100051</UPC>
<Description>Blue Potatoes-18-2 lb Vexar-US1-C</Description>
<Category>Blue Potatoes</Category>
<Price>63.03</Price>
</Entry>
<Entry>
<ProductID>BLU-048</ProductID>
<UPC>780811-100048</UPC>
<Description>Blue Potatoes-25 lb box-US1-A</Description>
<Category>Blue Potatoes</Category>
<Price>8.85</Price>
</Entry>
<Entry>
<ProductID>BLU-049</ProductID>
<UPC>780811-100049</UPC>
<Description>Blue Potatoes-25 lb box-US1-B</Description>
<Category>Blue Potatoes</Category>
<Price>7.89</Price>
</Entry>
```

```
</Pricelist>
```

We stored this file in the same directory where the SimpleTransform executable is located. We made an XSL document in the same directory and called it foo.xsl. The purpose of this stylesheet is to transform the input XML document, building a table in which each row corresponds to a product. The XSL document is shown in Example 3-7.

Example 3-7 foo.xsl

```
<?xml version="1.0" encoding="ibm-1047"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<HEAD>
<Title>Price List Viewer</Title>
</HEAD>
<BODY>
<CENTER>
<TABLE CELLSPACING="2" CELLPADDING="1" BORDER="0">
<col width="25" />
<col width="40" />
<col width="200" />
<col width="100" />
<col width="60" />
<TR>
<TD CLASS="Header" ALIGN="LEFT" VALIGN="BOTTOM">Product ID</TD>
<TD CLASS="Header" ALIGN="LEFT" VALIGN="BOTTOM">UPC</TD>
<TD CLASS="Header" ALIGN="LEFT" VALIGN="BOTTOM">Description</TD>
<TD CLASS="Header" ALIGN="LEFT" VALIGN="BOTTOM">Category</TD>
<TD CLASS="Header" ALIGN="RIGHT" VALIGN="BOTTOM">Price</TD>
</TR>
<xsl:for-each select="Pricelist/Entry">
<TR>
<TD CLASS="Row" ALIGN="LEFT" VALIGN="TOP">
<xsl:value-of select="ProductID" />
</TD>
<TD CLASS="Row" ALIGN="LEFT" VALIGN="TOP">
<xsl:value-of select="UPC" />
</TD>
<TD CLASS="Row" ALIGN="LEFT" VALIGN="TOP">
<TD CLASS="Row" ALIGN="LEFT" VALIGN="TOP">
<xsl:value-of select="Description" />
</TD>
<TD CLASS="Row" ALIGN="LEFT" VALIGN="TOP">
<xsl:value-of select="Category" />
</TD>
<TD CLASS="Row" ALIGN="RIGHT" VALIGN="TOP">
<xsl:value-of select="Price" />
</TD>
```

```
</TR>
</xsl:for-each>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

The result should be a foo.out document in which each product entry is a different row in an HTML table.

Once we have created the XML and XSL documents in the same directory with the SimpleTransform executable, we need to make some final adjustments. Enter the following commands:

```
export LIBPATH=$XALANCROOT/lib:$XERCESCROOT/lib:$LIBPATH
export ICU_DATA=$XERCESCROOT/lib
export PATH=$XALANCOUT/bin:$PATH
```

Now, it is time to execute the program. Enter:

```
SimpleTransform
```

At the end of the program, you have a new file: foo.out. You are not able to look at it (if you try OBROWSE foo.out you will see weird characters), because it is coded in UTF-8, but you can download it to your PC with FTP and look at it with Notepad. You can also use a tool such as viascii, which lets you to see ASCII files from your vi editor. If you want this tool, you can download it from:

http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty2.html

Another easy way to check that the resulting file is correct is to use the ICONV utility. For example, for the previous foo.out coded in UTF-8, we can enter the following command:

```
iconv -f utf-8 -t ibm-1047 foo.out > foo2.out
```

Now we can browse foo2.out, because it is in EBCDIC (IBM-1047 codepage).

We are going to do a second test. This time we are testing testXSLT from the command prompt. Enter the following commands:

```
testXSLT -IN $XALANCROOT/samples/SimpleTransform/foo.xml
         -XSL $XALANCROOT/samples/SimpleTransform/foo.xsl
         -OUT foo.out
```

Again you can use ICONV to convert foo.out from UTF-8 to EBCDIC IBM-1047. You can also update the source files from the programs to generate the output in the codepage you want.

3.2.3 Running the C++ XSLT sample on z/OS or OS/390

To test the sample, you only have to ensure that you have access to SIXMMOD1. For instance, if you want to execute the TraceListen sample you only have to build the JCL shown in Example 3-8.

Example 3-8 TraceListen JCL

```
//TOOLKIT2 JOB (POK,999),DAVID,MSGLEVEL=(1,1),MSGCLASS=X,
//  CLASS=A,NOTIFY=&SYSUID
//*
//*******************************************************************
//RUNIVP EXEC PGM=TRACELSN,
//     PARM='/-tt'
//STEPLIB DD DSN=<hlq>.SIXMMOD1,DISP=SHR
//        DD DSN=DAVID.SAMPLES.LOAD,DISP=SHR
```

The samples you can use are the same ones we discussed in the UNIX System Services section.

4

Services development environment

In this chapter we introduce some critical components of an application development environment that are required to modernize your legacy applications.

Today’s Web application development teams include business analysts, managers, host programmers, application programmers, Web page designers, graphics designers, Java programmers, and component developers. A different person might fill each role, or any one person might be required to play multiple roles. Planning a Web application has become complex because of the varied skills and numerous roles required. For example, on the EIS tier, you may have COBOL or PL/1 programmers with experience building IBM CICS or IMS transactions or other applications and databases associated with current business logic. Their approach to development is probably based on models of structural programming, and most likely does not separate the user interface from the business logic.

The processes for building such systems are specific to the host environment. Your development team needs professionals with experience in your existing business applications—perhaps host programmers—working on aspects of the middle tier. On the middle tier, you can have Java programmers building servlets, classes, or JSP code. Their development and architectural model is likely to be more object-oriented. It is crucial to get the business knowledge that is embedded in your existing applications and leverage it as you develop in the middle tier.

Graphic designers and HTML programmers develop for presentation on client systems (graphics, JSP pages, HTML). Today this means browsers, but other client platforms, such as hand-held devices and data-enabled phones, are becoming popular. Team members with these skills tend to have backgrounds in building user interfaces. Of course, managers of teams developing Web applications might come from any of these programming disciplines, or perhaps a technical business role, or some other technical lead position. Teams need development tooling that not only allows diverse roles to interact, but also integrates the members as a team.

Before we go any further, we need to clarify a few frequently used terms. For the purpose of this discussion, the terms *Services* and *Components* will be used in an interchangeable fashion. There is a slight semantic difference between these two terms. Services are run-time objects and Components are design-time objects. *Enterprise assets* or just *assets* in the context of this report refer to the application software systems of an enterprise (also sometimes referred to as *legacy systems*). *Harvesting* and *Re-engineering* are also used interchangeably. The terms harvesting, enterprise asset, and the basic thought processes of modernizing legacy systems are presented in the seminal paper *Enterprise Solution Structure* by E.C. Plachy and P.A. Hausler, published in 1999 in IBM System Journal, volume 38, number 1.

4.1 Elements of e-business development tools

For enterprises to fully engage and implement Web applications, development tools are needed to directly leverage existing assets, help solve the middle-tier skill and complexity problems, and support proven development practices. The need is for an integrated development environment in which critical aspects of Web application development and the associated modernization of existing applications can be addressed. Such tools would also include the ability to simplify the combination of the complex technologies involved—within and between the various tiers—in Web application development.

Development tools for the enterprise need to take the best capabilities of the point tools and combine them into a single coherent environment where team members with various technology backgrounds and experience can bring their particular expertise to bear. These tools also need to support the enterprise's requirement to create reusable components that can be leveraged throughout the e-business environment.

With the availability of XML, Java, and infrastructures like WebSphere on z/OS and OS/390, developers need a new generation tool set to build robust services-oriented architecture (SOA) based, mission-critical business applications. The convergence on the same platform of new technologies like XML and Java on one hand, and COBOL, CICS, and IMS on the other hand, is opening up a fascinating and exciting application solution model. The new model not only holds promise of lower cost of ownership and development, which is very good news for organizations, it also raises the possibility for many thousands of legacy application developers to blur the line that divided them from Web technology-based solution developers.

The toolsets that can make this happen are:

- WebSphere Studio Enterprise Developer (WSED)
- WebSphere Studio Asset Analyzer (WSAA)
- XML Repository

Figure 4-1 presents an overview of how these three toolsets are related.

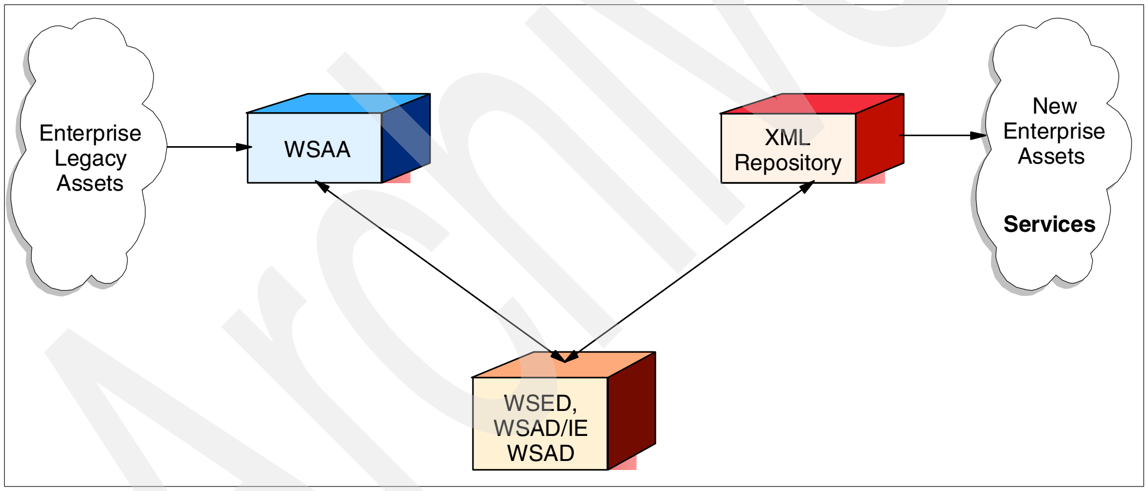


Figure 4-1 e-business development tools

While individual organizations and developer communities will have their preferred set of tools for specific tasks, we believe these three classes of toolsets provide a great way to kick-start your “Legacy Modernization” effort and help you achieve your e-business goal in the fastest possible time. This toolset environment can be supplemented by additional tools (such as XML editors, debuggers, and so forth), based on per individual preferences.

4.2 WebSphere Studio Enterprise Developer

WebSphere Studio Enterprise Developer offers tools for building and managing complex, component-based, N-tier Web applications to development teams with heterogeneous skill sets. It also leverages component architecture models and development tooling that can enable the successful creation, deployment, and maintenance of enterprise Web applications. With WebSphere Studio Enterprise Developer, an enterprise can create Web applications by integrating diverse employee skill sets and extending existing systems.

This approach allows enterprises to use proven run-time environments while helping to reduce deployment risks. WebSphere Studio Enterprise Developer also supports accepted practices and emerging Web application technologies to help ensure that development teams build robust, component-based applications. In particular, you can use a Model-View-Controller (MVC) architectural model and an open-source Struts (MVC2) implementation design.

WebSphere Studio Enterprise Developer also helps extend emerging Web application component-oriented technologies and projects, such as XML, Struts, Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP).

It is not our intent here to describe all the features of WebSphere Studio Enterprise Developer; other redbooks deal with the topic in great detail. For example, see *Legacy Modernization with WebSphere Studio Enterprise Developer,* SG24-6806. In this chapter we only highlight some of the important features that are critical to build an N-tier-based application on z/OS or OS/390.

4.2.1 Designing a Web application

The focus on modernizing enterprise applications and creating e-business solutions dictates specific development processes. Existing enterprise applications must act as a resource pool for the Web applications under construction. You will need new application code and business logic. You need to create reusable components so that, as your enterprise builds its Web applications, their capabilities can be leveraged as components in future Web applications.

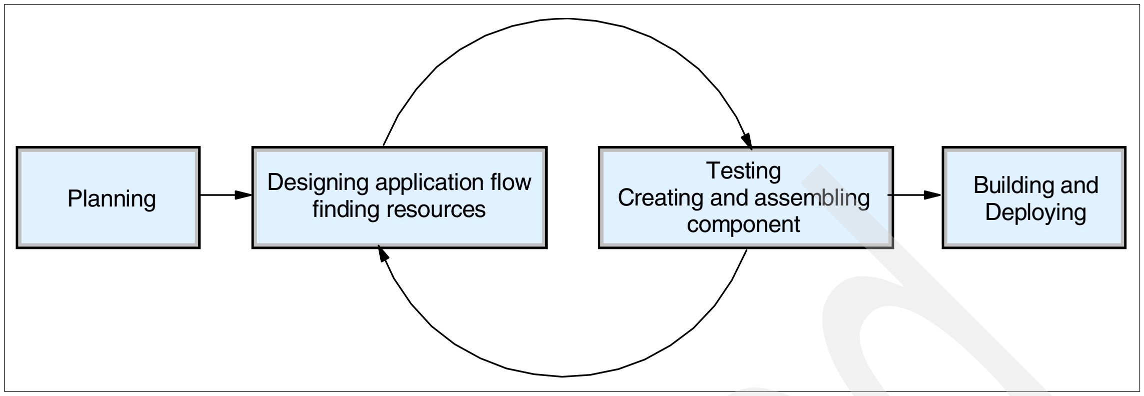

Planning
Designing application flow finding resources
Testing
Creating and assembling component
Building and Deploying

Figure 4-2 Web application development

Figure 4-2 shows a development process where existing components are identified, new components are created and application flow is defined, and components are linked together in an efficient way. Once you complete initial planning, simultaneous development begins that defines application flow, finds existing components, creates new components and engages in various levels of testing. As this development concludes, your teams can engage in build and deploy activities.

For a online demonstration of WebSphere Studio Enterprise Developer, see:

```
http://www.ibm.com/software/ad/studioedm/demo
```

In the following sections we briefly describe two important functions of WebSphere Studio Enterprise Developer from the component development point of view.

4.2.2 Creating components from existing assets

A key aspect of creating a Web application involves leveraging existing applications, and harvesting components from within the enterprise. However, developers may encounter significant difficulties when they try to create components based on traditional applications. WebSphere Studio Enterprise Developer simplifies the process by providing powerful componentization tooling that helps development teams turn existing applications into reusable components. WebSphere Studio Enterprise Developer adapter tooling provides a wizard-based user interface, which helps a developer identify important aspects of an EIS component.

The tooling automatically creates a Java 2 Connectivity (J2C) interface to the host component. This component interface is a Java class that runs on the Web server but can communicate with transactions or other capability on the host. This component interface can then be incorporated into the visual design tool, making it available as an action within the Web application. These connectors are automatically designed to be complete Web services, allowing the enterprise to directly engage Web service business paradigms.

4.2.3 Developing new components

In addition to leveraging existing capability, your development team will often need to create new components to be part of a Web application. WebSphere Studio Enterprise Developer provides powerful source editors integrated as a single development tool for building many component types required, including HTML, JSPs, EJB components, Java Beans, COBOL, PL/1, Assembler, and IBM Enterprise Generation Language (EGL).

Many combinations of these technologies can be used when implementing actions described in the visual assembly environment. This approach can allow WebSphere Studio Enterprise Developer to provide a fully integrated development environment in which the various Web application development roles can be performed effectively by a heterogeneous team.

For a comprehensive tutorial on how WebSphere Studio Enterprise Developer can be used in developing an n-tier application, refer to *Legacy Modernization with WebSphere Studio Enterprise Developer*, SG24-6806.

4.2.4 Summary of WebSphere Studio Enterprise Developer features

Struts application development tools

Struts is an open-source subproject of the Apache Jakarta project. The purpose of Struts is to encourage and support the “Model 2” model-view-controller design approach to building Web applications. In this model, applications are based on servlets and Java server pages (JSPs) that run in a servlet container such as WebSphere Application Server or Jakarta Tomcat. Struts comprises primarily a set of JSP taglibs and a relatively thin set of additional run-time support classes. For more information about Struts, see the Struts Framework Project page at:

`http://jakarta.apache.org/struts/index.html`

The Struts tools in WebSphere Studio Application Developer and WebSphere Studio Enterprise Developer integrate Struts. Struts includes the following components:

- An Action Servlet, which manages the flow of run-time events.

- A configuration file with which you configure the Action Servlet. (Using a configuration file means that changes to the flow of control do not necessarily require software changes.)
- A set of JSP taglibs to use in JSPs and to access user-specified data.
- A set of run-time classes.

The Struts tools make it easy to build and manage a Struts-based Web application. Struts tools do this by:

- Letting you set up a Struts project so that taglibs and other Struts-related resources are located properly; as a result, you can reference those resources without fail as you develop your application
- Providing wizards to create syntactically correct Action Form subclasses (form beans) and Action subclasses (actions), so you have a head start in developing the logic that is specific to your application
- Providing a specialized editor to access your Struts configuration file, which is the file that you modify to configure the Struts Action Servlet
- Providing a specialized editor to access your Struts application diagrams, which are files that help you visualize the flow structure of a Struts-based Web application
- Adding Struts support to the other editors (for example, VCT support for Struts taglibs in PageDesigner)
- Validating your Struts-based application

z/OS application development tools

z/OS application development tools provide an interactive, workstation-based environment where you can develop mainframe applications in ASM, COBOL, or PL/1. The environment gives you a seamless way to edit on the workstation and prepare output on the mainframe. Your interaction with z/OS includes these steps:

1. Create or modify the code in one of several editors (JLPEX, LPEX, or an editor that you introduce). The editor retains fixed lengths and column layouts, as appropriate.
2. Compile the code locally as a convenient way to validate the source.
3. Debug the code locally.
4. Generate and customize JCL as needed.
5. Transfer the source to the host, where z/OS tools submit the JCL or otherwise prepare the source, including pre-preparation steps for CICS and DB2 UDB.
6. Inspect the results of code preparation.

You can access z/OS datasets by way of a workstation-like directory structure; and you can process CLISTs, REXX EXECs, and USS shell scripts in the following way:

1. Edit them on the workstation.
2. Transfer them to z/OS.
3. Run them there.
4. View the output in the workstation environment.

The code you write can target CICS, IMS, or OS/390 UNIX System Services.

Enterprise Generation Language

Enterprise Generation Language (EGL) is a development technology that lets you quickly write full-function programs. The initial release of EGL lets you use a simple procedural language to create programs of the following types:

- A COBOL program that runs as a called program in CICS TS.
- A Java program that runs on z/OS UNIX System Services, Windows 2000, Windows NT, or Windows XP. You can deploy the Java program in the context of any of the J2EE containers:
  - J2EE application client
  - J2EE Web application
  - EJB container; in this case, you also generate an EJB session bean

At your request, EGL readies the generated parts for runtime. Specifically, EGL:

- Sends each generated part to the target platform
- Oversees a preparation step to compile Java programs; to translate, compile, and link CICS COBOL programs; and to bind load modules to a DB2 database
- Returns a confirmation message

EGL is built on the proven technology of VisualAge® Generator and offers several benefits:

- You can quickly implement business logic by using a procedural scripting language and a language-specific debugger.
- You can focus on the problem your code is addressing rather than on the technical complexities of CICS, MQSeries®, and SQL; for example, you can use similar I/O statements to access different types of external data stores.
- You can code in response to current platform requirements without worrying about future migration.

- You can produce multiple parts of an application system from the same source. After developing an EGL program, for example, you can generate a Java wrapper, an EJB session bean, and a back-end program. This increased efficiency comes into play, for example, when you develop software to give users access to a servlet, which in turn passes data to a generated Java wrapper, which in turn accesses either a generated program on CICS TS or an EJB Server.
- You can produce Java applications and servlets without learning object-oriented programming.
- You avoid having to configure a CICS connector when you deploy a generated program on CICS for z/OS. A generated Java wrapper reformats the data to be passed between a Web application server and a back-end program.

Java development tools

The Java development tools included with Enterprise Developer support the development of any Java application. They add a Java perspective to the workbench, as well as a number of views, editors, wizards, builders, and code merging and refactoring tools. The Java development tools offer the following capabilities:

- JDK 1.3 support
- Pluggable run-time support for JRE switching and targeting multiple run-time environments from IBM and other vendors
- Incremental compilation
- One debugger for both local and remote debugging
- Ability to run code with errors in methods
- Error reporting and correction
- Java text editor with full syntax highlighting and complete content assist
- Refactoring tools for reorganizing Java applications

Search tools

The Java development environment includes intelligent search tools for Java source files. The Java support allows you to precisely find declarations and references of Java elements (package, type, method, field). Searching is supported by an index that is kept up to date in the background as the resources corresponding to Java elements are changed.

Performance profiling

Enterprise Developer provides tools that enable you to test your application's performance early in the development cycle. This allows enough time to make architectural changes and resulting implementation changes. This reduces risk early in the cycle, and avoids 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 helps you to visualize your program execution easily, and explore different patterns within the program. The 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 show you which operations take the most time, and help you to find and solve memory leaks. You can easily identify repetitive execution behavior and eliminate redundancy, while focusing on the highlights of an execution. The profiling tools are especially designed for analyzing object-oriented programs.

Information display

Conventional performance tools, which are based on the procedural programming model, miss a lot of important information about the behavior of Java programs, which are object-oriented. The profiling tools model and present your program's execution in a way that is natural and consistent with the object-oriented model, and that retains all relevant information.

The profiling tools enable you to visualize the topology of your application, which is built from monitors, hosts, processes, and agents. You can look at different views of your application from either the monitor or the agent level. Information displayed from the monitor level is an aggregated view of your application. Output from and input to your application can be viewed in the Console view. The various other views that these tools offer help you to visualize the elements of a Java program (the objects, methods, calling sequences, object references, and threads) from many angles. Moreover, the views show you how these elements come together in your program's execution.

Pattern extraction capabilities

Pattern extraction takes a highly redundant mass of execution information, and reduces it to its fundamental form. It gives you an overall view of the execution of a program, with the choice of viewing more detail about every object or method call. Pattern extraction greatly simplifies run-time analysis. The profiling tools have powerful pattern extraction capabilities. They present recurring patterns of run-time behavior in a single, compact view.

Features to find and solve memory leaks

The profiling tools have unique features to help you find and solve memory leaks. The Object Reference view uses pattern extraction and visualization techniques to help you explore your program's data structures. It lets you quickly understand the pattern of references to and from a large numbers of objects, and helps you find objects that are holding onto references, thus preventing garbage collection.

The tools are very useful for developers because they help them to identify areas where changes would significantly improve performance. Developers can fine-tune their applications to ensure efficiency and the best possible response times for users.

Distributed process monitoring

The tools also give you the ability to concurrently monitor multiple processes that may be distributed on different machines, and you can launch remote applications.

Color-coding for classes

In the Execution Flow, Method Execution, and Method Invocation views, classes are assigned colors. The same classes are represented by the same color across these three different views. For every class, a unique color is used to draw its methods. This makes it easier to identify methods from the same class.

EJB development environment

The EJB development environment features full EJB 1.1 support, an EJB test client, a 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. The EJB development environment consists of multiple tools:

- Tools for import/export, creation and code generation, and editing, as well as support for standard deployment descriptors and extensions and bindings specific to WebSphere Application Server.
- EJB-to-RDB mapping tools that provide the model, run-time environment, and interface for editing the mapping between EJBs and relational database tables with top-down and bottom-up capability. The mappers support associations, inheritance, and converters and composers as helpers on column maps.
- A query engine that supports deployed code by generating SQL strings into persister classes.
- Tools that provide the ability to create, edit, and validate ear files.
- Editors for deployment descriptors.

J2EE perspective

All of the EJB development environment tools are accessible from the J2EE perspective. This is where your EJB projects and individual enterprise beans reside, and it is where you accomplish all of your enterprise bean development and testing activities.

Support for enterprise beans and access beans

The EJB development environment provides tools to help you create enterprise beans (either with or without inheritance), including session beans, container-managed persistence (CMP) entity beans, and bean-managed persistence (BMP) entity beans. Tools are also provided to create access beans and other EJB elements, such as associations.

Data persistence

The EJB development environment provides a mapping editor to help you map entity enterprise beans to data stores such as relational databases. There is support for top-down, bottom-up, and meet-in-the-middle development. You can also create schemas and maps from existing enterprise beans.

Deployment code

Enterprise Developer includes tools to set deployment descriptor and control descriptor properties for your enterprise beans and to generate the deployed classes that allow your beans to operate on a server. The tool that generates the deployment code is integrated with the Enterprise Developer generation options, so you can simply select individual enterprise beans as input and then select a menu item to automatically generate the deployment code. The tools support session beans, CMP entity beans, and BMP entity beans. They also allow you to create relational database tables for CMP entity beans. Once code has been generated for deployment, you can export your enterprise beans to a JAR file for installation on an EJB server, such as the WebSphere Application Server.

Verifying enterprise bean and access bean code

The EJB development environment automatically and seamlessly verifies that your enterprise bean code is consistent and that it conforms to the rules defined by the Enterprise Java Bean specification. Code verification occurs whenever an enterprise bean or its properties are changed. If any problems are detected, an error or warning icon appears beside the problematic lines of code and a message appears in the Tasks view at the bottom of the J2EE perspective

The EJB development environment also automatically verifies that access beans are constructed correctly and that they are consistent with their associated enterprise beans. Code verification occurs whenever you create or edit access beans.

Server Tools

Server Tools uses server instances and server configurations to test your projects. Server instances identify servers where you can test your projects. Server configurations contain setup information. Server Tools allows you to test your applications in different run-time environments that can be installed locally or remotely:

- The server tools feature includes a local copy of the full WebSphere Application Server run-time environment, where you can test Web projects, EJB projects, and ear projects.
- You can also test on a remote copy of the WebSphere Application Server. To do this, you must install on your remote machine:
  - WebSphere Application Server
  - IBM Agent Controller (included with Enterprise Developer as a separate install)
- Server Tools also supports the Apache Tomcat run-time environment, running locally. With Tomcat, you can only test Web projects that contain servlets and JSPs.
- A test environment called the TCP/IP Monitoring Server is also packaged with Server Tools. This is a simple server that forwards requests and responses, and monitors test activity. This run-time environment can only be run locally, and it only supports Web projects. You cannot deploy projects to the TCP/IP Monitoring Server.

Web development tools

Enterprise Developer provides the tools necessary to develop Web applications as defined in the Sun Microsystems Java Servlet specification. Web applications include static Web pages with HTML, Java server pages (JSP files), and servlets, along with all resource metadata and a deployment descriptor. The Web development environment provides you with wizards for generating Web pages driven by databases and Java beans, and tools for developing images and animated GIFs. Links are automatically updated when content changes.

This environment brings all aspects of Web application development into a common interface. Everyone on your Web site team, including content authors, graphic artists, programmers, and Web masters, can work on the same projects and access the files they need. Within the integrated Web development environment, it is easy to cooperatively create, assemble, publish, deploy, and maintain dynamic, interactive Web applications.

The Web development environment provides the following features:

- Web project creation, using the J2EE container structure
- Integrated, intuitive visual layout tools for JSP and HTML file creation and editing
- Servlet creation with a wizard
- Advanced scripting support to create client-side dynamic applications
- WebArt Designer to create graphic titles, logos, buttons, and photo frames

- Animated GIF Designer to create animation from still pictures, graphics, and animated banners
- Over 2,000 images and sounds in a built-in library
- Site style and template support
- Automatic update of links when resources are moved or renamed
- HTTP/FTP import
- FTP export (simple resource copy) to a server
- J2EE WAR/EAR deployment support
- Generation of Web applications from database queries and beans

Web services development tool

Enterprise Developer provides wizards and other tools to enable rapid development of Web services. Web services are modular, standards-based e-business applications that businesses can dynamically mix and match to perform complex transactions with minimal programming. Web services allow buyers and sellers all over the world to discover each other, connect dynamically, and execute transactions in real time with minimal human interaction.

Some examples of Web services are theatre review articles, weather reports, credit checks, stock quotations, travel advisories, or airline travel reservation processes. Each of these self-contained business services is an application that can easily integrate with other services, from the same or different companies, to create a complete business process. This inter-operability allows businesses to dynamically publish, discover, and bind a range of Web services through the Internet. The Web services development tools provided in Enterprise Developer are based on open, cross-platform standards:

- Simple Object Access Protocol (SOAP), which is a standard for reliably transporting electronic business messages from one business application to another over the Internet
- Web Services Description Language (WSDL), which describes programs accessible via the Internet (or other networks), and the message formats and protocols used to communicate with them
- Universal Description Discovery and Integration (UDDI), which enables businesses to describe themselves, publish technical specifications on how they want to conduct e-business with other companies, and search for other businesses that provide goods and services they need, all via online UDDI registries

Enterprise Developer facilitates the following processes to assist with building and deploying Web services-enabled applications:

- Discover: Browse the UDDI Business Registry to locate existing Web services for integration.
- Create or Transform: Create Web services from existing artifacts, such as beans, URLs that take and return data, DB2 XML Extender calls, DB2 stored procedures, and SQL queries.
- Build: Wrap existing artifacts as SOAP and HTTP GET/POST accessible services and describe them in WSDL. The Web services wizards assist you in generating a SOAP proxy to Web services described in WSDL and in generating bean skeletons from WSDL.
- Deploy: Deploy Web services in the WebSphere Application Server or Tomcat test environments using Server Tools.
- Test: Test Web services running locally or remotely to get instant feedback.
- Develop: Generate sample applications to assist you in creating your own Web service client application.
- Publish: Publish Web services to the UDDI Business Registry, advertising your Web services so that other businesses can access them.

Relational database environment

The Data perspective in Enterprise Developer allows you to create and manipulate the data design for your project, in terms of relational database schemas. The Data perspective lets you browse or import database schemas in the DB Explorer view, and create and work with database schemas in the Data view. You can explore, import, design, and query databases, working with either a local copy of an already deployed design, or creating an entirely new design to meet your requirements. The Data perspective provides a metadata model that is used by all other tools that need database information, such as connection information so that tools that are unaware of each other can share connections.

SQL query builder

The SQL query builder provides a visual interface for creating and executing SQL statements. You can create a simple statement or add complex expressions and grouping. When you are satisfied with your statement, you can use the SQL-to-XML wizard to generate XML and related artifacts, then use the files to implement your query in other applications, for example, a servlet or JSP.

You can create a simple query using the SQL statement wizard, or you can use the SQL query builder that supports a wider range of statements. There is also a SQL editor with highlighting and content assist that allows you to manually edit and create SQL files.

XML development environment

Enterprise Developer provides a comprehensive XML development environment that includes tools for building DTDs, XML schemas, and XML files. It also supports integration of relational data and XML.

XML editor

The XML editor is a tool for creating, viewing, and validating XML files. You can use it to create new XML files from scratch, from existing DTDs, or from existing XML schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and validate them.

DTD editor

The DTD editor is a tool for creating, viewing, and validating DTDs. Using the DTD editor, you can create and validate DTD elements, attributes, entities, and notations. You can generate XML schema files, and generate Java beans for creating XML instances of an XML schema. You can also use the DTD editor to generate a default HTML form based on the DTDs you create.

XML schema editor

The XML schema editor facilitates 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, generating DTDs and relational table definitions from XML schemas, generating Java beans for creating XML instances of an XML schema, and generating DDL from an XML schema.

XSL trace editor

The XSL trace editor allows you to apply an XSL stylesheet against an XML document to create a result document (HTML or XML). You can transform XML documents into HTML, text, or other XML document types. The editor displays the three documents (result, source XML, source XSL) and enables you to visually step through the XSL transformation script, examining the relationships between the three documents.

XML-to-XML mapping editor

The XML-to-XML mapping editor maps one or more source XML documents to a single target XML document. You can provide a source file (DTD or XML) and a target file, and define the mappings between the source and the target. Each mapping is a selection of a target field, a conversion function and source fields. Mappings can be edited, deleted, or stored for later use. After defining the mappings you can generate an XSLT script, which can then be used to combine and transform any XML documents that conform to the source DTDs.

XML and SQL query

You can use the XML and SQL query wizard to create an XML file from the results of an SQL query. You can optionally choose to create an XML schema or DTD file that describes the structure that the XML file has for use in other applications. You can also use the XML and SQL Query wizard to create a DADX file that can be used with the Web services tool. The generated DADX file will contain your SQL query.

Relational database-to-XML mapping editor

The RDB-to-XML mapping editor makes it easy to define the mapping between relational tables and a DTD file. You can map columns in one or more relational tables to elements and attributes in an XML document. You can generate a document access definition (DAD) script, used by IBM DB2 Extender, to either compose XML documents from existing DB2 data, or decompose XML documents into DB2 data. You can also create a test harness to test the generated DAD file.

4.3 Support for enterprise service development

WebSphere Studio Enterprise Developer and WebSphere Studio Application Developer provide comprehensive support for service development. In this section we present the model behind the "Services perspective" of the toolset.

WebSphere Studio Application Developer Integration Edition together with WebSphere Application Server provides you with the best solution for managing and integrating your business applications.

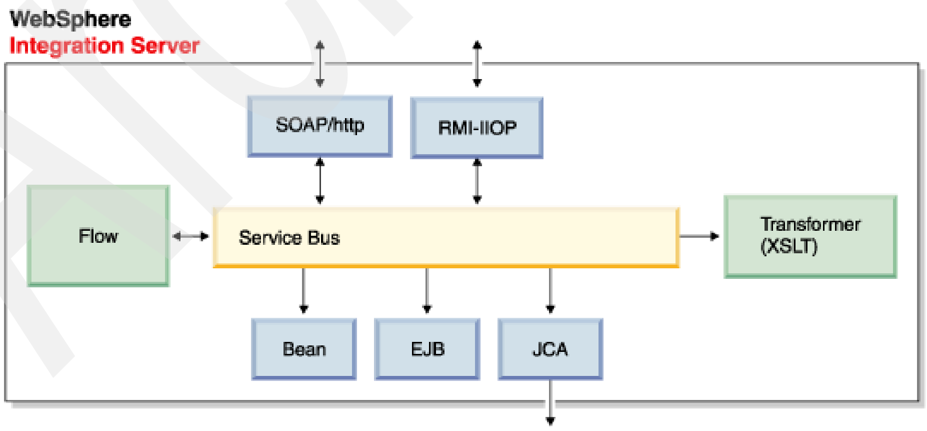


Figure 4-3 Services development

Application Developer Integration Edition includes the complete functionality of WebSphere Studio Application Developer, along with a new set of tools and wizards, collectively referred to as the Enterprise Services Toolkit. The Enterprise Services Toolkit is a fully service-oriented development environment for business and enterprise application integration. The *service bus* of the integration server (which is really a “logical” bus) acts as the point of integration for a wide variety of services. The Enterprise Services Toolkit gives you the tools and support you need to be able to consume and provide services to the integration server via the service bus.

The toolkit itself allows you to consume various service providers, such as SOAP, Java beans, Stateless Session EJBs, and J2EE Connector Architecture (JCA) services (for example, EIS services, CICS, IMS, HOD, and others). Features like flow composition can be used to compose a new service out of other services. Transformations allow you to map the data from one service to another in a flow composition.

Services deployed into the integration server can be provided as SOAP services, and via the EJB programming model. Other access options will follow in the future (such as Java Messaging Service (JMS)). The Enterprise Services Toolkit is based on open standards such as J2EE, JCA, WSDL, XSD, and XSLT. It also contains advanced technology that IBM is considering contributing to future standards.

4.3.1 Programming model

At the heart of the Enterprise Services Toolkit programming model are *Enterprise Services*, or services for short. Services are used to model different kinds of service providers in a consistent way. Figure 4-4 shows the currently supported providers. Note that in the enterprise services world a Web service is just one form of service provider.

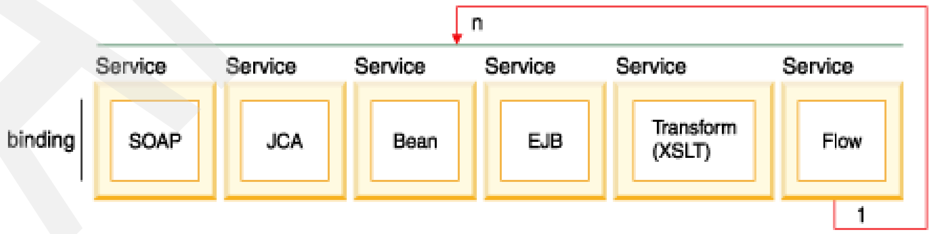


Figure 4-4 Enterprise service providers

Service

The *service* is the part of the programming model that ties everything together. The Enterprise Services Toolkit uses the Web Services Description Language (WSDL) as its model for describing any kind of service.

Note: You may think that the W in WSDL means the language is for describing Web services only, but this is not true. The inventors equipped the language with a smart extensibility mechanism that allows you to describe any kind of service, Web or otherwise.

XML schema

Service messages are described by the XML Schema language. You use XML Schema to describe the business data that flows in and out of the services.

Flow

Flows are a specific form of service implementation. They allow you to compose a service out of other services. We also refer to this type of flow as a *Microflow* or *Service Flow*.

A flow consists of service nodes, each node representing the invocation of a service operation. The service nodes are tied together by control links which indicate the sequence of execution and under which condition execution takes place.

The flow composition is described using the Flow Definition Markup Language (FDML).

Transformer

Transformers allow you to map between service messages. In fact, transformers allow you to map multiple input messages to a single output message.

Use them wherever you have to produce a new service message from service messages produced by other services. Transformers are mainly used in Data Mapping nodes within flows. This includes type conversions, splits, joins, and others.The transformation implementation is described in the form of XSLT.

For more comprehensive information about the services perspective, refer to the Help documents of the respective product.

4.4 WebSphere Studio Asset Analyzer

A key aspect of today's Web application development effort revolves around understanding and leveraging of services and services-oriented architecture

(SOA). SOA allows teams to reuse proven and reliable services instead of duplicating development effort. A service is defined as any piece of code that provides a service to another aspect of an application. It can be built in any language and may ultimately run in any environment. With the emerging new technology of “Grid Computing,” it will soon be possible to build applications where different services, which form parts of a solution delivery, can be executed on disparate systems participating in a grid computing infrastructure. For more information, see *Introduction to Grid Computing with Globus*, SG24-6895.

Services may also be harvested from existing application assets within the enterprise. The use of services provides significant benefits in many respects, including reuse, reliability, maintenance, scalability, and ultimately time to market of business applications.

The modernization of enterprise applications relies heavily on harvesting enterprise assets that can become services. When a particular capability becomes a service, that capability can be used again and again by many different applications. The goal is to deploy these as Web services in an execution environment.

One of the biggest stumbling blocks in harvesting enterprise assets from which to develop reusable services or components is, how to mine the nuggets that are hidden within the myriad application systems developed over the last 20 to 30 years. The original creators of legacy assets may be long gone from the organization, and all too often there is little or no documentation for older systems. These are just a few of the challenges you may encounter when attempting to harvest enterprise assets. WebSphere Studio Asset Analyzer (WSAA) was developed to satisfy the compelling need for some kind of technology infrastructure to assist in the harvesting effort.

The goal WSAA is trying to achieve is nothing new. For the last ten to fifteen years, maybe more, technology leaders like IBM have worked to build a special class of technology infrastructure called “Enterprise Repository” and some associated tools (source code scanners, among others).

WSAA provides the following information:

- **Application knowledge:** Rapid understanding of application components and construction, plus a high-level view or blueprint of an application.
- **Business knowledge:** Understanding of business and processing flow through drilling down into applications.
- **Change knowledge:** Identification of areas of the application that will change based on the requirements, right down to the line of code and including where all data items and indirect data items are used.

- **Data knowledge:** Databases and files that make up the application, plus data flows that provide a logical connection between programs, between processes, and between applications.
- **e-Business knowledge:** Ratings of program modules for their usefulness as connectors, plus prebuilt connectors and generated data definitions that can be imported into WebSphere development tooling like the Enterprise Access Builder or WebSphere Studio Enterprise Developer.

The first step in using WSAA is to scan the artifacts that you want WSAA to know about. These could be z/OS platform artifacts like COBOL and PL/I programs, BMS maps, JCL, IMS PSBs and DBDs, and region configurations for IMS and CICS. The information about these artifacts on z/OS are gathered by running *scanners* (tools provided by WSAA) on z/OS.

WebSphere Studio Application Analyzer also scans for more modern artifacts like J2EE applications, including war and ear files, EJB JAR files, Java source code and Class files, XML, HTML, JSP files and taglib files, C and C++ Source files, and so forth.

Artifacts on the z/OS platform can reside in source code management systems like SCLM, while on the Windows 2000 platform they can reside on Rational Clear Case or on a WebDAV server.

Figure 4-5 shows the six major components of WebSphere Studio Asset Analyzer; a description of the components follows the figure.

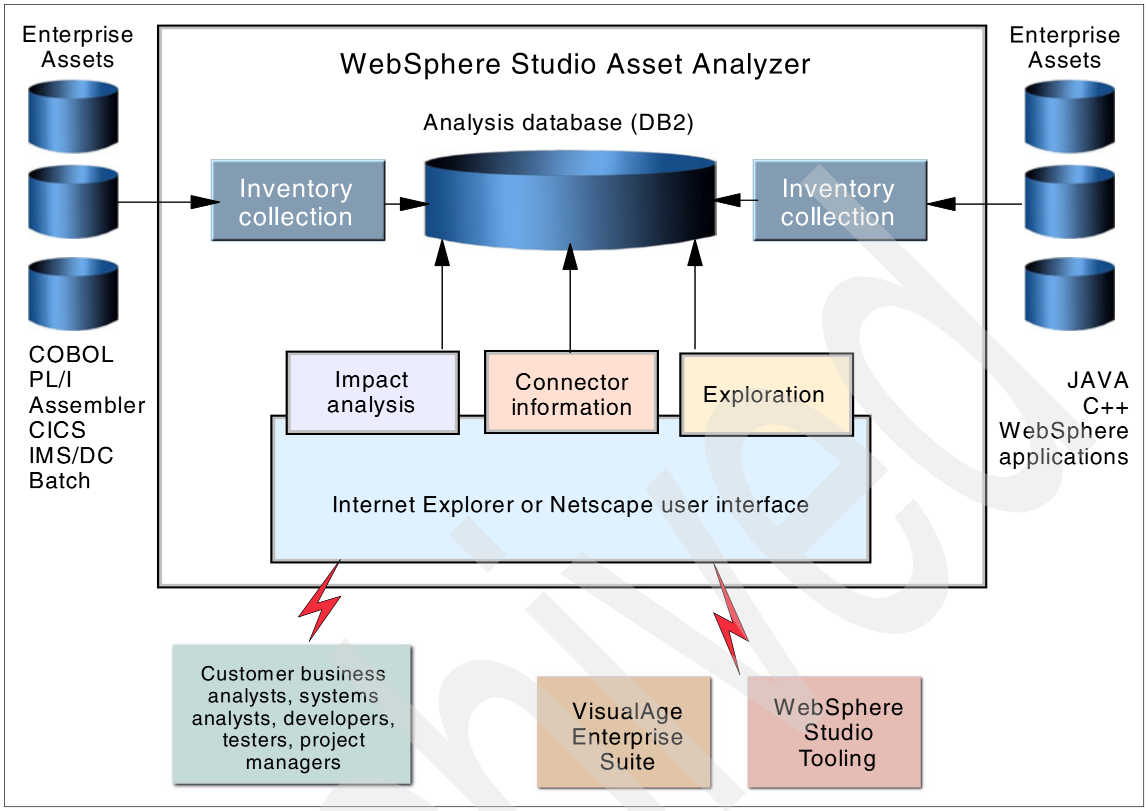


Figure 4-5 WebSphere Studio Asset Analyzer

- **Inventory collection**: This component inspects various z/OS artifacts to determine basic information about each, such as their programming language. Then a more in-depth analysis is performed to determine the relationships between all artifacts inspected.
- **Analysis database**: A DB2 database is the repository of meta-information about the artifacts that are inventoried.
- **Impact analysis**: With the inventory complete (or as complete as possible), impact analysis questions can be asked about the information in the database: What parts within an application, and outside an application, need to be considered when a data element is modified? What happens when a CALL signature is modified? What is affected when a specified block of code is changed?
- **Connector information**: Typically, n-tier applications, where one tier exists on a z/OS platform, need a bridge, or connector, between tiers. Discovering the information necessary for other tools, such as the Enterprise Access

Builder (EAB) of VisualAge for Java, to build these connectors can be very difficult and time consuming. The Connector Builder Assistant (CBA) component helps find and assemble the information set required to build connectors.

- **Exploration**: After you have taken an inventory, or even partially completed it, you can view the inventory. The inventory consists of a number of different artifacts (described in inventory collection) that have relationships to each other. These relationships are highlighted for you.
- **Web browser interface**: Almost all of the capabilities of WebSphere Studio Asset Analyzer are available through a Web browser interface. Using the IBM HTTP Server on z/OS and Net.Data® for DB2 access, this interface provides the window into WebSphere Studio Asset Analyzer capabilities.

For an online demonstration of WebSphere Studio Asset Analyzer, visit:

http://www.ibm.com/software/ad/studioedm/demo

4.5 XML repository

While WSAA provides the functionality to understand and harvest enterprise assets, we need another infrastructure to publish and manage the assets created from it and those that have been newly created. A new class of infrastructures known as XML repositories is emerging. Because they are design-time repositories, they provide a function different from that of UDDI repositories which provide execution-time Web services support.

An XML repository manages the development and deployment of XML assets (for example, XML schemas, DTDs, instance documents, style sheets, and WSDL documents), usually utilizing a Web-based interface. The repository enables an organization to take control of their XML assets for reuse throughout the enterprise. The interface should also provide, through the Internet, facilities for collaboration with suppliers, customers, trading partners, and industry groups.

4.5.1 Repository features

Repository for XML assets: An XML repository enables an organization to reduce development costs and take control of the exchange of messages by managing those XML assets at the document level and the component level (for example, elements, attributes, types, and model groups) from a centralized repository. The documents and components can be categorized, staged, browsed, and the inter-relationships searched to create a comprehensive view of an enterprise's XML assets. Figure 4-6 provides an overview of an enterprise XML repository's functional domain.

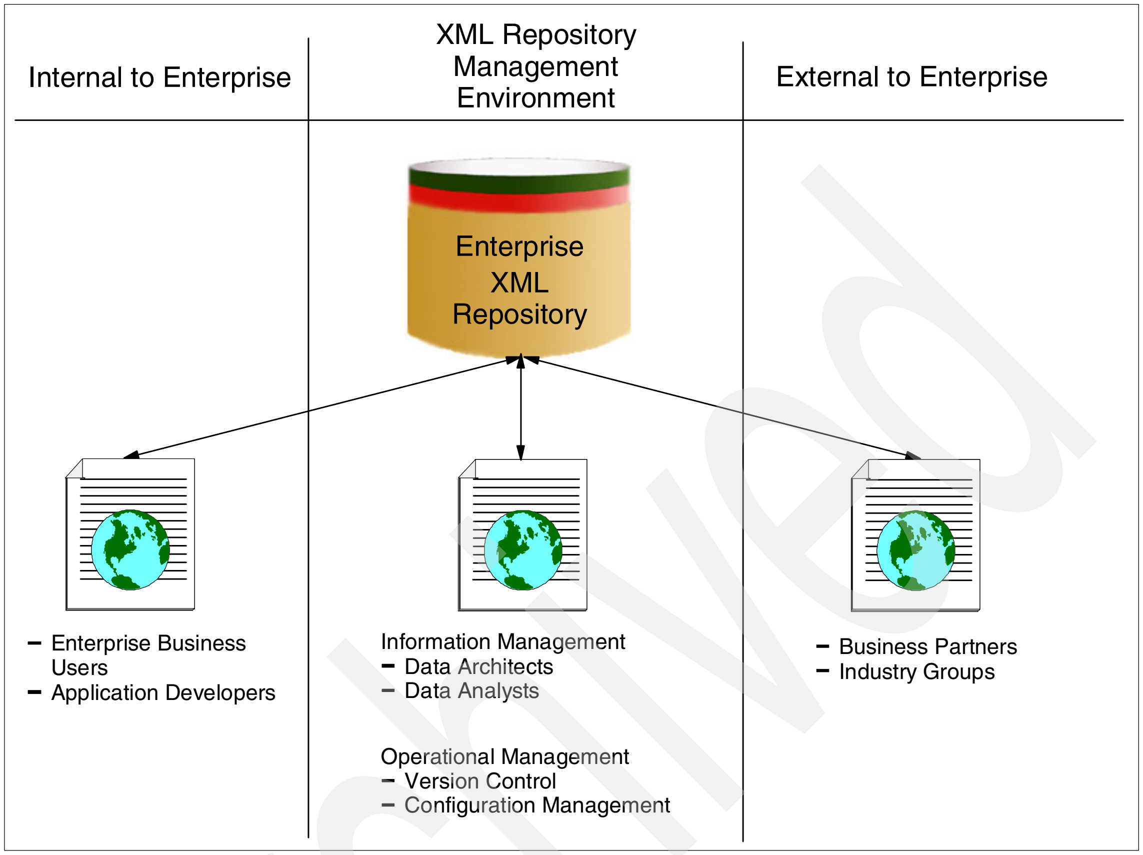


Figure 4-6 Enterprise XML Repository functions

XML vocabularies: e-business relationships are being built through the establishment of common vocabularies by companies and industry groups. An XML repository should enable the extensible power of XML by enabling the analysis of schemas and DTDs (the grammars) at the component-level, creating a data dictionary (vocabulary elements/tag definitions) of an enterprise's XML assets. The inter-relationships of these components can be browsed, searched, re-used, and re-constructed to create an infinite number of new, semantically different schemas.

Collaboration- Companies are developing XML schemas and DTDs jointly across different operational areas that are of mutual interest. An XML repository facilitates this inter-operation and cooperation through Web-based access to an organization's XML assets. It should provide a WebDAV protocol-based interface to achieve this collaboration.

Access, set by permission, creates a virtual workplace and allows people around the world to collaborate on the development of schemas and DTDs, which leads to e-business standards.

Administration - Managing the XML assets of disparate departments, divisions, and trading partners will require powerful administration facilities. An XML repository empowers organizations to create user-defined metadata and stages (see Staging in Repository functions) to which XML assets must adhere. This leads to a structured approach for organizing, searching, publishing, and migrating XML assets through the various stages of their life cycle (development, production, depreciation). It must give full control via a Web interface to set permissions, e-mail notification, security, and other essential administration functionality.

4.5.2 Repository functions

Some of the functions the repository should provide are the following:

Client interface - Provide all WebDAV-compliant devices, for example, a browser, an application development environment like WebSphere Studio Enterprise Developer, stand-alone XML editors like TurboXML, XMLSpy, and so forth.

Indexing - Performed at two levels, the asset level (schemas, DTDs, and others) and the component level (element, attribute, types, and so forth).

Searching - Using the robust metadata captured, information can be queried with a high degree of granularity at the document level or at the component level.

Namespace management - Registry of corporate and industry standard Namespaces. URL mapping and resolution of XML vocabularies.

Staging - The administrator can define a series of stages that an XML asset will move through during its life cycle (for example, testing - production - depreciation). Each transition from one stage to another may have associated rules of acceptance.

"Diffing" - The ability to track changes between revisions, in the form of highlighting and underscoring to document changes between versions of a file. This gives authorized users the streamlined ability to audit (approve/reject) changes made to an asset to ensure a smooth transition through the life cycle stages.

Reporting - A comprehensive documentation view of an asset. This enables XML assets or combinations of assets to be turned into documents for powerful reporting and communication throughout an organization, with trading partners, or with industry groups.

Component-level composability - The XML artifact development environment should be integrated with the repository. In other words, component tools environments like WebSphere Studio Enterprise Developer, XML editors, and so forth should be able to have an interface with the Repository

Asset relationship tracking - The repository should track the relationship between documents and their components and allow these relationships to be easily viewed.

Data dictionary/Type management - (content models, data types) - Enterprises should have the ability to manage, index, and reuse vocabulary components (Types) at the corporate or industry level.

Source control and version management system - The repository should provide full source management control of XML assets with check-in/check-out and versioning throughout the life cycle of the asset.

Categories - This hierarchal perspective provides an organization with the ability to logically view and understand the structure and status of its XML assets. Management facilities are available to deactivate, re-name, and add categories as necessary.

Automated notification - People can subscribe to relevant XML files and be automatically notified via e-mail or on their repository home page as XML files transition through user-configured life cycles.

If you are going to use XML, you must have a repository to even have a hope of managing your DTDs/Schemas and XML messages.The problems you face without a repository include: not knowing where an element was used, inability to document the physical implementation of an element or attribute, inability to sort out how DTDs were related to each other and to XML documents, XML documents constructed without mandatory elements or attributes, invalid values in attributes, and the inability to version DTDs. All these issues can be resolved by an XML repository.

5

XML and Enterprise COBOL

In this chapter we provide an introduction to the XML capabilities of IBM Enterprise COBOL for z/OS and OS/390 V3R1.

We describe how to promote data interchange in XML format between traditional legacy COBOL programs with minimal effort. We discuss the interaction between COBOL and Java, the XML parser integrated in COBOL, and how to use WebSphere Studio Enterprise Developer to let legacy COBOL programs accept and respond in XML format.

5.1 Overview

IBM Enterprise COBOL for z/OS and OS/390 V3R1 lets you integrate your traditional COBOL programs into the e-business world by enabling you to invoke an XML parser from the program and to operate with Java components across distributed applications.

The ability to have an XML parser that is accessible to your programs and that can be invoked with one COBOL sentence, is the key for all business processes. This XML parser lets you promote the exchange and use of data in standardized formats, including XML and Unicode, and enables you to reuse existing applications in WebSphere and traditional z/OS environments.

In this chapter we describe how the COBOL XML parser operates, and based on this XML parser, we discuss how to work with the new tool WebSphere Studio Enterprise Developer to generate the necessary contents to adapt our traditional COBOL programs to the XML world.

5.2 COBOL and Java interoperation

Enterprise COBOL provides object-oriented syntax to facilitate the interoperation of COBOL with other languages, such as Java and C++. For example, you can instantiate Java classes from COBOL programs, invoke methods on Java objects, and define Java classes that can be instantiated in Java or COBOL.

Example 5-1 is an Object Oriented COBOL Program in which we are defining a class called DavidDogs.

Example 5-1 Sample COBOL program

```
(1)  cbl dll,thread,pgmname(longmixed)
(2)  Identification division.
(3)  Class-id. DavidDogs inherits Base.
(4)  Environment Division.
(5)  Configuration section.
(6)  Repository.
(7)      Class Base      is "java.lang.Object"
(8)      Class DavidDogs is "DavidDogs".
(9)  Identification division.
(10) Object.
(11)  Data division.
(12)  Working-storage section.
(13)  01 DogName pic X(6).
(14)  Procedure Division.
(15)   Identification Division.
```

```
(16)   Method-id. "init".
(17)   Data division.
(18)   Linkage section.
(19)   01 inDogName pic X(6).
(20)   Procedure Division using by value inDogName.
(21)     Move inDogName to DogName.
(22)   End method "init".
(23)   Identification Division.
(24)   Method-id. "getDog".
(25)   Data division.
(26)   Linkage section.
(27)   01 DogResponse pic X(10).
(28)      02 Name   pic X(6)
(28)      02 Filler pic X.
(29)      02 Response pic X(9).
(30)   Procedure Division returning DogResponse.
(31)     Move DogName to Name.
(32)     Move "Guau Guau" to Response.
(33)   End method "getDog".
(34) End Object.
(35) End class DavidDogs.
```

In the following discussion we mention some key aspects of the code shown in Example 5-1, and explain why we have structured it in this particular way. The numbers in parentheses refer to the line numbers in the code.

- (1) The first line in our program contains instructions to the compiler. With this release, compilation of COBOL programs containing CICS statements no longer requires a separate translation step. An integrated translator approach is an alternative to using the separate translator. With the integrated translator approach, the COBOL compiler handles both native COBOL and imbedded CICS statements in the source program. The recommendation is to include COBOL compiler options in our COBOL source file.

  We use option `dll` every time we generate an object module that is enabled for Dynamic Link Library support (DLL). Our program has an OO syntax, so we need this option activated because the output of a COBOL class definition is a dll residing in UNIX System Services.

  Option `thread` indicates that our program is enabled to run in a Language Environment with multiple POSIX threads running.

  Option `pgmname(longmixed)` indicates that our program name has to be processed as it is, without truncation or changing it to uppercase letters. (This option is important if you are working with Java classes, as well, because Java classes are in lowercase most of the times.)

- (2) In an OO COBOL program we need an `Identification division` to provide the name of the class we are defining. In this case we are defining a class with the name *DavidDogs*, so we code an Identification division followed by a Class-id clause (3) where we specify the class name (DavidDogs) and that this class inherits from the `Base` class. The Base class is an Object Java class (java.lang.Object), so we are mixing Java and COBOL classes in our inheritance hierarchy. Anyway, all classes we define must inherit directly or indirectly from this class (java.lang.Object), and the name we have used (Base) is the recommended name for the Object class.

- (4) In the `Environment Division` we map our internal classes (defined in the current COBOL program) with external classes known in our runtime environment. In this case, we declare a `Configuration section`, and inside this section a `Repository` section (6), where we declare that the Base class is `java.lang.Object` (7) (the top class in the Java hierarchy), and DavidDogs (8) is externally known as it is, DavidDogs.
- (9) An `Identification division` section is declared again, but this time it is followed by an `Object` section (10), so here we describe the instance data that the class needs. After Object we declare a `Data division` (11), and after that a `Working-storage section` (12), in which we specify fields shared between all methods described in this class. These fields are private to the class, so if you want to make them accessible by other classes, you have to create a **`set`** or **`get`** method to do it. In our case we have defined a field (13) in which we are going to store the name of our dog.
- In (14) we declare a `Procedure division` where we enumerate the methods in use by our class. The first method is `init` (16); in this case it awaits an input parameter `inDogName` (19), declared in the `Linkage section`. Our init method only stores (21) this input parameter in the field DogName previously declared (13), so it can be shared among all methods in this class.
- In (24) we define a `getDog` method that gives the response of our dog (the name of our dog is taken from the `DogName` field previously declared in the Working-Storage section, the value of which was stored by the ain method).
- In (34) we end the Object description, and in (35) we end the Class declaration.

For the compilation of this COBOL class, we use a utility called **`cob2`**, which resides in UNIX System Services, commonly in /usr/lpp/cobol/lib. The resulting dll will reside in the UNIX System Services as well.

Try with following commands:

```
cob2 -c -qdll,thread DavidDogs.cbl
cob2 -bdll -o libDavidDogs.so DavidDogs.o
                     /usr/lpp/java/IBM/J1.3/bin/classic/libjvm.x
                     /usr/lpp/cobol/lib/igzcjava.x
javac DavidDogs.java
```

(This file is generated in the first step, and it defines a Java class that involves the generated COBOL object.)

As you can see, the program is very similar to the concept of Java classes. Once you have compiled this class, it can be invoked in the same way as a Java class. Object instances of COBOL classes can be created from Java or COBOL, and the methods of the classes can be invoked from Java classes or COBOL classes.

On the other hand, you can invoke a traditional COBOL program from a COBOL class using the known `CALL` instruction. This lets you include traditional COBOL programs in the OO programming world.

Object-oriented COBOL programs and z/OS Java programs are always run in a UNIX System Services environment.

Another way to access Java classes from COBOL classes is the JNI (Java Native Interface). In this case you have to copy a copybook called *JNI.cpy* in the Linkage section of your COBOL program. You can find this copybook in *Enterprise COBOL for z/OS and OS/390 Programming Guide* , SC27-1412. This copybook maps COBOL data definitions to JNI data types, and it has the JNINativeInterface structure, which has pointers to the different callable services of JNI. From this structure you obtain the pointers, and then with CALL operations you can invoke the services. For more details on how to code you JNI interface in your COBOL program, refer to Chapter 29 in the Programming Guide.

5.3 XML support in Enterprise COBOL for z/OS

Enterprise COBOL for z/OS and OS/390 V3R1 gives you some basic XML capability in your COBOL programs. You have an integrated XML parser in COBOL, so with a simple instruction you can invoke it and map XML data structures into COBOL structures. Example 5-2 is a basic sample, in which we have coded an XML parser that makes a trace of the incoming XML document. This XML document is stored in an MVS dataset, but commonly XML data come from a CICS communication area, or an IMS message queue.

Example 5-2 Processing an XML document in a COBOL program

```
Process flag(i,i)
 Identification division.
 Program-id. xml1.
 Environment division.
 Input-output section.
 File-control.
*
     Select xml-file
         Assign to xmlinput
         File status is xmlfile-status.
```

```
*
 Data division.
  File section.
  Fd xml-file
     label records are standard
     recording mode is f
     record contains 80 characters
     block contains 0 records
     data record is xml-file-input.
  01 xml-file-input pic x(80).
  Working-storage section.
*************************************************************
* We define a table to load into storage the full XML document.
*************************************************************
   01 current-element pic x(30).
   01 table-1.
      02 record-1 occurs 50 times pic x(80) value spaces.
   01 switches.
      05 xmlfile-status pic xx value '00'.
         88 inputxml-success value '00'.
   01 eof pic x value ' '.
   01 xml-document-length pic 999 value zeros.
   01 ind pic 999 value zeros.
 Procedure division.
  mainline section.
*************************************************************
* Procedure fileread reads full XML document into storage   *
*              before it is processed                       *
*************************************************************
perform fileread.
     XML PARSE table-1 PROCESSING PROCEDURE xml-handler
       ON EXCEPTION
         display 'XML document error ' XML-CODE
       NOT ON EXCEPTION
         display 'XML document successfully parsed'
     END-XML
     goback.
  fileread.
     open input xml-file.
     if not inputxml-success
        display 'error opening xml input file'
        exit.
     perform readline until eof = '1'.
     exit.
  readline.
     compute ind = ind + 1.
     move spaces to record-1(ind).
     read xml-file into record-1(ind) at end move '1' to eof.
     exit.
  xml-handler section.
     evaluate XML-EVENT
*************************************************************
* Insert your code in the event you need. We only have     *
*     entered a display to check the parsing of            *
*             the document.                                *
*************************************************************
      when 'VERSION-INFORMATION'
        display 'version information tag:<' XML-TEXT '>'
      when 'ENCODING-DECLARATION'
        display 'encoding declaration tag:<' XML-TEXT '>'
      when 'STANDALONE-DECLARATION'
        display 'standalone declaration tag:<' XML-TEXT '>'
*==>Order XML events most frequent first
      when 'START-OF-ELEMENT'
        display 'Start element tag:<' XML-TEXT '>'
        move XML-TEXT to current-element
      when 'ATTRIBUTE-NAME'
        display 'Attribute name tag:<' XML-TEXT '>'
      when 'ATTRIBUTE-CHARACTERS'
        display 'Attribute characters tag:<' XML-TEXT '>'
      when 'ATTRIBUTE-CHARACTER'
        display 'Attribute character tag:<' XML-TEXT '>'
      when 'CONTENT-CHARACTERS'
        display 'Content characters:<' XML-TEXT '>'
      when 'END-OF-ELEMENT'
        display 'End elementtag:<' XML-TEXT '>'
        move spaces to current-element
      when 'START-OF-DOCUMENT'
        display 'Start of document'
      when 'END-OF-DOCUMENT'
        display 'Attribute value character:<' XML-TEXT '>'
      when 'START-OF-CDATA-SECTION'
        display 'Start of CData:<' XML-TEXT '>'
      when 'END-OF-CDATA-SECTION'
        display 'End of CData:<' XML-TEXT '>'
      when 'CONTENT-CHARACTER'
        display 'Content character:<' XML-TEXT '>'
      when 'PROCESSING-INSTRUCTION-TARGET'
        display 'PI target:<' XML-TEXT '>'
      when 'PROCESSING-INSTRUCTION-DATA'
        display 'PI data:<' XML-TEXT '>'
      when 'COMMENT'
        display 'Comment:<' XML-TEXT '>'
      when 'EXCEPTION'
        compute xml-document-length = function length(XML-TEXT)
        display 'Exception ' XML-CODE 'at offset'
            xml-document-length '.'
      when other
```

```
       display 'Unexpected XML event:' XML-EVENT '.'
     End-evaluate.
     exit.
 End program xml1.
```

This program is very simple, and its only purpose is to provide a trace to see the event sequence, but of course is very easy to add **move** instructions in the **when** declarations, and to move the contents of XML-TEXT to Working storage fields and manipulate them as desired.

To execute the program, we could use the JCL in Example 5-3, where XML1 is the result of the compilation of the program listed previously.

Example 5-3 Execution JCL

```
//DAVID1 JOB (POK,999),DAVID,MSGLEVEL=(1,1),MSGCLASS=X,
//  CLASS=A,NOTIFY=&SYSUID
//*
//PAS001  EXEC PGM=XML1
//STEPLIB DD DSN=DAVID.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//XMLINPUT DD DSN=DAVID.XML(XMLDOC1),DISP=SHR
//SYSIN   DD DUMMY
```

To execute the program, we use a sample XML document called XMLDOC1, stored in a dataset called DAVID.XML. The contents of the XML document are shown in Example 5-4.

Example 5-4 XMLDOC1

```
<?xml version="1.0" encoding="ibm-1140" standalone="yes"?>
<!--This document is just an example-->
<MyDogs>
  <Name>Maripuri</Name>
  <Breed type="Tekel" size="small" />
  <Name>Golfa</Name>
  <Breed type="Schnawzer" size="small" />
  <Name>Lara</Name>
  <Breed type="Qart-Hadast" size="large" />
  <Name>Chispa</Name>
  <Breed type="Setter" size="large" />
  <Name>Mini</Name>
  <Breed type="Maltes" size="small" />
  <?Caution with Maripuri?>
  <meat>What you want</meat>
</MyDogs>
```

The results of the execution are shown in Example 5-5.

Example 5-5 Output from the sample COBOL program

```
Start of document
version information tag:<1.0>
encoding declaration tag:<ibm-1140>
standalone declaration tag:<yes>
Comment:<This document is just an example>
Start element tag:<MyDogs>
Content characters:<
Start element tag:<Name>
Content characters:<Maripuri>
End elementtag:<Name>
Content characters:<
Start element tag:<Breed>
Attribute name tag:<type>
Attribute characters tag:<Tekel>
Attribute name tag:<size>
Attribute characters tag:<small>
End elementtag:<Breed>
Content characters:<
Start element tag:<Name>
Content characters:<Golfa>
End elementtag:<Name>
Content characters:<
Start element tag:<Breed>
Attribute name tag:<type>
Attribute characters tag:<Schnawzer>
Attribute name tag:<size>
Attribute characters tag:<small>
End elementtag:<Breed>
Content characters:<                                         >
Start element tag:<Name>
Content characters:<Lara>
End elementtag:<Name>
Content characters:<
Start element tag:<Breed>
Attribute name tag:<type>
Attribute characters tag:<Qart-Hadast>
Attribute name tag:<size>
Attribute characters tag:<large>
End elementtag:<Breed>
Content characters:<                                       >
Start element tag:<Name>
Content characters:<Chispa>
End elementtag:<Name>
Content characters:<
Start element tag:<Breed>
```

```
Attribute name tag:<type>
Attribute characters tag:<Setter>
Attribute name tag:<size>
Attribute characters tag:<large>
End elementtag:<Breed>
Content characters:<                                            >
Start element tag:<Name>
Content characters:<Mini>
End elementtag:<Name>
Content characters:<
Start element tag:<Breed>
Attribute name tag:<type>
Attribute characters tag:<Maltes>
Attribute name tag:<size>
Attribute characters tag:<small>
End elementtag:<Breed>
Content characters:<                                            >
PI target:<Caution>
PI data:<with Maripuri>
Content characters:<                                                        >
Start element tag:<meat>
Content characters:<What you want>
End elementtag:<meat>
Content characters:<                                                     >
End elementtag:<MyDogs>
Attribute value character:<>
XML document successfully parsed
```

Remember that we have coded this program to read from an external MVS dataset, but a common situation could be to read XML document from CICS COMMAREA. In "XML converters for traditional COBOL programs" on page 121, we explain how to integrate XML in our COBOL program working in a CICS environment.

With the instruction **XML PARSE** you specify the procedure where you handle the different events that occur while parsing the XML document. In our case we specify `xml-handler` as the handler procedure, so a few lines later we code this procedure with an instruction **EVALUATE**, which lets us check the event type that is happening. In this **EVALUATE** instruction, we use XML-EVENT special register to check the type of event. There are some special registers to receive and pass information to the parser, and XML-EVENT is one of them. Some of the events controlled by this special register are:

- **START-OF-DOCUMENT**: Occurred once, at the beginning of the XML document.
- **VERSION-INFORMATION**: If you have the optional *version* declaration on your XML document. In this case, the version is stored in another special register, XML-TEXT.

- **ENCODING-DECLARATION**: If you have specified the *encoding* attribute in your XML document. Again, XML-TEXT contains the value.
- **STANDALONE-DECLARATION**: If you specify the *standalone=yes|no* attribute, in which case this value is stored in XML-TEXT.
- **DOCUMENT-TYPE-DECLARATION**: If you include a DTD declaration in the XML document.
- **COMMENT**: If you include some element in the form <!-- .... -->
- **ATTRIBUTE-NAME**: Once per attribute name in the element specification. XML-TEXT contains the name of the attribute.
- **ATTRIBUTE-CHARACTERS**: Each fragment of the attribute value is going to be processed by this event, and its text is going to be stored in XML-TEXT.
- **ATTRIBUTE-CHARACTER**: For the predefined entity name.
- **END-OF-ELEMENT**: For the end of each processed element.
- **PROCESSING-INSTRUCTION-TARGET**: For each *PI* after character <?.
- **PROCESSING-INSTRUCTION-DATA**: After the first word of a *PI*, the rest is considered data.
- **CONTENT-CHARACTERS**: Character data between start and end tags of an element.
- **CONTENT-CHARACTER**: For entity references in character data.
- **END-OF-ELEMENT**: Final tag in an element.
- **START-OF-CDATA-SECTION**: Start of a CDATA declaration.
- **END-OF-CDATA-SECTION**: End of a CDATA declaration.
- **END-OF-DOCUMENT**: End tag in the XML document.

As you can see, the XML parser in COBOL is an event-based parser, so the way to parse your documents is going to be very dependent on the way you manage the events.

The original XML document usually comes from CICS, MQSeries, or similar, but if you want to process a document from a file, you have to load it in a data structure defined in the Working Storage section, prior to the event handling.

One important point is that XML COBOL parser is not a conforming XML processor according to the standards. While it parses your XML document, it makes some well formedness controls, but in general, it is not a validating parser. It is not DTD compliant. For an explanation of the validations covered by COBOL XML parser, refer to *Enterprise COBOL for z/OS and OS/390 Programming Guide V3R1,* SC27-1412.

5.4 WebSphere Studio Enterprise Developer & COBOL

On this section we give a brief overview of the new facilities introduced by WebSphere Studio Enterprise Developer, a new development tool that helps in the integration of COBOL programs in the Web Services environment.

We are going to focus our discussion on XML facts (the final point of this section), but we are also going to present some other important aspects of the new tool.

If you want further explanation about WebSphere Studio Enterprise Developer and how to work with Struts or EGL programs, refer to the redbook *Legacy Modernization with WebSphere Studio Enterprise Developer*, SG24-6586.

5.4.1 New options in WebSphere Studio Enterprise Developer

Struts applications

Struts is an open source framework for building Web applications. It is part of the Jakarta project, sponsored by the Apache Software Foundation.

Struts helps developers build Web applications by supporting the generation of many components included in the MVC (Model, View, Controller) model. These generated components can be used later in an application, avoiding the need to implement them. The MVC model uses servlets and JSPs as tools to implement this scenario. The framework provides classes that can be extended by developers to implement their final application.

Figure 5-1 shows a possible real-world scenario utilizing Struts.

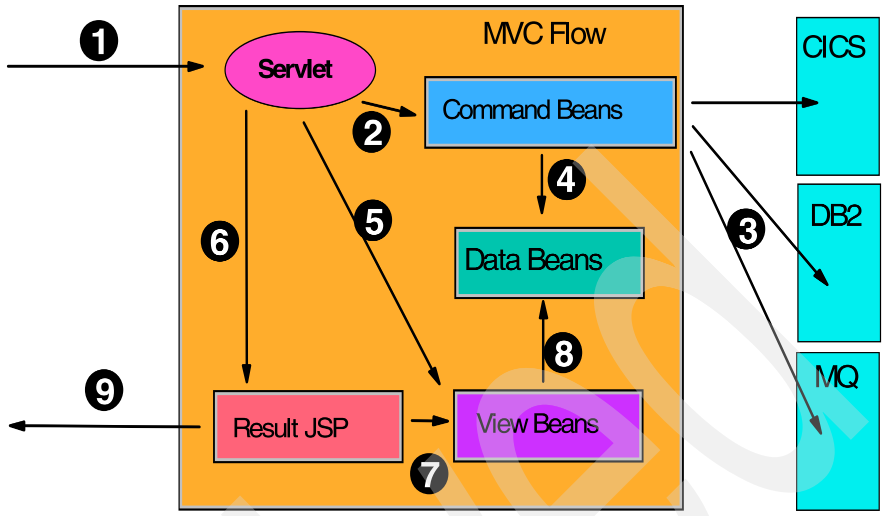


Figure 5-1 A possible scenario utilizing Struts

The numbered steps in the process are as follows:

1. A client from the browser sends a request to the server (it may be the HTTP handler in WebSphere Application Server). This request is received by a servlet that is running in a Web Container.
2. The servlet is an extension of a class provided by the Struts framework, called org.apache.struts.action.ActionServlet. This servlet routes the HTTP request to other Action objects deciding what business logic is going to be performed. These Action objects are extensions of another class provided by Struts called org.apache.struts.action.Action; they are the interface with our legacy application. The servlet uses command beans to process the request. These command beans can connect with legacy programs running, for example, in CICS Transaction Gateway. This servlet has an XML properties file where you can customize to specify messages, login procedures, and so forth.
3. These command beans can access backend environments, for example CICS, interchanging data with it through the COMMAREA. (Later we will describe how a traditional COBOL program can be updated to read XML files instead of CICS binary data format.)
4. The results of commands are data beans (JavaBeans); for example, the result of a CICS transaction is a COMMAREA represented in a Java record. If you have prepared a COBOL program running on CICS to respond with XML, data read from the COMMAREA can be an XML response.
5. The servlet allocates view beans that are used to process and format the data stored in the data beans into formats suitable for HTML output. (This is

optional, but sometimes required data beans may be predefined). The Struts view is made up of various components; the main one being the Java Server Page (JSP). The JSP is not a Struts component (you have to make it), but Struts provides you with some helpful elements: *form beans*, an extension of a Struts class called org.apache.struts.action.ActionForm, responsible for retaining and validating data entered by the client; and a *JSP tag library*, which includes HTML tags, beans tags, logic tags, and template tags to design our JSP.

6. The servlet invokes the JSP to generate the HTML output.
7. The JSP uses the view beans to retrieve formatted results.
8. The view beans use the data beans to process and format the results.
9. The JSP generates the HTML result page.

From an MVC point of view, Struts provides:

Model: This is the most important part of the application because it represents the business logic. The model captures the state of the application. Here Struts does not provide anything because it is your own application. Usually this model is represented with Enterprise Java Beans, but it could be a legacy program.

View: The view formats the data from the data bean and converts them into a valid HTML format to respond to the client. It does not include knowledge of the model or controller. Here Struts provides an org.apache.struts.action.Action class that developers use to create form beans, which are used to pass data between the controller and view. In addition, Struts provides four tag libraries (HTML tag library, beans tag library, logic tag library, and template tag library) that the developer can use in his JSPs to develop the interactive part of the application.

Controller: The controller manages the execution flow of the application, passing appropriate information between the model and the view. Struts provides an org.apache.struts.action.Action class that developers use to create the classes that control the flow of the application. Also, Struts provides an org.apache.struts.action.ActionServlet class to implement a controller servlet.

Struts provides utility classes to support XML parsing, automatic population of JavaBeans properties based on the Java reflection APIs, and internationalization of prompts and messages.

WebSphere Studio Enterprise Developer has a wizard that generates all Struts components, so it is easy to use these components in your applications. To learn more about this wizard and Struts, refer to *Legacy Modernization with WebSphere Studio Enterprise Developer*, SG24-6586.

EGL Programs

EGL (Enterprise Generation Language) is a high-level programming facility that can be used to create programs in a generic language, and then generate Java code or COBOL code. To access the generated code, the generation process can also generate Java wrappers that can be included in Java programs that have to access EGL generated code, such as Struts action classes.

When the EGL generator generates COBOL code, it can also generate a Java wrapper class that uses a J2EE Connector resource adapter to access the COBOL code through a CICS transaction gateway.

The generated COBOL code can be deployed easily to the z/OS host with a wizard provided by WebSphere Studio Enterprise Developer. The wizard helps to handle the file transfer of the COBOL program, the DB2 bind, the compilation, and the linkedit.

With the EGL support, you develop and test the EGL program; then, from this program you can generate the Java or COBOL source code; and then you can prepare all steps involved in the generation of the executable from the tool.

You use Struts to work with EGL because Struts creates many of the components (action servlet, JSPs, and so forth) that you need in your EGL program.

5.4.2 XML converters for traditional COBOL programs

With WebSphere Studio Enterprise Developer, you can easily convert traditional CICS-COBOL applications to the XML messaging format and introduce them in the Web services world. To allow XML documents to flow through to those business programs, you have to develop additional extensions to the original COBOL program. These extensions are used to convert XML messages into traditional data format for your program, and traditional responses from your program into XML messages. These extensions use the new XML capability described in “XML support in Enterprise COBOL for z/OS” on page 111.

WebSphere Studio Enterprise Developer helps you in the development of these extensions, saving you the laborious and error-prone task of coding the processing procedure for the XML PARSE verb. To build them you only have to follow a simple series of steps. First of all, you import the COBOL source program file into the WebSphere Studio Enterprise Developer, and then the tool generates a set of COBOL programs called “XML converters” (Inbound and Outbound), based on the original binary (CICS traditional) interface. The tool also generates a template COBOL program called “Converter driver” that illustrates how to invoke the converters. This converter driver manages the invocation flow among new converters and your original COBOL program.

For example, if you have the COBOL source for a traditional CICS program that currrently is running on your installation and that it is part of an entire application, and this COBOL program is the front-end for the incoming/outgoing requests to the application, *this* is the program you have to convert. You can do an FTP of your source COBOL file to your PC (for example, with the *Transfer* option in a PCOM session), and once you have it there, you can import this text file to the WebSphere Studio Enterprise Developer. To better illustrate this, let’s consider the COBOL source file shown in Example 5-6 (obtained from the samples that come with WebSphere Studio Enterprise Developer).

Example 5-6 Sample COBOL program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. DFHOACTD.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

    EXEC SQL INCLUDE SQLCA  END-EXEC.

01 TMP            PIC X(40) VALUE SPACES.
01  SQL-MESSAGE             VALUE SPACES.
    05  MSG       PIC X(10).
    05  RC        PIC X(10).
01  SQLERROR.
    05  MSG2      PIC X(10) VALUE 'SQLERRM:  '.
    05  ERRM      PIC X(5).
    05  ERMC      PIC X(70) VALUE SPACES.
01  SQLSTAT.
    05  MSG3      PIC X(10) VALUE 'SQLSTATE: '.
    05  SQLSTATT  PIC X(5).
01  SQLERRORP.
    05  MSG4      PIC X(10) VALUE 'SQLERRP:  '.
    05  SQLERP    PIC X(8).

01  ABEND-MESSAGE.
    05  MSG5      PIC X(12) VALUE 'ABEND CODE: '.
    05  ABEND-CODE PIC X(4).

01  HV-DATA.
    05 HV-CUSTNO       PIC S9(9) COMP VALUE +0.
    05 HV-ACCTNO       PIC S9(9) COMP VALUE +0.
    05 HV-BALANCE      PIC S9(6)V9(2) COMP-3 VALUE +0.

LINKAGE SECTION.
01  DFHCOMMAREA.
```

```
    05  CUSTNO     PIC S99999.
    05  ACCTNO     PIC S99999.
    05  BALANCE    PIC S9999V99.

PROCEDURE DIVISION.
START-PARA.

    MOVE 999999999 TO ACCTNO
    MOVE 'SQLCODE: ' TO MSG.
    MOVE 'DFHOACTD PROGRAM STARTED. ' TO TMP.
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(TMP)
      LENGTH(40)
      END-EXEC.

    MOVE CUSTNO TO HV-CUSTNO.
    MOVE 'SEARCHING WITH CUST NO:' TO TMP.
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(TMP)
      LENGTH(40)
      END-EXEC.
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(CUSTNO)
      LENGTH(5)
      END-EXEC.
    EXEC CICS HANDLE ABEND
      LABEL(ABEND-PARA)
      END-EXEC.

    EXEC SQL SELECT ACCT_NUMBER, BALANCE
             INTO :HV-ACCTNO, :HV-BALANCE
             FROM ACCOUNT
             WHERE BALANCE IN
             (SELECT MAX(BALANCE) FROM ACCOUNT
             WHERE CUST_ID = :HV-CUSTNO) END-EXEC.

    MOVE SQLCODE TO RC.
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(SQL-MESSAGE)
      LENGTH(20)
      END-EXEC.

    MOVE SQLERRML TO ERRM.
    MOVE SQLERRMC TO ERMC.
    EXEC CICS WRITEQ TD QUEUE ('CSMT') FROM (SQLERROR)
      LENGTH(85) END-EXEC.
    MOVE SQLERRP TO SQLERP.
    EXEC CICS WRITEQ TD QUEUE ('CSMT') FROM (SQLERRORP)
      LENGTH(18) END-EXEC.
    MOVE SQLSTATE TO SQLSTATT.
    EXEC CICS WRITEQ TD QUEUE ('CSMT') FROM (SQLSTAT)
```

```
      LENGTH(15) END-EXEC.

    IF  SQLCODE EQUAL ZERO
        MOVE HV-ACCTNO TO ACCTNO
        MOVE HV-BALANCE TO BALANCE
    END-IF.
    GO TO RETURN-PARA.
ABEND-PARA.
    EXEC CICS HANDLE ABEND
      CANCEL
      END-EXEC.
    EXEC CICS ASSIGN
      ABCODE(ABEND-CODE)
      END-EXEC.
    EXEC CICS WRITEQ TD QUEUE ('CSMT') FROM (ABEND-MESSAGE)
      LENGTH(16) END-EXEC.
RETURN-PARA.
    MOVE 'DFHOACTD PROGRAM STOPPED.' TO TMP.
    EXEC CICS WRITEQ TD QUEUE('CSMT')
      FROM(TMP)
      LENGTH(40)
      END-EXEC.

    EXEC CICS RETURN
    END-EXEC.
```

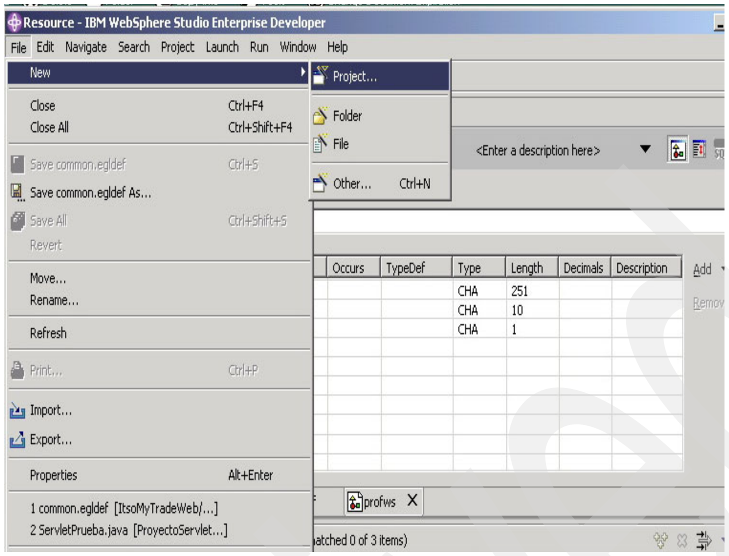
This program is part of a CICS application to manage customers and accounting information, and it is in charge of the accounting details for the customers. The application has two main modules working as the front end for incoming/outgoing requests, but for this discussion we are going to concentrate only on one module. It uses DB2 to store data, and the access to the application is now via a 3270 terminal, using CICS. We want to convert this program to be able to accept and respond with XML messages.

Use the following steps to accomplish this conversion:

1. Do an FTP or TRANSFER from the 3270 session (TSO) to the PC of the source COBOL program. If the file you have downloaded is called DFH0ACTD, rename this file to DFH0ACTD.cbl, so you now have a file on the PC called E:\ejemplos\DFH0ACTD.cbl.

   Start WebSphere Studio Enterprise Developer.

2. From the main screen in WebSphere Studio Enterprise Developer, open the Resource perspective by selecting **Window** → **Open Perspective** → **Resource**. Create a new Project for COBOL Resources by selecting **File** → **New** → **Project** (Figure 5-2).


Resource - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Launch Run Window Help
New
Close Ctrl+F4
Close All Ctrl+Shift+F4
Save common.egldef Ctrl+S
Save common.egldef As...
Save All Ctrl+Shift+S
Revert
Move...
Rename...
Refresh
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
1 common.egldef [ItsoMyTradeWeb/...]
2 ServletPrueba.java [ProyectoServlet...]
Project...
Folder
File
Other... Ctrl+N
<Enter a description here>
Add
Remov
profws
atched 0 of 3 items)

Occurs	TypeDef	Type	Length	Decimals	Description
		CHA	251		
		CHA	10		
		CHA	1		

Figure 5-2 Create a new project

3. On the next screen, select **Simple** → **Project**, and click **Next** (Figure 5-3).

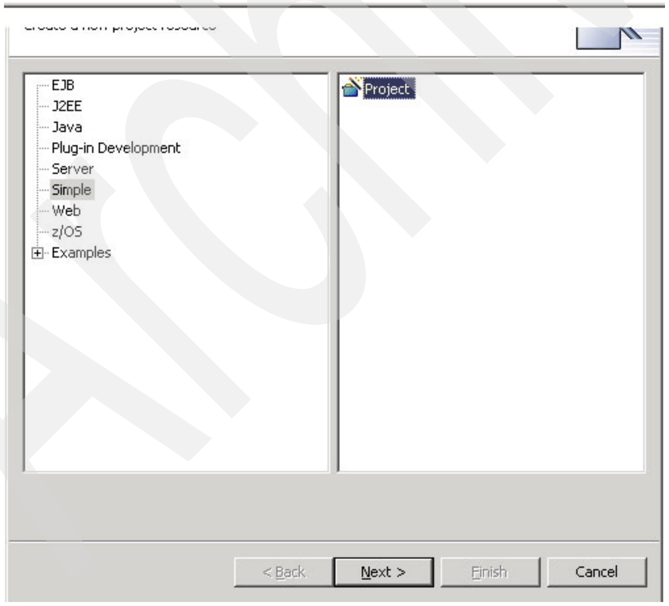
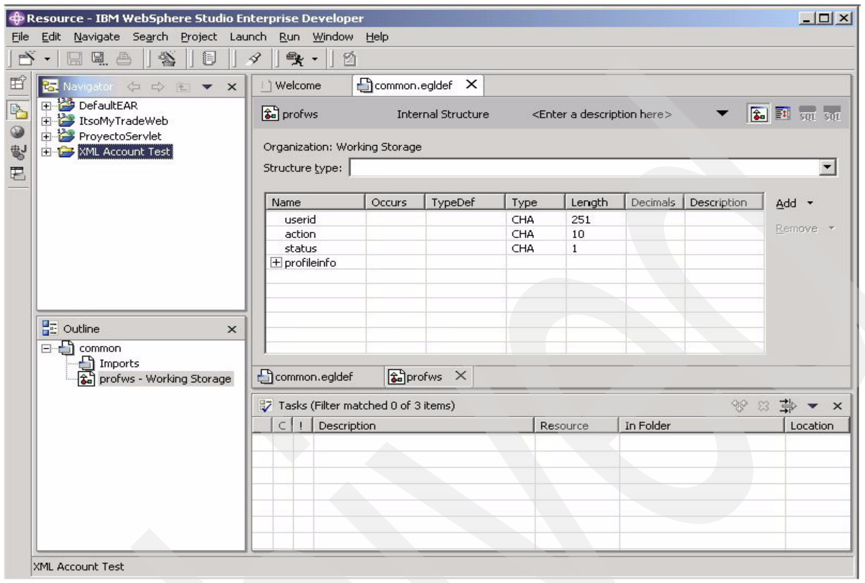


Figure 5-3 Continue creating a new project

4. Type the name of the project you want to create (we used `XML Account Test`), and click **Finish**. After a moment, you get an screen like the one shown in Figure 5-4.



Organization: Working Storage
Structure type:

Name	Occurs	TypeDef	Type	Length	Decimals	Description
userid			CHA	251		
action			CHA	10		
status			CHA	1		
⊞ profileinfo						

C	!	Description	Resource	In Folder	Location

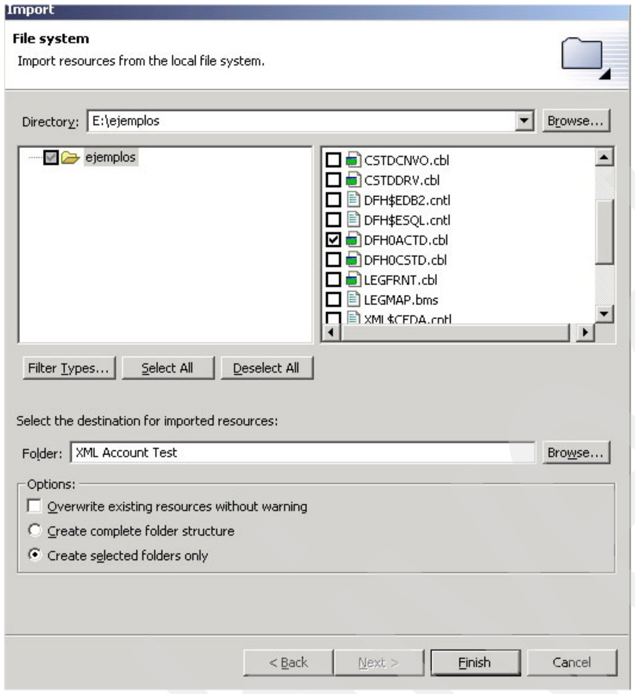
Figure 5-4 New created project

5. Select your project by right-clicking it.

   On the resulting pop-up menu, select **Import**. On the next screen select **File System** and click **Next**.

   On the next screen specify the Directory where you downloaded the file (in our case it was E:\ejemplos).

   The resulting screen is shown in Figure 5-5. The files in the directory you specified are listed in the right-hand pane. From this list, select the COBOL source program file you want to work with. In our example, this was DFH0ACTD.cbl.


Import

File system

Import resources from the local file system.

Directory: E:\ejemplos

- [ ] CSTDCNVO.cbl
- [ ] CSTDDRV.cbl
- [ ] DFH$EDB2.cntl
- [ ] DFH$ESQL.cntl
- [x] DFH0ACTD.cbl
- [ ] DFH0CSTD.cbl
- [ ] LEGFRNT.cbl
- [ ] LEGMAP.bms
- [ ] XML$CEDA.cntl

Select the destination for imported resources:

Folder: XML Account Test

Options:

- [ ] Overwrite existing resources without warning
- ( ) Create complete folder structure
- (•) Create selected folders only

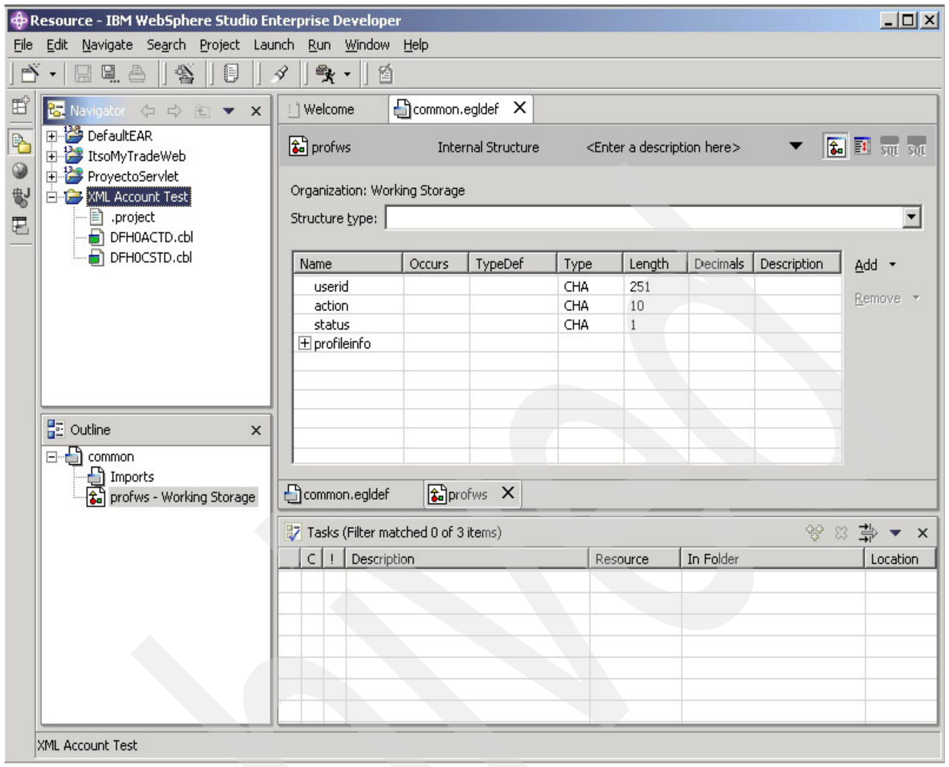
Figure 5-5 Importing a COBOL program

6. If you want to, you can select more than one source COBOL program (if your application has many programs).

   In our case, we have selected more than one COBOL program: DFH0ACTD.cbl for accounting details, and DFH0CSTD.cbl for customer details. This second program was downloaded with a file-transfer as well.

   To select more than one, simply click on all the files you want to work with. Once you have selected the files you want, click **Finish**. The resulting screen is shown in Figure 5-6.

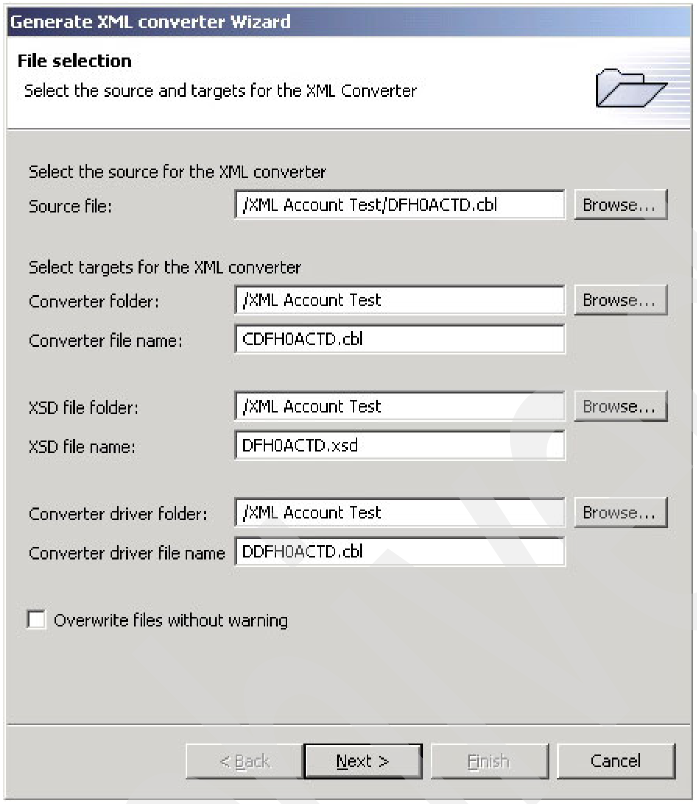
   You may need to expand the project listing by clicking the “**+**” symbol to the left of the project name in order to open it and see the contents (your COBOL files).



Name	Occurs	TypeDef	Type	Length	Decimals	Description
userid			CHA	251		
action			CHA	10		
status			CHA	1		
⊞ profileinfo						

Figure 5-6 Selecting COBOL programs

7. Now, it is time to generate the XML converters for your COBOL program. Select the COBOL file you want to convert (in our case it was DFH0ACT.cbl) and right-click it. On the resulting pop-up menu, select **Enable XML** → **Generate XML converter**. The Generate XML converter Wizard screen, shown in Figure 5-7, is returned.


Generate XML converter Wizard

File selection

Select the source and targets for the XML Converter

Select the source for the XML converter

Source file: /XML Account Test/DFH0ACTD.cbl

Select targets for the XML converter

Converter folder: /XML Account Test

Converter file name: CDFH0ACTD.cbl

XSD file folder: /XML Account Test

XSD file name: DFH0ACTD.xsd

Converter driver folder: /XML Account Test

Converter driver file name: DDFH0ACTD.cbl

☐ Overwrite files without warning

Figure 5-7 Generate XML Converter Wizard

The fields on this screen have the following meanings:

– **Source File**: The name of your COBOL source file. The path for this file is relative to the project in which you imported it. In our case, we imported the COBOL file into our Project name XML Account Test (not into a subdirectory of the project), so the path is /XML Account Test.

– **Converter folder**: The folder where you are going to store the automatically generated COBOL converter. In our case it is our Project name again, so we will have everything in the same directory.

– **Converter file name**: The name of the converter COBOL program generated by the WebSphere Studio Enterprise Developer. The default value is the same name as our original COBOL file, but starting with *C*. We changed this value to ACTDCNV.cbl, but this is optional. This converter is going to convert incoming and outgoing message to the appropriate format. In the same file (ACTDCNV.cbl), you are going to have two programs: ACTDCNVI for incoming XML messages, and ACTDCNVO for outgoing XML messages. Notice that the names for these programs are generated by default with the name of the converter file name plus ***I*** for incoming, and ***O*** for outgoing.

– **XSD file folder**: The WebSphere Studio Enterprise Developer is going to generate a schema according to the data definitions we have in our original COBOL program. This is useful to control the format of the XML messages going to and coming from our COBOL program. Here you specify the folder (again it is relative to the name of the project) where you want the schema file stored.

– **XSD file name**: The name for our resulting schema file.

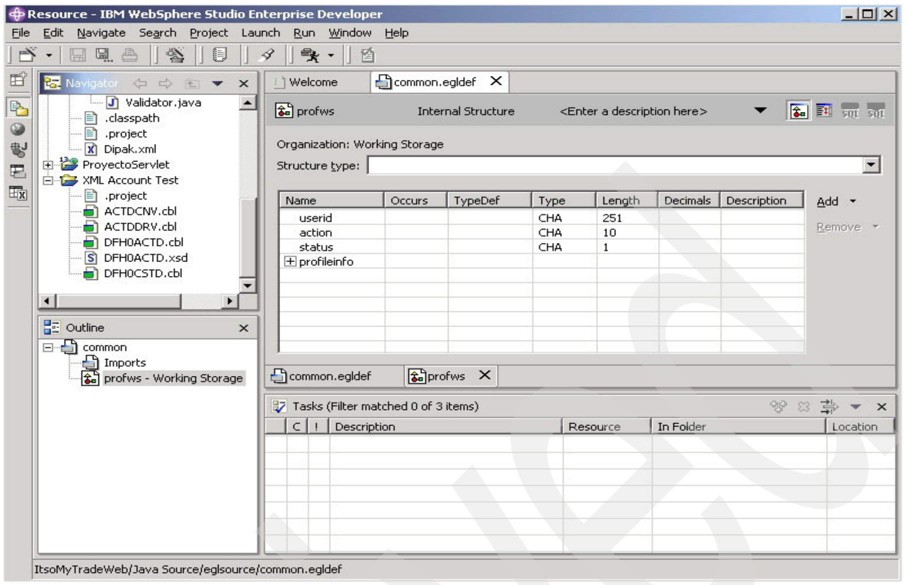
– **Converter driver folder**: The converter is a COBOL program that shows the invocation sequence for the inbound converter (*ACTDCNVI*), your existing COBOL program and the outbound converter (*ACTDCNVO*). Here you specify the folder where this driver is going to be generated (relative to the project).

– **Converter driver file name**: The name you are giving to this COBOL driver. Once this program file is generated, you are going to see that it manages possible exceptions, controls the first invocation to ACTDCNVI to receive the incoming XML message, then the invocation to your original COBOL program (legacy application), giving it received data in the XML message through the COMMAREA, and after the execution of your legacy program, it invokes to ACTDCNVO to convert the results of your COBOL program in a XML message. In our case, we called this converter driver ACTDDRV.

When you have filled in all the fields, click **Next**.

8. On the next screen the only thing you have to fill is the Program name, that is the PROGRAM-ID that appears in the IDENTIFICATION DIVISION. For example, if you enter ACTDCNV as the program name, the wizard will generate ACTDCNVI for the inbound converter program name, ACTDCNVO for the outbound converter, and ACTDCNVD for the converter driver.

9. Figure 5-8 shows the resulting screen with all of the files listed.



Organization: Working Storage

Structure type:

Name	Occurs	TypeDef	Type	Length	Decimals	Description
userid			CHA	251		
action			CHA	10		
status			CHA	1		
+ profileinfo						

Figure 5-8

10. You have to edit the converter driver file (ACTDDRV) and look for a comment in the middle stating: Execute Legacy Application. This is where you code the invocation to your original COBOL program. The general structure of the driver is shown in Example 5-7.

Example 5-7 Locate "Execute Legacy Application"

```
(Invocation to incoming XML converter program)
          ..................
          ..................
   * + -------------------------- +
   * | Execute Legacy Application |
   * + -------------------------- +
   * . EXEC CICS LINK
   * .   PROGRAM('LEGACY')
   * .   COMMAREA(DFHCOMMAREA)
   * .   ...
   * . END-EXEC ...OR
   * .
   * . call 'LEGACY' using DFHCOMMAREA
          ..................
          ..................
(Invocation to incoming XML converter program)
```

11. At this point, you have the following files:

 – DFH0ACTD.cbl: Original COBOL program.
 – ACTDCNV.cbl: File that contains two programs (and other routines), ACTDCNVI to process incoming XML messages, and ACTDCNVO for outgoing XML messages.
 – ACTDDRV.cbl: The COBOL program to control the invocation flow sequence among input, original COBOL program, and output.
 – DFH0ACTD.xsd: A schema of the XML document that can be used in the servlet receiving incoming XML document.

The first invocation has to be to the driver (ACTDDRV.cbl) that is in charge to receive the XML message, to invoke to the incoming converter (ACTDCNVI), then to invoke the legacy program, and finally to invoke the output converter (ACTDCNVO). Of course, you have to work (by hand) with this driver. The first thing you have to decide is how to invoke the driver and from where. You will probably want to do this from a servlet or Java component.

At this point, it is very useful to use components generated by Struts and EGL. You can generate an EGL Java Wrapper to access an EGL CICS COBOL program, so an Action Struts class can access this EGL CICS COBOL program, passing it the initial XML message. This EGL CICS COBOL program can invoke, or have inside, the driver we generated.

The schema document (DFH0ACTD.xsd) can be used in the Struts Action Servlet to validate the incoming XML document.

The new process that we have just generated is now enabled to process XML input. The flow can be summarized as follows:

- An Action Servlet generated with Struts receives an incoming request from the client (XML document).
- Action Servlet validates the XML document with DFH0ACTD.xsd schema.
- Action Servlet invokes a Command Bean (Java wrapper generated with WebSphere Studio Enterprise Developer), passing it the XML document.
- Action bean (Command Bean) invokes COBOL driver (ACTDDRV) through J2EE Connector, passing it XML data in the COMMAREA.
- The COBOL driver invokes to the input converter (ACTDCNVI) that converts XML into binary data.
- Then, the COBOL driver invokes to the Legacy COBOL program, passing it binary data through the COMMAREA.
- The COBOL driver invokes to the output converter (ACTDCNVO), which converts the result to an XML output.

- Another Action Bean receives the response from the COMMAREA in XML format.

6

WebSphere Application Server on z/OS and OS/390

In this chapter we describe the WebSphere Application Server 4.01 for z/OS and OS/390 and discuss its relationship with J2EE applications and XML. We also give a brief overview of Web services for DB2 UDB and some considerations regarding the J2EE application runtime environment.

6.1 IBM WebSphere Application Server

In Chapter 7 we discuss in great detail the relationship between XML and Web services applications. By way of introduction, though, we first want to reflect on some issues regarding the Web-serving environment to enable Web applications for use inside WebSphere Application Server, as well as considerations related to both the development and run-time environments.

WebSphere Application Server for z/OS and OS/390 provides a run-time environment for Java 2 Enterprise Edition (J2EE) applications.

WebSphere for z/OS supports both Enterprise JavaBeans and Web components. These two types of J2EE application components can use:

- The application programming interfaces (APIs) and services that the Java 2 Standard Edition (J2SE) Software Development Kit (SDK) V1.3 provides
- Enterprise services such as Java Database Connectivity (JDBC), Java Naming and Directory Interface (JNDI), and the Java Transaction Service (JTS) and API (JTA)

The J2EE specifications dictate which APIs and services each type of application component may use, and the environment in which they must run. Enterprise beans run in the EJB container, and Web applications run in a Web container. These two containers in the WebSphere for z/OS J2EE server conform to the J2EE specifications for run-time environments. Figure 6-1 illustrates how the pieces are related. (Other configurations are possible, and are discussed in later sections.)

In this configuration, the address spaces (AS) related to a J2EE instance are the control region and one or more server regions. The number of server regions depends on the service level defined to WLM so that WLM can start server regions automatically to achieve the service goal.

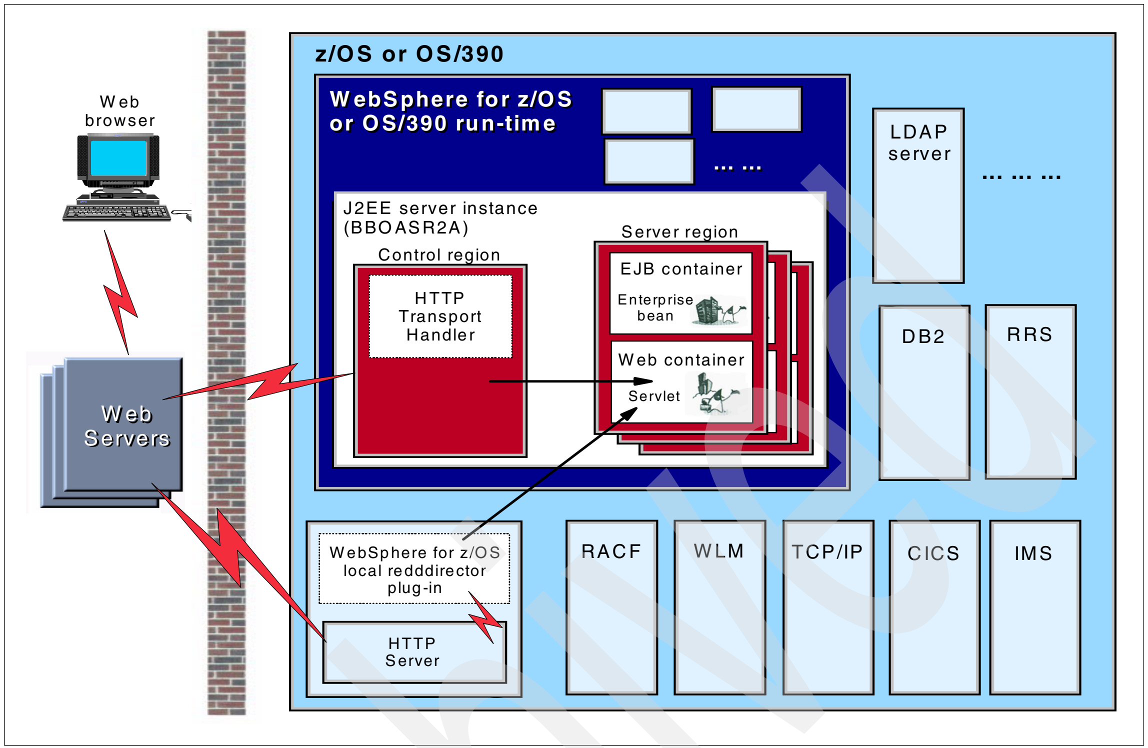


Figure 6-1 WebSphere Application Server in z/OS and OS/390

6.2 The WebSphere for z/OS environment

Figure 6-1 shows the pieces that comprise the WebSphere for z/OS run-time environment for Web applications, including the different address spaces that allow different configurations. There are two possible scenarios depending on how the HTTP listener is configured. The result is that clients can access Web applications in the following ways:

1. Using an HTTP Transport Handler
   Each J2EE server can be configured to directly receive HTTP requests for servlets, rather than relying on an HTTP Server to route the requests to the Web container.
2. Using an HTTP Server
   HTTP requests are routed to Web applications running in a Web container or residing in V3.5 runtime within the HTTP Server address space.

WebSphere for z/OS uses specific directories in the hierarchical file system (HFS) for configuration data. These files, shown in Figure 6-2, are the following:

- The current.env file contains the environment variables defining the HTTP Transport Handler server environment variables and environment files.
- The webcontainer.conf file contains definitions that enable the J2EE server's Web container to logically separate one or more Web applications from others installed in the same container, using constructs called *virtual hosts* and *context roots*.
- The jvm.properties file, which is associated with a specific J2EE server instance, contains a pointer to the webcontainer.conf file.

6.2.1 Enterprise applications (J2EE applications)

An enterprise application is a grouping of one or more Web, EJB, or application client modules. Besides being an efficient grouping mechanism, enterprise applications make it much easier to deploy and maintain code at the level of a complete application instead of as individual pieces. Enterprise applications can also override settings within the contained modules' deployment descriptors (XML documents) to combine or deploy them in a more useful way.

WebSphere for z/OS provides a J2EE server instance for J2EE application components: Enterprise beans and Web applications. J2EE servers contain at least one EJB container and one Web container. The EJB container manages Enterprise beans, while the Web container manages Web applications.

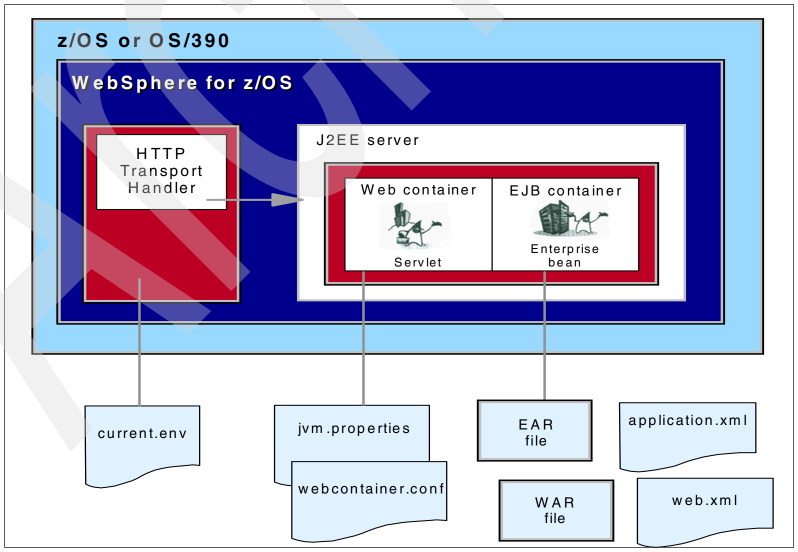


Figure 6-2 J2EE components

Web applications

A Web module contains HTML, images, JSPs, Java classes and servlets, and all other resources required to create a Web application. Like the other modules, Web modules contain a deployment descriptor. In Web modules, the deployment descriptor, *web.xml*, has servlet initialization and mapping information, as well as other settings for running the Web module within an application server.

EJBs

EJB modules contain EJB beans, their server-specific deployment code, a deployment descriptor (an XML file), and optionally, helper classes. They contain the business logic of your application, and are typically called by Web application clients, or other EJB modules.

An enterprise application may contain jar files to be used by the contained modules. This allows sharing of code at the application level, and is one place to put utility jar files that are used by multiple Web or EJB modules. Placing these jar files in the enterprise application instead of on a global classpath means they do not require special publishing and setup when moving to a new server.

.xml files related to the Web application

A Web application needs to be packaged in a Web Archive (WAR) file, and then this WAR file is packaged in an Enterprise Archive (EAR) file, perhaps along with the Enterprise beans (and their jar files) that your Web application uses. To carry out both tasks there are several tools, which were discussed in Chapter 4, “Services development environment” on page 81. Both war and ear files contain XML files that describe each component in the application. These XML files enable the WebSphere for z/OS J2EE server to provide the correct execution environment for the application components. The content of the XML files is something that you supply when you package the application, but the files themselves are generated for you.

The files related with the application are also shown in Figure 6-2.

The web.xml file is the “deployment descriptor” for the Web application. Normally this file is created by the tool you use to create the WAR. Since the WebSphere Application Server needs to understand this information, it is provided with an XML parser. The parser used in WebSphere 4.0.1 corresponds to the Apache Software Foundation's Xerces Java Parser 1.0.3. One of the CLASSPATH variables we discuss in 6.2.2, “JAR files and classes” on page 141 is the path to xerces.jar, which contains this Apache implementation of SAX APIs.

This is not the latest level of xerces and you may consider it to be too low for your applications to use. *Applications are not obliged to use this parser*. The xerces provided with WebSphere Application Server is intended for its internal use; the recommendation is that any application that depends on XML should package its own parser as part of the application, or use the APP_EXT_DIR environment variable pointing to shared jar files among Web applications. In 6.4, “Development-time and run-time considerations” on page 145 we discuss this issue in more detail.

Since there may be multiple levels of xerces along a “general” classpath, we have developed a simple servlet you can include in your applications to determine which level of xerces an application is using.

This servlet may be useful to determine the actual version of XML4J in use by your application running under WebSphere Application Server. Our servlet, shown in Example 6-1, is very simple. It invokes dynamically a class called org.apache.xerces.impl.Version. This class is included in xercesImpl.jar, the jar file containing classes for xerces parser in the XML Toolkit V1R4. This Version class writes a message in the log with the version number.

Example 6-1 Version servlet

```
package david;

import java.lang.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import org.apache.xerces.*;
import org.apache.xerces.impl.Version;

public class VersionInfo extends GenericServlet {
   public void init() throws ServletException {
      ServletContext context = getServletContext();
      log("CLASSPATH IN USE: " +
context.getAttribute("com.ibm.websphere.servlet.application.classpath"));
   }

   public void service(ServletRequest req, ServletResponse res)
                     throws ServletException, IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();

      ServletContext context = getServletContext();
      out.println("<H4>Server Name: " + req.getServerName() +"</H4>");
      out.println("<H4>Port in Use: " + req.getServerPort() + "</H4>");
      out.println("<H4>Server Version: " + context.getServerInfo() + "</H4>");
      out.println("<H4>Current Classpath:</H4><H8>" +
context.getAttribute("com.ibm.websphere.servlet.application.classpath") +
"</H8>");

      try {
         Class c = Class.forName("org.apache.xerces.impl.Version");
         org.apache.xerces.impl.Version o = (org.apache.xerces.impl.Version)
c.newInstance();
            System.out.println("***** The application is using the following
version of the XML parser: *****");
            String [] arrayForMain = new String[1];
            o.main(arrayForMain);
      } catch(ClassNotFoundException e) {
         out.println("ERROR1: " + e.getMessage());
         // Add code here to check for the org.apache.xerces.framework.Version
class.
         // Since you were unable to find impl.Version, you are actually
accessing
         // an older version of the parser
      } catch (IllegalAccessException e) {
         out.println("ERROR2: " + e.getMessage());
      } catch (InstantiationException e) {
         out.println("ERROR3: " + e.getMessage());
      }
   }

   private String getServerInfoName(String serverInfo) {
      int slash = serverInfo.indexOf('/');
      if (slash == -1) return serverInfo;
      else return serverInfo.substring(0, slash);
   }
}
```

You can include this servlet in your application. Each time you invoke it, the servlet responds with the environment information. The response is written to the WebSphere Application Server log, for example:

```
XML4J 4.0.2
```

6.2.2 JAR files and classes

There are several places where you can put the classes needed for the application to run. The aim is that as the application is running, all of the invoked classes can be found and loaded.

- A Web module has two special folders: WEB-INF/classes and WEB-INF/lib. The classes folder may contain “loose” Java classes (classes that are not inside a jar file), and can be used for servlet support or utility classes within the scope of the Web application. The lib folder may contain jar files that are also used by the Web application.

- An enterprise application may contain jar files to be used by the contained modules. This allows sharing of code at the application level, and is one place to put utility jar files that are used by multiple Web or EJB modules.

6.3 Application deployment

In this section we describe the process of deploying a Web service application in WebSphere Application Server V4.01. The flow is illustrated in Figure 6-3.

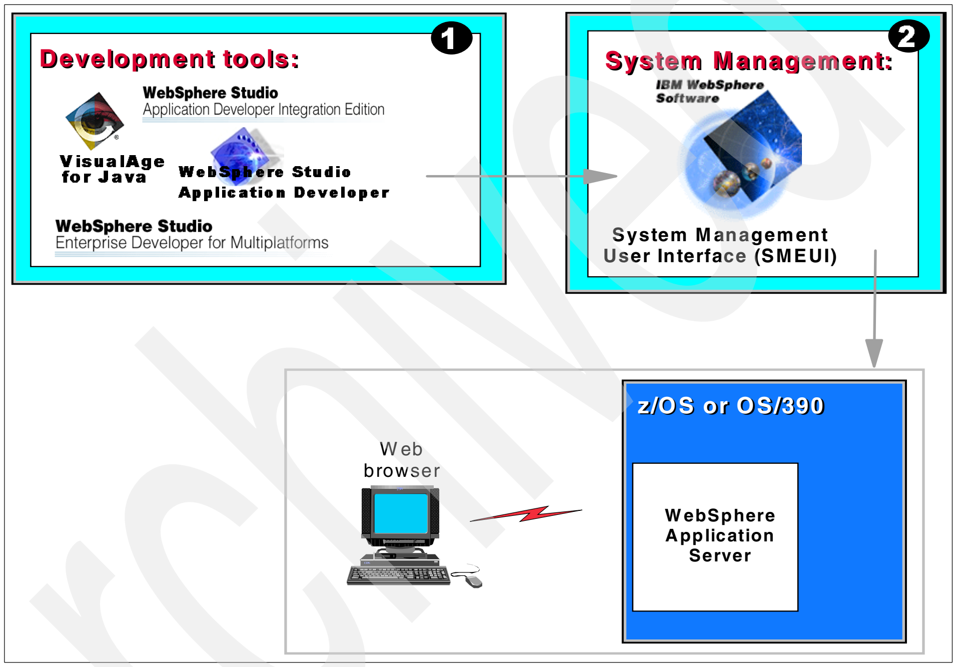

Development tools:
1
WebSphere Studio Application Developer Integration Edition
VisualAge for Java
WebSphere Studio Application Developer
WebSphere Studio Enterprise Developer for Multiplatforms
System Management:
2
IBM WebSphere Software
System Management User Interface (SMEUI)
z/OS or OS/390
Web browser
WebSphere Application Server

Figure 6-3 Deployment flow

Step 1 is a development task. The goal is to develop an application and generate an ear file using any of the development tools. All files belonging to a single application can be bundled into a single archive file and deployed to a runtime server by placing the archive file into a specific directory on the target server.

Import one or more ear files that contain applications you want to work on, or create a new application and import one or more Enterprise Java Bean .jar or .war files into the new application.

Step 2 is the application deployment, which is illustrated in Figure 6-4. WebSphere for z/OS comes with Windows-based tools for application deployment: DDT/390fy, AAT and SMEUI.

You import an ear file from your development workstation using the Systems Management User Interface (SMEUI) and, optionally, the Application Assembly Tool (AAT).

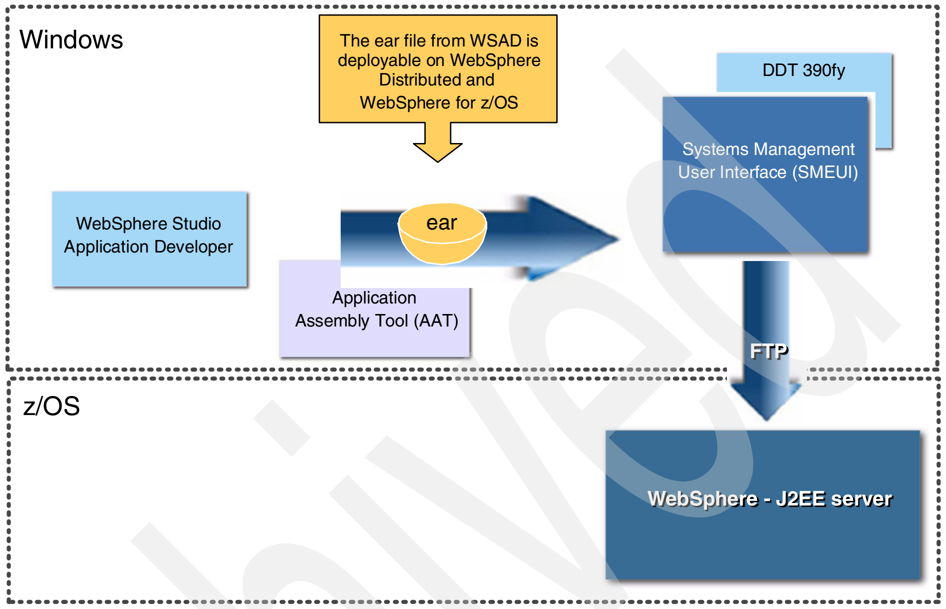

Windows
The ear file from WSAD is deployable on WebSphere Distributed and WebSphere for z/OS
DDT 390fy
Systems Management User Interface (SMEUI)
WebSphere Studio Application Developer
ear
Application Assembly Tool (AAT)
FTP
z/OS
WebSphere - J2EE server

Figure 6-4 Deployment

With WebSphere 4.0.1 service level W401400, ear files generated using WebSphere Studio Application Developer can be deployed using the SMEUI short path. SMEUI then automatically invokes DDT/390fy and you are not required to use the AAT. However, you still need to use the AAT when using some WebSphere application extensions.

The SMEUI is a graphical administrative application. In addition to WebSphere systems management functions, it is used to deploy Web and EJB modules from an ear file into the WebSphere 4.01 environment on z/OS or OS/390. You must use the SMEUI installation files that come with your version of WebSphere for z/OS.

Note: The SMEUI tool comes with WebSphere for z/OS. The tool is located in the WebSphere directory: /usr/lpp/WebSphere/bin/bboinst.exe

We used SMEUI version 4.0.020.

For an e-business application, the SMEUI does the following:

- Sets up the security environment to host the enterprise bean according to its deployment descriptor.
- Registers the enterprise bean, its environment properties, resource references, and so forth in the JNDI name space.

At the end of a successful deployment process, SMEUI automatically copies the application files to the z/OS UNIX file system using FTP.

One important thing to point out is the *context root.* Context roots are used to associate received URLs with an application in the host. WebSphere knows a URL is intended for a webapp based on a match with the context root. Context roots are defined during development with WebSphere Studio Application Developer, or during application assembly using AAT. The application is bound to a virtual host by making a match between the context root value and the value found in the webcontainer.conf file associated with the WebSphere server on z/OS.

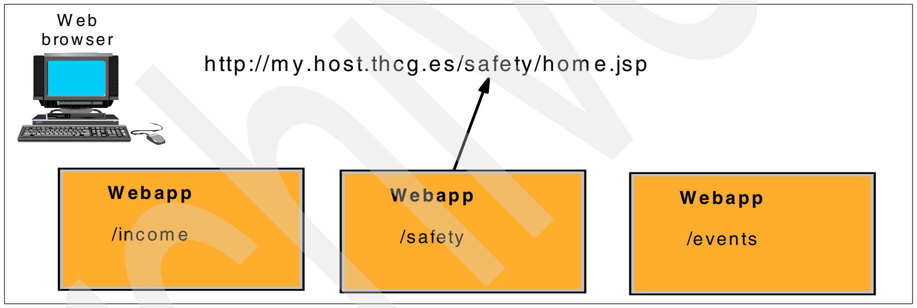


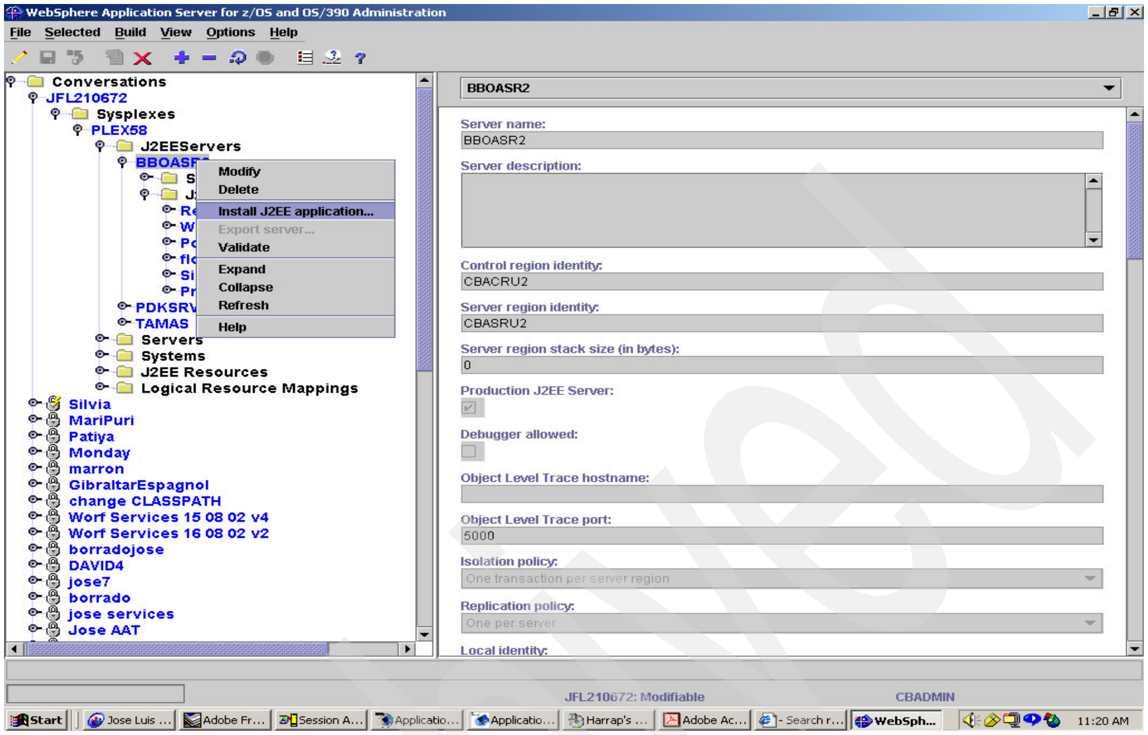
Figure 6-5 Context root

For further information, see "Websphere V4.0 and V4.0.1 for z/OS - Configuring Web Applications." This document can be obtained from the following URL:

`http://www.ibm.com/support/techdocs`

With the ear file generated, you need to start the SMEUI and install the J2EE application on the z/OS server, assuming a Web container has already been configured.

Start the SMEUI and create a new conversation. Select the J2EE server where you want to add the new application. Figure 6-6 shows a conversation.



Server name: BBOASR2

Server description:

Control region identity: CBACRU2

Server region identity: CBASRU2

Server region stack size (in bytes): 0

Production J2EE Server: ✓

Debugger allowed:

Object Level Trace hostname:

Object Level Trace port: 5000

Isolation policy: One transaction per server region

Replication policy: One per server

Local identity:

Figure 6-6 SMEUI

Then, select the assembled ear file for installation into the server and follow the SMEUI instructions to:

- VALIDATE
- COMMIT
- ACTIVATE the conversation.

SMEUI transfers the file onto the WebSphere z/OS server using FTP. After activating this conversation you are ready to try the URL of the J2EE application.

For further information about SMEUI and J2EE refer to the following Web site:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zswpdf/was401.html`

6.4 Development-time and run-time considerations

At compile time, a Java compiler needs to know about every class or jar file that your code refers to, so that it can safely compile and type-check against these classes. The Java compiler collects all the information about the needed classes.

At run time, WebSphere Application server for z/OS uses a classloader mechanism to find and load classes. Your application may compile correctly, but still have ClassNotFoundExceptions at run time.

You can successfully use WebSphere for z/OS classloader defaults for most applications. If you encounter classloader errors, however, you might need to alter classloader operation or repackage application components. To do so, you need to understand how the classloaders interact, and how their operation can be altered.

Refer to *WebSphere Application Server V4.0.1 for z/OS and OS/390 Assembling Java 2 Platform, Enterprise Edition (J2EE) Applications,* SA22-7836 for a complete description of WebSphere z/OS classloaders operation. In this redbook, we limit our discussion to elements relevant to application packaging of XML parsers.

6.4.1 Overview of WebSphere classloader operations

A *classloader* is a class that performs the function of loading a named class or interface. When an application client requests an Enterprise bean, for example, the WebSphere for z/OS run-time creates a classloader to find and load the classes from the appropriate jar module.

Each of the WebSphere for z/OS classloaders share the following common features, but each has different values that define the classloader and its behavior:

- Context classpath

  Each classloader has an associated classpath that is defined for the specific type of classloader, and for the module (jar or war) associated with that classloader. The classpath defines the part of the HFS file system that the classloader searches to locate a requested class.

- Delegation mode

  Each classloader has an associated parent classloader and a delegation mode. The delegation mode determines when the classloader will delegate a load request to its parent; the classloader may delegate to its parent either before or after searching its own classpath.

Visibility mode

WebSphere for z/OS supports the following *classloader modes*, also known as *visibility modes*:

- Application mode, which is the default classloader mode
- Compatibility mode

- Server mode
- Module mode
- J2EE Application mode

Although IBM recommends using the default classloader mode (application mode), you may alter the mode if the needs of your application warrant a change in classloader behavior. Base your choice of classloader mode on the information presented in the following table.

Table 6-1 WebSphere for z/OS classloader guidelines

Use classloader	When your application
Application (default)	Is composed of a number of modules packaged in only one ear file per application, and those modules within an application need to work together. In other words, each application is complete and independent of other applications. Use SMEUI to set this classloader through environment variable `com.ibm.ws390.server.classloadermode=2`
Server	Is composed of a number of modules packaged in more than one ear file, and those modules need to work together. Use SMEUI to set this classsloader through environment variable `com.ibm.ws390.server.classloadermode=3`
Compatibility	Needs to be moved from WebSphere for z/OS Standard Edition V3.5 to WebSphere for z/OS V4.0.1 Use SMEUI to set this classloader through environment variable `com.ibm.ws390.server.classloadermode=1`
Module	Is composed of multiple modules, each of which may have a distinct version of some shared code segment, and thus must be kept isolated. Use SMEUI to set this classloader through environment variable `com.ibm.ws390.server.classloadermode=0`
J2EE Application	Requires classloader behavior that is compliant with the J2EE 1.3 specification. This mode is set through the JVM property `com.ibm.ws.classloader.J2EEApplicationMode=true|false`

Notes® on class packaging for specific classloader modes:

- For all classloader modes, parent classloaders cannot see the classes handled by child classloaders.
- If your application modules use a MANIFEST classpath, that classpath overrides the following default classloader operation:
  - Module mode: All module classloaders in an application are visible to each other, and all utility jar files in the application are visible to all module classloaders in the application.

  - Compatibility mode: Utility jar files are visible to all module classloaders in all active applications.
- Package EJB JAR and WAR modules that make up an application in the same ear module.
- For module mode: If a WAR module needs to access an EJB JAR module, use a MANIFEST classpath entry in the WAR module to document the dependency.
- Common classes used by EJB JAR and WAR modules should be put in a JAR that is added to the ear file, and referenced in the MANIFEST classpaths of the modules.

  Alternatively, you can specify shared classes on the APP_EXT_DIR environment variable for the J2EE server. This is the technique we used for all the WebSphere tests documented in this redbook.
- Class library jar files to be used only by WAR modules should be added to the WEB-INF/lib directory of the WAR module.

The variable `com.ibm.ws390.server.classloadermode` is set for the server through the SMEUI by specifying its value on the properties form for a J2EE server. It is also possible to override the value for a given server instance by setting `com.ibm.ws390.server.classloadermode` in its JVM properties file.

Each of the classloaders identified is defined as a child of the classloader above it. This conforms to a chain that in general is similar to the one shown in Figure 6-7, though details may differ depending on the visibility mode.

WebSphere for z/OS creates one application classloader for each application installed in the J2EE server. The classpath for a given classloader includes the module paths for all war or jar files within the application.

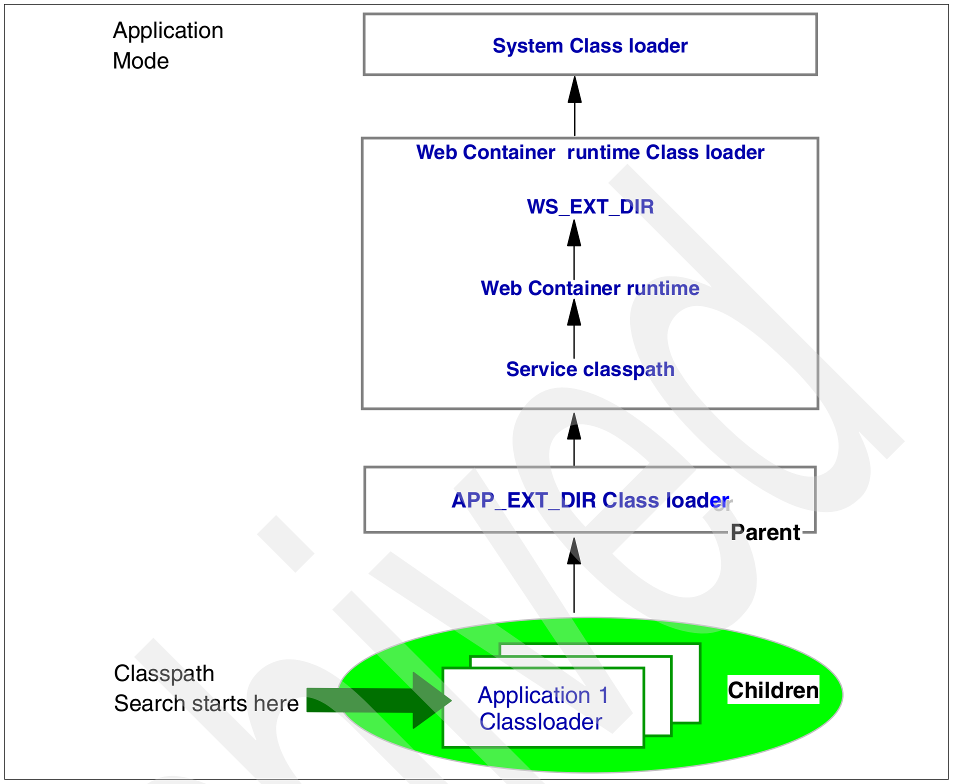


Figure 6-7 Classloader: Application mode

The Application extensions classloader is the parent of all application classloaders. The classpath for the Application extensions classloader consists of the path specified on the APP_EXT_DIR environment variable for the J2EE server. This variable is intended for classes that can be shared by all applications installed in the J2EE server. This makes it the preferred choice to place XML parser classes when multiple applications require access to the parser classes. As stated, this is the solution we used for all our tests running in WebSphere.

Search can only go up the tree; it cannot go down. When a classloader is requested to find a class, it does not search down. If the class is not found going up the tree a ClassNotFoundException occurs.

With delegation defaults in effect, the search for a particular class begins with the application classloader associated with the application containing that class. If the class is not found in that application’s modules, the application classloader passes the search request to its parent. The search request progresses up the tree until the class is found or a ClassNotFoundException occurs.

The delegation mode for the application classloaders may be altered, as explained in the next section, “Delegation mode.”

When starting the server, the BBOJ0018I message indicating the selected visibility mode appears in the Server Region Address Space log, as shown in Figure 6-8. This example indicates application mode for our BBOASR2 server.

```
BBOJ0018I WebSphere for z/OS server "BBOASR2" Visibility mode is
"Application".
```

Figure 6-8 Visibility mode message

To change from one visibility mode to another you need to change the `com.ibm.ws390.server.classloadermode` variable to the desired value using SMEUI, then stop and restart the J2EE server.

Delegation mode

Each classloader has an associated *parent* ClassLoader. The delegation mode specifies whether the ClassLoader will delegate a search request to its parent *before* or *after* searching its own ClassPath.

- When delegation is *true*, the classloader first delegates the request to its parent classloader. If none of the parent classloaders can find the class, the original classloader attempts to load the class.
- When delegation is *false*, the classloader first attempts to load the class itself before going to its parent.

Default delegation values are assigned depending on the visibility mode you are using. You can change the defaults by setting the following variables in jvm.properties.

- Server mode, when `com.ibm.ws390.server.classloadermode=3`

  All module classloaders of all applications in the same server instance are in the same group and all applications have visibility from each other. Default delegation values are:

  ```
  com.ibm.ws.classloader.warDelegationMode = false
  com.ibm.ws.classloader.ejbDelegationMode = true
  ```

- Application mode, when `com.ibm.ws390.server.classloadermode=2`

  Applications have no visibility from each other and they do not share jar files. Each application is isolated. Default delegation values are:

  ```
  com.ibm.ws.classloader.warDelegationMode = false
  com.ibm.ws.classloader.ejbDelegationMode = false
  ```

- Module mode, when `com.ibm.ws390.server.classloadermode=0`

Each module in the same or different applications is independent and module classloaders have no visibility from each other even in the same application. Default delegation values are:

```
com.ibm.ws.classloader.warDelegationMode = false
com.ibm.ws.classloader.ejbDelegationMode = true
```

- Compatibility mode, when `com.ibm.ws390.server.classloadermode=1`

  All EJBJarClassloaders of all applications in the same server instance are in the same group, so applications have EJB jar files visibility to each other. But WAR modules are independent, so all the WARClassLoaders have no visibility to each other. Default delegation values are:

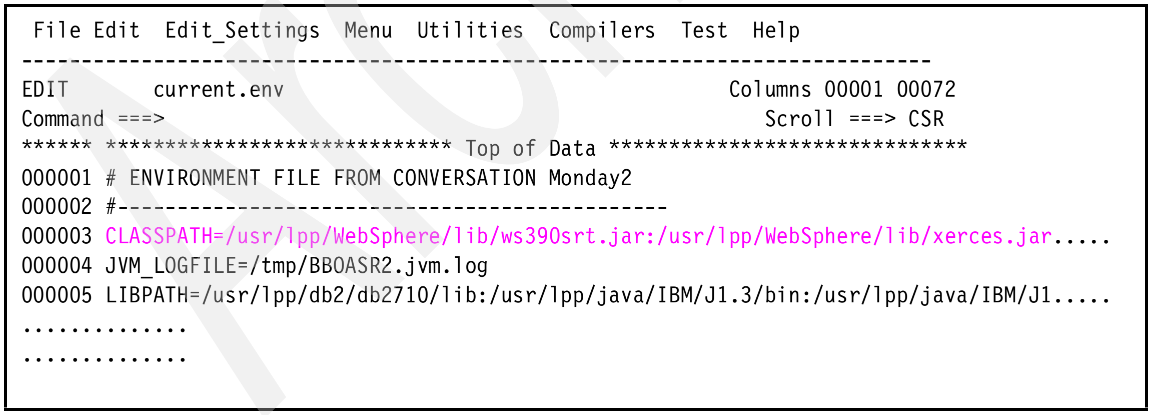
  ```
  com.ibm.ws.classloader.warDelegationMode = false
  com.ibm.ws.classloader.ejbDelegationMode = true
  ```

6.4.2 Classpath

ClassLoaders are in charge of loading required classes from the specific classpath to which they have been assigned. ClassLoaders go to a classpath to look for that required class.

As part of the installation and customization of WebSphere Application Server you define a classpath environment variable CLASSPATH using the SMEUI. This variable contains the definition of the paths to the server jar files where the System ClassLoader is going to look for general classes.

A typical example is shown in Figure 6-9.

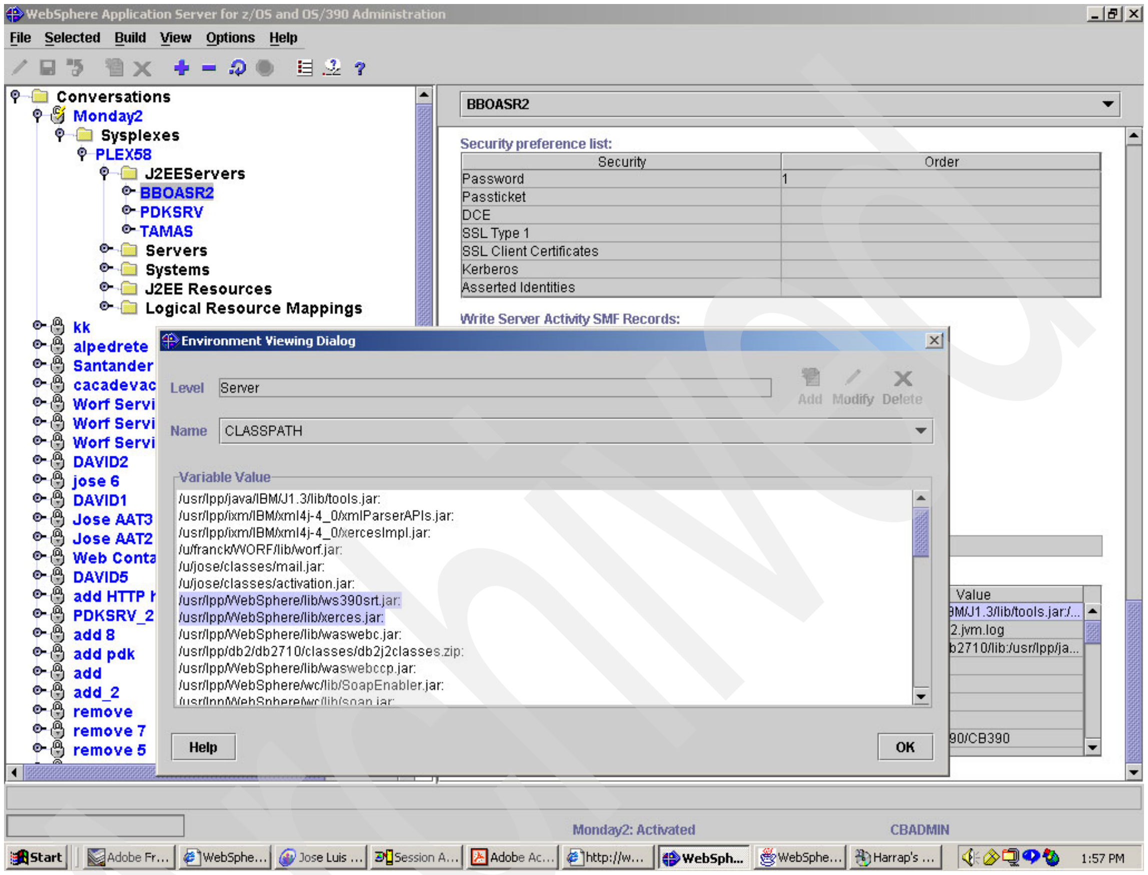


```
 File Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-----------------------------------------------------------------------------
EDIT       current.env                                         Columns 00001 00072
Command ===>                                                      Scroll ===> CSR
****** **************************** Top of Data *****************************
000001 # ENVIRONMENT FILE FROM CONVERSATION Monday2
000002 #--------------------------------------------------
000003 CLASSPATH=/usr/lpp/WebSphere/lib/ws390srt.jar:/usr/lpp/WebSphere/lib/xerces.jar.....
000004 JVM_LOGFILE=/tmp/BBOASR2.jvm.log
000005 LIBPATH=/usr/lpp/db2/db2710/lib:/usr/lpp/java/IBM/J1.3/bin:/usr/lpp/java/IBM/J1.....
..............
..............
```

Figure 6-9 CLASSPATH in current.env

Once the Control Region is started, these files are the server classpath for the system classloader.

Modifications to this variable are made through a conversation with SMEUI. Figure 6-10 displays a conversation with SMEUI to browse the CLASSPATH variable for the system classloader.



Security preference list:

Security	Order
Password	1
Passticket	
DCE	
SSL Type 1	
SSL Client Certificates	
Kerberos	
Asserted Identities	

Write Server Activity SMF Records:

Level: Server

Name: CLASSPATH

Variable Value

```
/usr/lpp/java/IBM/J1.3/lib/tools.jar:
/usr/lpp/ixm/IBM/xml4j-4_0/xmlParserAPIs.jar:
/usr/lpp/ixm/IBM/xml4j-4_0/xercesImpl.jar:
/u/franck/WORF/lib/worf.jar:
/u/jose/classes/mail.jar:
/u/jose/classes/activation.jar:
/usr/lpp/WebSphere/lib/ws390srt.jar:
/usr/lpp/WebSphere/lib/xerces.jar:
/usr/lpp/WebSphere/lib/waswebc.jar:
/usr/lpp/db2/db2710/classes/db2j2classes.zip:
/usr/lpp/WebSphere/lib/waswebccp.jar:
/usr/lpp/WebSphere/wc/lib/SoapEnabler.jar:
/usr/lpp/WebSphere/wc/lib/soap.jar:
```

Value
9M/J1.3/lib/tools.jar:/...
2.jvm.log
b2710/lib:/usr/lpp/ja...
90/CB390

Figure 6-10 Visualize CLASSPATH using SMEUI

6.5 Application considerations

We have discussed several aspects of J2EE application deployment, and the use of WebSphere classloaders. These functions lead to more choices on how you assemble your applications.

One of the things that confuses new developers is the manner in which each of the classloaders is handled in WebSphere Application Server Version 4.01. They are implemented in a hierarchy that defines a search order as follows:

- Except for the delegation mode nuance, the application classloaders (WARClassLoaders and EJBClassLoaders) are searched first
- APP_EXT_DIR
- WebSphere Classloader
- System Classloader

The key thing to remember when assembling your application is that requests only go *up* the search tree. That means that a request to load a class located in the application classloaders that originates from the System classloader will result in a ClassNotFoundException error. Conversely, a request to load a class located in the system classloader that originates from the APP_EXT_DIR classloader will be successfully passed on up the chain.

The end result is that if a developer packages a jar file in a WebSphere system classpath, it cannot depend on classes within modules down the tree *because the classloader can't see them*. During our tests, we triggered a number of ClassNotFoundException errors.

To further complicate things, WebSphere supports different *visibility* modes, and different delegation mode choices may modify the search sequence, as explained in the previous sections. Since there are multiple possible combinations, the difficulty is to decide which one is the best to assemble your applications.

Assembling J2EE applications

Although WebSphere contains XML parsers as part of its code, they are for internal use of the WebSphere server itself. It is recommended that you provide your own version of the parser classes, xercesImpl.jar and xmlParserAPIs.jar, when using XML in your applications. The same recommendation applies to applications using SOAP and requiring soap.jar.

If you are deploying an XML-based or a Web services application, you should package the required jar files either in WEB-INF/lib together with the rest of the jar files, or in a location recognized by the WebSphere classloaders.

During our tests, we used Application visibility mode with the use of the Application Extensions directory (APP_EXT_DIR) and found that it provides a very convenient and efficient way to package the applications. See “The Web Services WORF sample” on page 202.

The APP_EXT_DIR alternative allows consistency across applications since they all refer to the same set of jar files, something which also simplifies maintenance.

However, placing the jar files in the APP_EXT_DIR classpath creates dependencies across applications. It also requires an additional setup step to create the application extensions directory when you move applications from one run-time environment to another.

If easy portability across multiple run-time servers is the main concern, you may choose to package all the war and jar files the application needs into one single ear file. The parser and SOAP jar files being included reside in the WEB-INF/lib directory. While this is one way to make the application self-contained and more easily portable, there will be an added cost for maintenance because of the jar files duplicated across multiple applications.

SDK 1.4

XML and Java are rapidly evolving technologies. As XML is gathering more and more importance, development and deployment patterns also change.

IBM has announced a new version of the IBM SDK for z/OS, Java 2 Technology Edition V1.4. With this SDK level you can take advantage of enhancements, such as a new Just-in-Time (JIT) compiler, the inclusion of XML parser, security APIs, and persistent reusable technology.

The Java API for XML (JAXP) processing has been added to the Java 2 Platform. It provides basic support for processing XML documents through a standardized set of Java Platform APIs. The Java API for XML processing includes the basic facilities for working with XML documents through the following standardized set of Java Platform APIs:

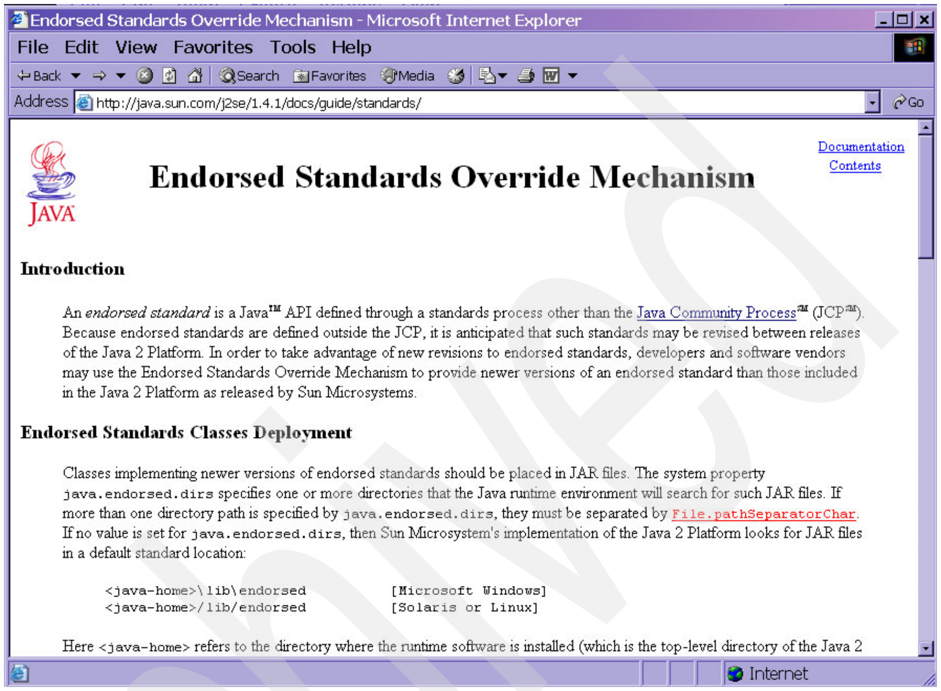
- Document Object Model (DOM) Level 2
- Simple API For XML Parsing (SAX) 2.0
- XSLT 1.0
- Pluggability Layer

The prerequisite to use JDK 1.4 is z/OS V1.4 or later.

It is possible with JDK 1.4 to override the XML parser provided with a given implementation. In order to take advantage of new revisions to endorsed standards and override the JAXP implementation in J2SE SDK 1.4 and above, the *Endorsed Standards Override Mechanism* allows developers to provide versions of an endorsed standard newer than those included in the Java 2 Platform as released by Sun Microsystems.

More detailed information on the *Endorsed Standards Override Mechanism* is provided at the following URL:

http://java.sun.com/j2se/1.4.1/docs/guide/standards/


Endorsed Standards Override Mechanism - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://java.sun.com/j2se/1.4.1/docs/guide/standards/
Go
Documentation Contents
JAVA
Internet

Endorsed Standards Override Mechanism

Introduction

An *endorsed standard* is a Java™ API defined through a standards process other than the Java Community Process℠ (JCP℠). Because endorsed standards are defined outside the JCP, it is anticipated that such standards may be revised between releases of the Java 2 Platform. In order to take advantage of new revisions to endorsed standards, developers and software vendors may use the Endorsed Standards Override Mechanism to provide newer versions of an endorsed standard than those included in the Java 2 Platform as released by Sun Microsystems.

Endorsed Standards Classes Deployment

Classes implementing newer versions of endorsed standards should be placed in JAR files. The system property `java.endorsed.dirs` specifies one or more directories that the Java runtime environment will search for such JAR files. If more than one directory path is specified by `java.endorsed.dirs`, they must be separated by `File.pathSeparatorChar`. If no value is set for `java.endorsed.dirs`, then Sun Microsystem's implementation of the Java 2 Platform looks for JAR files in a default standard location:

```
<java-home>\lib\endorsed            [Microsoft Windows]
<java-home>/lib/endorsed            [Solaris or Linux]
```

Here `<java-home>` refers to the directory where the runtime software is installed (which is the top-level directory of the Java 2

Figure 6-11 Endorsed Standards Override Mechanism

Classes implementing newer versions of endorsed standards should be placed in jar files. The system property java.endorsed.dirs specifies one or more directories that the Java runtime environment will search for such jar files.

If more than one directory path is specified by java.endorsed.dirs, they must be separated by File.pathSeparatorChar. If no value is set for java.endorsed.dirs, then Sun Microsystem's implementation of the Java 2 Platform looks for jar files in a default standard location:

```
<java-home>\lib\endorsed          [Microsoft Windows]
<java-home>/lib/endorsed          [Solaris or Linux]
```

Here `<java-home>` refers to the directory where the runtime software is installed.

The Java runtime environment will use classes in such jar files to override the corresponding classes provided in the Java 2 Platform as shipped by Sun.

For more information on JAXP in JDK 1.4 see the Java API for XML Processing Frequently Asked Questions Web page at:

http://java.sun.com/xml/jaxp/faq.html#JDK14

Part 2

Service-oriented architecture

In this part we provide a comprehensive introduction to services-oriented architecture (SOA) and Web Services. We discuss the services development environment, service-based solution topologies, JCA, and some design guidelines.

7

Service-oriented architecture and Web services

This chapter introduces *service-oriented architecture* (SOA) and how it can be used to develop new generation e-business applications by exploiting existing legacy applications on z/OS and OS/390.

It also presents details about how Web services-based solutions can be implemented on z/OS and OS/390 to satisfy the same requirements.

7.1 Introduction

Service-oriented architecture (SOA) is a concept specifying that an application can be made up from a set of independent but cooperating subsystems or services. Such a framework isolates each service and exposes only the necessary declared interfaces to other services.

This not only allows architects to organize and reduce dependencies in their products, but also provides for a tailored mix of services in the deployed environment. This approach can be used to support existing requirements, enterprise application integration, for example, as well as provide a foundation for extending the platform to meet specific business demands, such as rapid solution delivery for e-business.

The SOA model isolates aspects of an application so that, as technology changes, services (components) can be updated independently, limiting the impact of changes and updates to a manageable scope. Managing change is an important benefit of leveraging component architectures and models. If not managed well, change can result in the degradation of a modern Web application into unwanted complexity. A comprehensive design pattern can bring much needed structure to Web application development.

By re-engineering applications on an EIS server (for example CICS or IMS-based EIS) into services, we can utilize the Host applications as re-usable services to a wide range of applications for e-business. Using standard architectures like J2EE, specifically the J2EE Connector Architecture (JCA) component of the architecture, organizations can develop a capability to provide solutions by harvesting components from existing information assets (in-house applications, third-party packages, and so forth).

Also, with the help of a new generation of development tool sets, like WebSphere Studio Enterprise Developer (WSED), WebSphere infrastructure on z/OS, and XML Toolkit for z/OS, we are now in a position to modernize organizations' legacy assets to SOA-compliant components. One of the major benefits of being able to deliver this new architecture on the z/OS platform is reducing the "total cost of ownership" (TCO) of running e-business applications.

7.2 SOA definition

Any service-oriented architecture contains three roles: a service requestor, a service provider, and a service registry (see Figure 7-1).

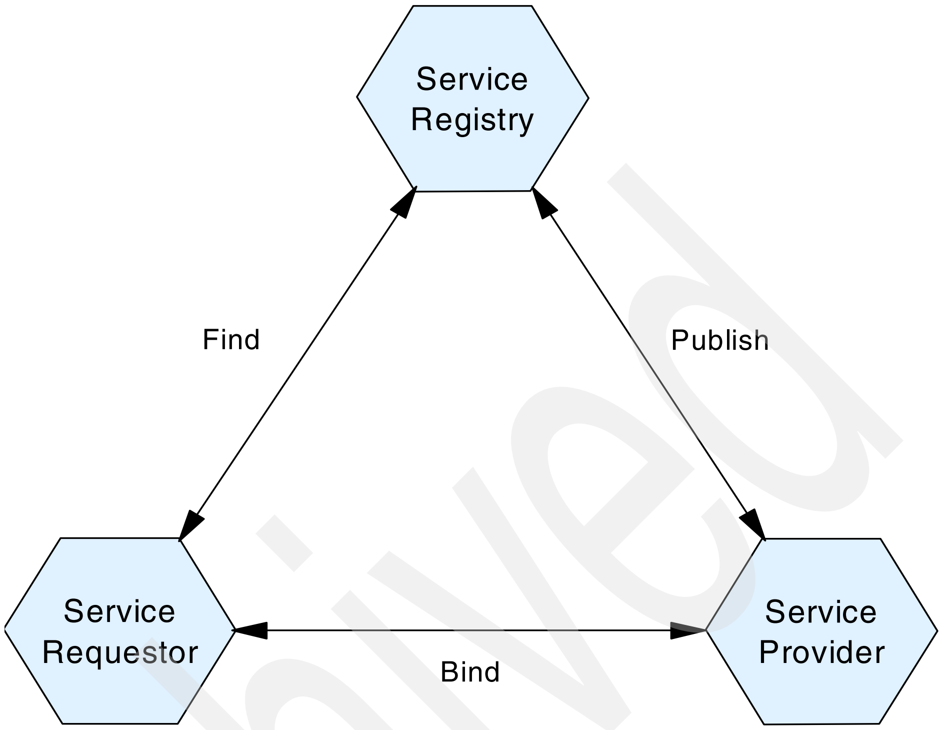


Figure 7-1 Web services

- A *service provider* is responsible for building a useful service, creating a service description for it, publishing that service description to one or more service registries, and receiving service invocation messages from one or more service requestors.
- A *service requestor* is responsible for finding a service description published to one or more service registries, and for using service descriptions to bind to or invoke services hosted by service providers. Any consumer of a service can be considered a service requestor.
- The *service registry* is responsible for advertising service descriptions published to it by service providers, and for allowing service requestors to search the collection of service descriptions contained within the service registry. Once the service registry makes a match between the service requestor and the service provider, the service registry is no longer needed for the interaction.

Any program or network node can play each of these roles. In certain scenarios a single program might fulfill multiple roles; for example, a program could be a service provider providing a service to downstream consumers, as well as a service requestor, itself consuming services provided by others.

SOA also includes three operations: publish, find, and bind. These operations define the contracts between the SOA roles.

- The *publish* operation is an act of service registration or service advertisement. It acts as the contract between the service registry and the service provider.
- With the *find* operation the service requestor states one or more search criteria, such as type of service, quality of service, and so forth. The result of the find operation is a list of service descriptions that match the find criteria.
- The *bind* operation embodies the client-server relationship between the service requestor and the service provider. The bind operation can be dynamic (or late), such as on-the-fly generation of a client-side proxy based on the service description used to invoke the service; or it can be very static (or early), such as a developer hand coding the way a client application invokes a service (that is, during application construction time).

For a more in-depth discussion of SOA and other related topics, refer to *Building Web Services with Java* by Steve Graham, *et. al.* (ISBN 0-672-32181-5).

7.2.1 Role of XML in SOA

The key to the entire service-oriented architecture approach is the service description itself. Many aspects of a service need to be communicated, and data-centric XML has been suggested as the basis for service descriptions. XML schema is the base data type mechanism in the service description; however, those organizations which are still using DTD can mitigate the type requirements by defining an abstract type library for the DTD.

7.2.2 SOA development strategy

Web Services, defined at length later in the chapter, are being developed as *the* means to implement an SOA approach. However, not all organizations are ready to embrace it right away. There are several problems, both technical and developmental. For example, SOAP protocol is still not efficient enough, WSDL and UDDI are still evolving, and so forth.

In this section we present a *tactical approach* to adopt an XML-based SOA that will enable an organization to make a start right away instead of waiting for a full-function Web services infrastructure to be in place. The model is based on practical experience implementing an XML-based service model for z/OS and

OS/390 environments. This approach fully complements the Web services development model.

The main reason for presenting this model is that re-engineering the existing organizational assets (transactions, for example) to develop Web services is not a trivial task. It requires a considerable amount of effort, plus knowledge and experience, to design a proper reusable Service. Some people think it is a simple one-to-one translation of an existing program module interface to XML tags. *It is not.* The services that we are going to harvest from legacy programs need to be made highly reusable, not just developed for solving single point-to-point solutions.

The essence of the service needs to be captured in a *data-centric XML interface*. The interface description needs to be properly typed, especially, the "Business Data" content of a message. By following a data-centric XML approach we can externalize the domain validation rules within a procedural code module, for example, a COBOL program. This will considerably reduce the load of exception messages over the Web.

Finally, the point we want to emphasize is that if you want to start on the path of *legacy application modernization*, then adopting a service-oriented architecture is not only the right way to start, it will secure your investment in terms of adoption of Web Services technology in a big way. In 7.2.4, "Service development approach" on page 167, we show you how this can be done.

7.2.3 Web Services Inter-operability Stack

In this section we present the tactical or short-term solution by comparing and contrasting it with the Web Services Inter-operability Stack model. Hopefully, this will show how the tactical approach complements the Web services model.

The three stacks of Web services are:

- Wire stack
- Description stack
- Discovery stack

The wire stack

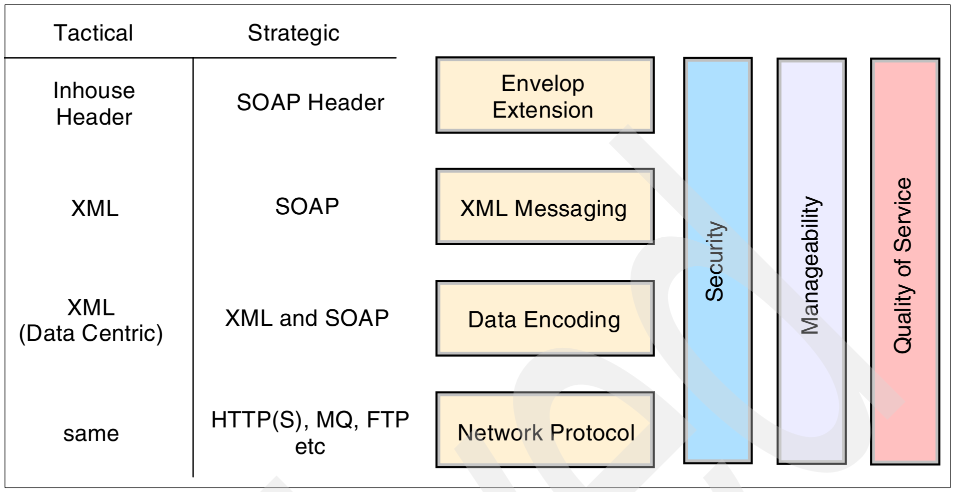


Figure 7-2 Web Services inter-operability

The wire stack represents the technologies that determine how a message is sent from the service requestor to the service provider. For the tactical approach, the protocol stack remains the same: HHTP/HTTPS, SMTP, FTP, MQSeries, and so forth.

At the Data Encoding level, there are several possibilities. Organizations which have not yet adopted XML Schema can use DTD with a “home grown” type library to implement data-centric XML-oriented messages.

The next two layers, SOAP encoding and SOAP Header, can be bypassed by adopting XML over RPC, and developing a message header convention to which the involved parties agree.

The description stack

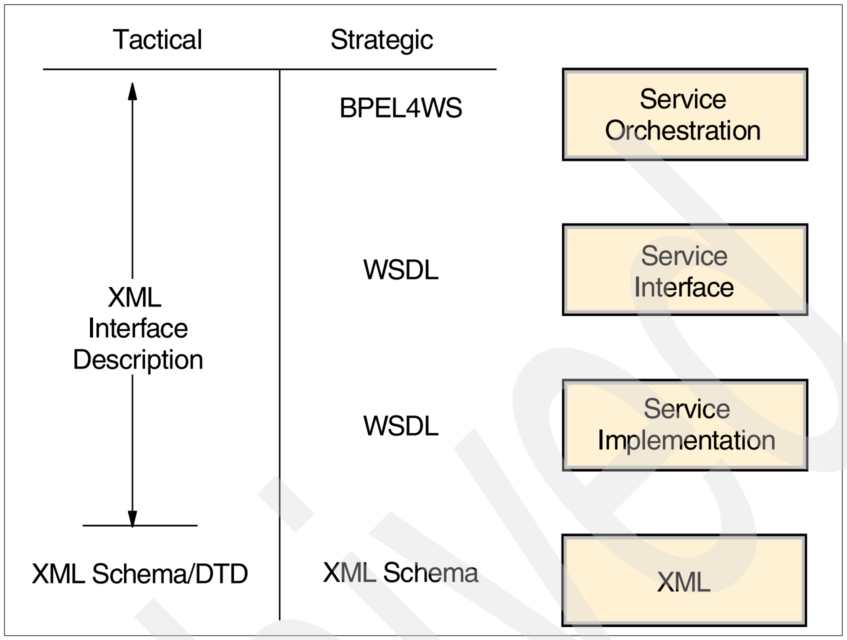


Figure 7-3 Description stack

The main purpose of service description is to communicate those aspects of a service that might be important to the service requestor. XML is the basis for service description. Web Services Description Language (WSDL) is the interface definition language for Web services. It describes the set of operations supported by a Web service, various binding contracts, and so forth. The topmost layer in the description stack is the *Service Orchestration* layer. This is implemented using Business Process Execution Language For Web Services (BPEL4WS) or BPEL for short.

For the tactical solution we present a rigorous data-centric XML interface of the service (that is, a pair of Request and Response messages), similar to the requirement of the PortType element in a WSDL document.

The discovery stack

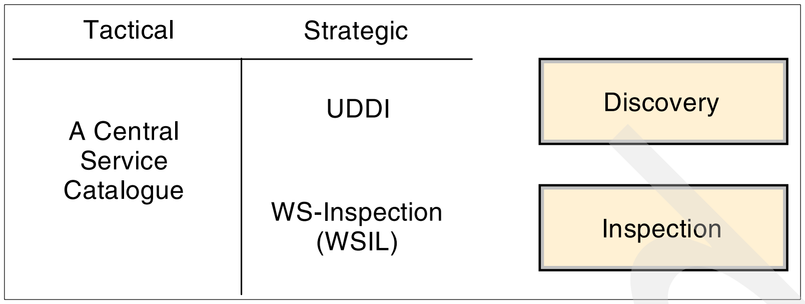


Figure 7-4 Discovery stack

The discovery stack organizes technologies associated with Web services discovery. The first level of the stack represents a simple inspection level. Inspection is a technique for discovering the service description given that the details about the service are already known.

Because service discovery is a very broad concept, it is unlikely that one solution addresses all of its requirements. The Universal Description, Discovery, and Integration (UDDI) specification addresses a subset of the overall requirements by using a centralized service discovery model. The Web Services Inspection Language (WS-Inspection) is another service discovery mechanism that addresses a different subset using a distributed usage model. The WS-Inspection specification is designed around an XML-based model for building an aggregation of references to existing Web service descriptions.

The discovery level represents the capability of discovering Web services and service providers using a capability-based lookup.

In the tactical approach both of these layers can be substituted for by a centralized service catalog facility, or better yet, by an XML Repository product. By using an XML Repository, we are alluding to an *early binding* scenario, meaning at design time. Developers do a service capability-based search to locate an appropriate service in the XML Repository. For example, within an organization, it will identify which teams and systems are responsible for the service. An XML interface definition of all the elements constituting the service interface will then be retrieved from the Repository.

In conclusion, we would like to reiterate, you can start building applications based on SOA architecture even without a full Web services infrastructure complement. In the following section we present a logical view of an XML-based services model drawn from a real-life implementation. Subsequently, in the Services Development Environment chapter, we show how tools like WSED can be utilized to automate generation of many components of Services. And in the Solution

Topology chapter, we show how different runtime models can implement applications based on SOA approach.

7.2.4 Service development approach

In this section we present an approach for converting an existing host transaction into an SOA-compliant service. The service will be delivered as a pair of Request and Response XML-encoded messages.

The steps in the process are as follows:

1. If the transaction is a conversational one, convert it into an atomic transaction, meaning request and response mode. The amount of effort required will vary depending on how well the code is written, the amount of documentation (if any) available, and the knowledge of the programmer about the system. However, there is some good news on the horizon: the WebSphere *Asset Analyzer*, a legacy asset harvesting infrastructure, will simplify the task enormously. Refer to 4.4, “WebSphere Studio Asset Analyzer” on page 99 for more information.
2. In designing the new service, keep the following design principles in mind:
   - Design by contract
   - Loosely coupled and highly cohesive modules
   - Encapsulation
3. Once the service interface (for example, a transaction) is established, design the XML rendering of this interface. Many programmers are tempted to use XML to encode data blindly in a one-for-one translation, meaning one element in a COBOL data structure translates to one XML tag. This is a gross under utilization of XML’s capability. At this stage you have to remember the capability of XML to capture metadata. We elaborate on this issue in 10.2, “XML-based message design” on page 223.
4. Design a pair of DTDs corresponding to a request/response interface to a transaction.

   We followed a three-tier architecture for defining the DTD. Even though we used DTD, the architecture holds true for Schema as well. The three-tier architecture is discussed in Chapter 10, “Some key design guidelines” on page 219.
5. Once the service interfaces have been designed, you can use the *Services Perspective* component of either WSED or WSAD/IE to develop the Service implementation. Refer to 4.3, “Support for enterprise service development” on page 97 to get an overview of this tool set.
6. At runtime, the conversion of the native data stream generated by a transaction (representing a service) can be done several ways. The available

options are presented in Chapter 8, “Some service-based solution topologies” on page 189. Also in that chapter we present various possibilities for delivering the XML stream to different client systems. All these options are z/OS and OS/390 platform based.

For a discussion of the architecture, refer to 10.2.1, “Architecture for XML messages” on page 223.

7.3 Web Services overview

As stated before, *Web Services* are the means to implement an SOA approach. *Web Services* are self-contained, self-describing, modular applications that provide certain operations and services through the Web. These applications are independent of specific operating systems, programming languages, or platforms. A typical situation is a customer making a transaction from a Web browser, accessing a Web Service, and generating a CICS transaction or a DB2 entry. Web Services guarantee a transparent layer so that the customer makes their transaction in the same way—regardless of whether it is in DB2 or another database management system. In this scenario, XML is the messaging way to connect this Web Service with the customers or with another Web Service (application). This is because XML guarantees the transparency needed by XML Web Services since XML is independent of the platform, operating system, or programming language (it is only text).

Each Web Service offers a set of services or operations to customers (or other Web Services), and it needs a *service description* which contains the necessary details to access it, like message formats, transport protocols, and location. These service descriptions are expressed in a standard language called WSDL (Web Services Description Language).

The idea of the Web Services is that they are published on the network, and then any customer or application (Web Service) can find them and invoke their operations.

7.3.1 Web Services components

Web Services are deployed on the Web by *service providers*, and their functions are described in a service description implemented in a WSDL document or a set of WSDL documents. But to let customers find Web Services, they need a third participant, the *service registry*. This is a registry of Web Services, functions provided by the Web Services, and protocols used by them.

An illustration of the components that make up Web Services is presented in Figure 7-5. The functions of the components are:

- Service provider: Creates the Web Service and its service definition, and publishes the service in the service registry based on the standard UDDI (Universal Description, Discovery and Integration) specification. This UDDI offers a set of APIs to ask the service registry for the needed Web Services.
- Service requestor: Finds the Web Service (once it has been published) via the UDDI interface supported by the service registry. Once it has found the service, it can bind and invoke the service with the URL provided by the service registry.
- Service registry: Provides a WSDL service description and a URL pointing to the service requested by the service requestor. These functions are supported by a standard set of APIs (UDDI).

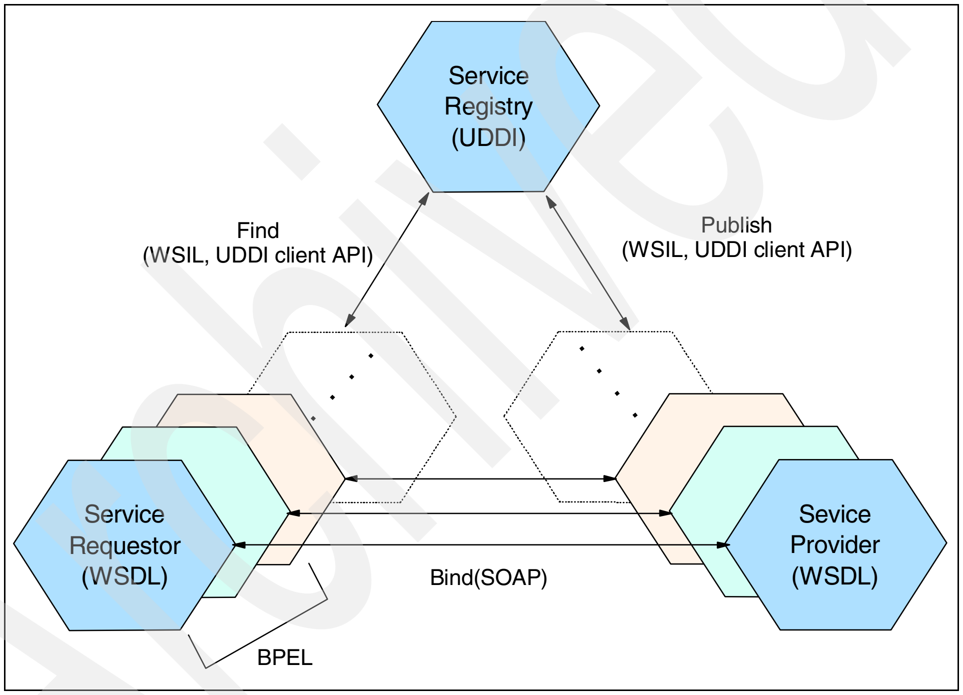


Figure 7-5 Web services

So, what is a Web Service? It is an interface that describes a collection of network-accessible operations, described using WSDL, published to UDDI and found in UDDI using WSIL; at execution time the service provider and requestor are bound by SOAP. Finally, a set of Web Services can be orchestrated to execute a Business Process in a certain order by a BPEL (Business Process Execution Language) document.

7.3.2 Web Services operations

As shown in Figure 7-5, there are three basic operations:

- Find: Search for the service and retrieve the service description locally or from a remote service provider. The service description will be used to bind with the service provider.
- Publish/Unpublish: The service provider registers its service via UDDI in a service registry providing the service description (publish), or it can unpublish an old Web Service, so it becomes unavailable.
- Bind: The service requestor contracts with service provider to get the available services and to invoke them. Bind operation is the negotiation between these two parties before the requestor can access services on the provider.

7.3.3 Web Services implementation

Web Services Description Language (WSDL)

Neither the requestor nor the provider is required to know the other platform, programming language, or distributed object model (if any). All these points are covered by the service description. WSDL is a basis for this service description, and now is supported by the W3C consortium as a standard.

The latest draft of the W3C specification can be found at:

http://www.w3.org/TR/2002/WD-wsdl12-bindings-20020709/

WSDL documents are XML documents (WSDL it is an XML-based interface) that describe Web Services in terms of a set of endpoints (input and output) and possible messages on each point. These messages can be RPC invocations or embedded documents. The description is very abstract, and is bound to the particular network protocol and message format used on this environment, defining what is called an end point. Many times, WSDL descriptions depend on the APIs used in a particular development environment.

WSDL defines:

- Web Services interfaces, including:
  - Operation types (one-way, request-response, notification)
  - Messages defining a Web Service
  - Data types (XML Schema)
- Web Service access protocol (SOAP over HTTP, and so forth)
- Web Services contact end points (Web Service URL, for example)

Compliant server applications must support these interfaces, and the customers know how to access these services thanks to the service description (WSDL documents).

There are several visual tools that are very helpful in the generation of all these components. In the WebSphere Studio Application Developer Integration Edition you will discover a number of graphical tools to guide you through the preparation of all the components and WSDL documents needed in the implementation of Web Services. And in WebSphere Application Server 4.01 for z/OS there is support for SOAP, with special SOAP extender classes optimized for this environment.

The following example, extracted from the samples provided by WSAD IE, could be part of a set of WSDL documents describing Web Services provided by a Java Bean.

Example 7-1 Service WSDL document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="UserManagerJava"
    targetNamespace="http://sample.flow/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
    xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
    xmlns:tns="http://sample.flow/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <import location="UserManager.wsdl" namespace="http://sample.flow/"/>
    <binding name="UserManagerJavaBinding" type="tns:UserManager">
        <java:binding/>
        <format:typeMapping encoding="Java" style="Java">
            <format:typeMap formatType="java.lang.String"
typeName="xsd:string"/>
            <format:typeMap formatType="boolean" typeName="xsd:boolean"/>
        </format:typeMapping>
        <operation name="addUser">
            <java:operation methodName="addUser" parameterOrder="name"
returnPart="result"/>
            <input name="addUserRequest"/>
            <output name="addUserResponse"/>
        </operation>
        <operation name="doesUserExist">
            <java:operation methodName="doesUserExist"
                parameterOrder="name" returnPart="result"/>
            <input name="doesUserExistRequest"/>
            <output name="doesUserExistResponse"/>
        </operation>
    </binding>
    <service name="UserManagerService">
```

```
        <port binding="tns:UserManagerJavaBinding"
name="UserManagerJavaPort">
            <java:address className="flow.sample.UserManager"/>
        </port>
    </service>
</definitions>
```

In this sample we are using a previously developed Java Bean called *UserManagerJava*. This Java Bean manages an internal list of users, providing two methods: addUser(user_name) to add a new user to the list, and doesUserExist(user_name) to check if the user passed as parameter was previously entered in the list.

In an XML document, each *element* (tag) can be qualified by an element called *name space*, that is a way to distinguish this element on this document from another element with the same name but defined in a different XML grammar. You can find it, for example, in the tag <java:operation.....>. In this tag, java is the prefix for the element, and operation is the proper element. The prefix java is associated with the namespace http://schemas.xmlsoap.org/wsdl/java (using the instruction <xmlns:java="http:.....">), so this is the namespace for operation.

Many times you import one XML document into another, and if both documents have an element with the same name (for example, both documents have an element called `operation`), we would have trouble distinguishing the operation element of one document from the operation element of the other document because after the import, we only have one document with the join of the two original documents. To avoid this problem, each element can be qualified by a URI (this URI may or may not exist; in fact, many times we use standard URIs such as `http://www.w3.org/2001/XMLSchema` that have been created only for the purpose of indicating that this is an XML Schema). This URI can be associated to the element by a prefix, so what you use to qualify an element is a short prefix (for instance, prefix `java`), but really what qualifies this element is the URI associated to that prefix.

Namespaces are defined by the user, but it is very common to find the conventions described in Table 7-1.

Table 7-1 Frequently used namespace conventions

Namespace	Description	Common prefix	URI
WSDL	Elements of a WSDL document. It is used as a default namespace for a WSDL file.	None	`http://schemas.xmlsoap.org/wsdl/`
W3C XML Schema Definition	Standard XML data and element types for which we have to define types.	xsd	`http://www.w3.org/2001/XMLSchema`
WSDL SOAP Link	Elements to define a SOAP link.	soap	`http://schemas.xmlsoap.org/wsdl/soap/`
WSDL HTTP Link	Elements to define a HTTP link.	http	`http://schemas.xmlsoap.org/wsdl/http/`
WSDL MIME Link	Elements to define a MIME message link with multiple parts.	mime	`http://schemas.xmlsoap.org/wsdl/mime`
User-defined types	Types defined by the user.	xsdl	Defined by the user
WSDL entities	Messages, operations, port types, links, and WSDL services created by the user.	tns	Defined by the user

Before we continue with the explanation of our example, we are going to review some of the most-used WSDL syntax. Be aware that there may be some variations from the following descriptions because of the development platform in use.

<definitions>	Root element on a WSDL document.
<documentation>	To insert commentaries.
<import>	To import other file into current, with new entities defined in the other file.
<types>	Type definitions.
<message>	Set of parts sent/received by an end point in a Web Service. We define the message in an abstract way, and then we can split it into parts (fields).
<part>	Each piece in which we divide the message.

<portType>	Service interface. It can have <operation> elements inside to define operations allowed on this interface. It can have <input>/<output> elements to define the input/output messages on this operations. It can have a <fault> element to identify a message structure previously defined.
<binding>	Description of the service offered by a port. It can have <operation> elements inside to define an operation on that port, and <input>/<output> elements to describe messages accepted by these operations.
<soap:binding>	SOAP link configuration. Inside it could have <soap:operation> elements to replace SOAP defaults, or to specify a <SOAPAction> object.
<soap:body>	To describe how to write message inside a SOAP envelope.
<service>	Set of ports that cooperate to give a Web Service.
<port>	Address where a service is accessible.
<soap:address>	Physical address where you have to send SOAP messages.

At the beginning of the document in Example 7-1 on page 171, we have a `definitions name` that is used to qualify this service definition and is the root element of the document. It is for informational purposes only, and its name is up to you. In our case, we used `UserManagerJava`.

Then we have a `targetNamespace` that points to the name space where all other definition components in this definition group will be defined. In our case, we created a Java package called sample.flow where we have two Java Beans: `UserManager` and `GroupManager`. These Java Beans are going to be the applications to provide the services. In the example, what we see are UserManager WSDL documents. WSDL documents for both Java Beans reside in the same URI (namespace).

After that is `xmlns="http://schemas.xmlsoap.org/wsdl/"` which is the default namespace for all elements in the XML document that are not qualified (elements with no prefix). This name could vary depending on the tool in use to generate the WSDL definition, or the WSDL version specification, but anyway it is a fixed namespace. For the latest draft of the WSDL specification, see:

`http://www.w3.org/TR/wsdl12/#aii-targetNamespace`

Following this definition, you have three lines (`xmlns:format`, `xmlns:java`, and `xmlns:tns`) that specify namespaces for elements that start with the prefixes format:, java: and tns:, and as you see, you have the possibility of associating an XML Schema to the WSDL document (prefix xsd). This would be used to control data format being used on this service. The prefix `format` is used for elements describing data types used in messages, prefix java is used for elements describing binding parameters (operations, parameters for the operations, and so forth), and prefix tns is used for target namespace (Java Beans in our example).

After all prefix definitions, you can see an `import` tag. This import tag includes other WSDL documents. It lets us divide the WSDL document into logical pieces, so the resulting documents are not so complex. In our example, we have separated the main WSDL document in two parts:

- Service WSDL definition file: This is the document shown in Example 7-1 on page 171. It describes the end service, in our example it describes the Java Bean with its operations.
- Interface WSDL definition file: In our example it could be the document shown on Example 7-2 on page 176. This document describes the message format sent/received.

Going on with our example, the next element is the `binding` element. Remember that one of the few rules XML documents have is that any opened element has to be closed, so the binding element is closed near the end of the WSDL document. A binding defines message format and protocol details for operations and messages defined by a particular communication. There may be any number of bindings for a given type of communication. This element has a name, `UserManagerJavaBinding` in our example, that must be unique among all bindings defined in the WSDL document, and a type (in this case `tns:UserManager`) that relates to the portType defined in the service WSDL document shown in Example 7-1.

With `<java:binding />` we are considering Java (class invocation) as the communication protocol, which means that a Java Bean is the service provider, and it is invoked via the Java platform. This is a particular implementation of WSAD IE, and it may vary depending on the development tool being used.

Under the `<java:binding>` tag we see a `<format:typeMapping>` tag, and inside it (between tags `<format:typeMapping>` and `</format:typeMapping>`) we have a mapping between Java types and type names used in the description of the messages in the WSDL document. With

```
<format:typeMap formatType="java.lang.String" typeName="xsd:string" />
```

what we are creating is a new type name called xsd:string that relates to Java type String, so we can say later in the WSDL document that a message is of xsd:string type.

With the `operation` tag we define two operations that this Java Bean, or service provider, is going to support: `addUser` and `doesUserExist`. These two names must be unique in the WSDL document. Inside the `operation` tag indicated between `<operation>` and `</operation>`) you can find the Java Bean method invoked when using this operation, with attribute methodName, and parameters used with that method (with attribute parameterOrder that specifies the order of the parameters as well). With the returnPart attribute we specify the method result. All these parameters are embedded in messages, and the format of these messages is described on the interface WSDL definition file shown in Example 7-2. At this point, on the operation tag, we only specify the names of input/output messages accepted, and later we will describe them.

Finally, we have a service section, with a service tag, that describes the provided service. First of all, you give this service a unique name (name attribute) among all service sections defined on the service definition, and then you specify a port inside that service section. A port describes an individual starting/end point, giving the address to use to access this port. If you look at our example, we have two attributes and a sub-element on the port tag:

- binding="tns:UserManagerJavaBinding" refers to the binding previously declared; it is qualified with a namespace.
- name="UserManagerJavaPort" is a unique name among all ports given to this specific one.
- java:address is the address associated to this port. In our case, because Java is the protocol being used, it is the name of a Java class, flow.sample.UserManager.

The last point we have to describe is the message format. We do this in the interface WSDL definition document, shown in Example 7-2.

Example 7-2 Interface WSDL document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="UserManager" targetNamespace="http://sample.flow/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://sample.flow/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <message name="addUserRequest">
        <part name="name" type="xsd:string"/>
    </message>
    <message name="addUserResponse">
        <part name="result" type="xsd:boolean"/>
    </message>
    <message name="doesUserExistRequest">
        <part name="name" type="xsd:string"/>
    </message>
    <message name="doesUserExistResponse">
        <part name="result" type="xsd:boolean"/>
    </message>
    <portType name="UserManager">
        <operation name="addUser" parameterOrder="name">
            <input message="tns:addUserRequest" name="addUserRequest"/>
            <output message="tns:addUserResponse" name="addUserResponse"/>
        </operation>
        <operation name="doesUserExist" parameterOrder="name">
            <input message="tns:doesUserExistRequest"
name="doesUserExistRequest"/>
            <output message="tns:doesUserExistResponse"
name="doesUserExistResponse"/>
        </operation>
    </portType>
</definitions>
```

Between each pair of tags, `<message>` and `</message>`, we describe the message name (related to those specified on input and output parameters in the `<operation>` tag), the part name (parameters of the message) and the type of the part.

The description of the WSDL document, which in the end is an XML document, is very easy. But we have two points to consider:

- We need a set of APIs to manipulate WSDL documents. For example, JSR-110 (Java Specification Request) provides a set of APIs for representing and manipulating services described by WSDL. With these APIs, WSDL documents can be parsed in an uniform way by the client, regardless of the origin of the description. These APIs are included in the Java 2 SDK Standard Edition v 1.3, and are approved by the Java Community Process (JCP).
- WSDL documents of our UserManager Java Bean sample specify the end points to communicate with it. In the package sample.flow we have another Java Bean called `GroupManager`, which is going to manage groups of users. We need WSDL documents to describe end points of this second Java Bean, similar to the one we made in the previous sample. When we have all the WSDL documents to describe the end points to access the Java Beans, we still need to define the interactions between the two Java Beans, and the flow of messages/data.

Universal Discovery Description and Integration (UDDI)

UDDI is a standard Web service to provide directory services. It lets applications or programmers find Web services at runtime or development time.

There are two types of UDDI registries: public and private. A list of the public UDDI registries is at the following URL:

`http://www.uddi.org/`

These UDDI registries are public search engines to find all published Web services. When you publish something on a public UDDI registry, it is reflected in the other UDDI registries.

Private registries may be a particular company registry for internal purposes, or a commercial sector registry, handled by a group of companies. IBM provides an UDDI registry that you can deploy in a servlet, using DB2 as data container.

You have three UDDI search classes:

White pages:	Companies by name
Yellow pages:	Companies by service type
Green pages:	Technical issues to access certain Web services

UDDI4J is part of the IBM Web Services Toolkit. It is a set of Java APIs to access these UDDI registries.

UDDI defines the following entities:

Enterprise entities:	Service providers, URLs, contact data (telephone, address, and so forth)
Enterprise services:	Service types operated by an enterprise entity
Links layouts:	Technical specifications to access Web services of an enterprise service

UDDI has a metadata construct called *tModel*. Specification of tModel can be found at:

`http://www.uddi.org/`

With tModel you can:

- Define a Web service (electronic fingerprint)
- Define rules to interpret data parts

UDDI4J is supported on WebSphere Application Server 4.01, and it is an open source Java class library that provides the necessary APIs to operate with UDDI registries. UDDI4J includes the client-side implementation, source code, and javadoc for the APIs.

Web services discovery

Currently there are two approaches to Web services discovery using UDDI directory. These are:

- UDDI Client API for Java, C++, and so forth
- WSIL

In the first approach Web services discovery is done through the use of a centralized model (tModel). Requests pertaining to the service and business-related information are issued directly against the UDDI repository using client-specific UDDI APIs.

In the second approach, the WS-Inspection (WSIL) relies upon a completely distributed model for providing service-related information; the service description may be stored at any location, and requests to retrieve the information are generally made directly to the entities that offer the services. The WS-Inspection relies upon UDDI to define the description format. For a detailed discussion of the WS-Inspection spec, see:

`http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html`

SOAP and XMLP

The current industry standard for XML messaging is SOAP, but it is being replaced by another new standard, *XML Protocol* (XMLP), which is going to be the new standard for XML messaging.

At the moment, W3C has a draft for XMLP. The requirements for XMLP can be found at:

`http://www.w3.org/TR/2002/WD-xmlp-reqs-20020626`

The last version of SOAP is SOAP 1.2, and the W3C is working toward the reconciliation of SOAP 1.2 and XMLP, so in the future both protocols could work together. The current working draft for SOAP 1.2 can be found at:

`http://www.w3.org/TR/2002/WD-soap12-part0-20020626/`

Anyway, SOAP is the most important basis for the new XMLP protocol.

SOAP uses XML to interchange structured data between network applications. SOAP has three components:

Envelope:	Structure that describes what is in the message.
Rules:	A set of rules to define the structure of the data types defined in the applications.
Conventions:	Describe RPC (Remote Procedure Calls) invocations and responses carried in the SOAP Body element.

SOAP can cooperate with other protocols like HTTP, FTP, SMTP, RMI/IIOP, MQSeries, and so forth, or it can be enveloped by these protocols.

Many times, Web services developers have to use optimized programming language-specific bindings generated from WSDL, and many times WSDL service definitions are created with visual tools such as WebSphere Studio Application Developer Integration Edition which, along with its Enterprise Services Toolkit, provides a set of tools to simplify the development of the Web Services at the SOAP level. SOAP has extensions in WebSphere Application Server 4.01, and it is supported in this environment.

The steps involved in a SOAP communication are as follows:

1. The service requestor creates a SOAP message. In the body of the SOAP message there is an XML document that can be a SOAP RPC request or a document-centric message, as indicated in the service description. The service requestor gives its SOAP message and the address of the service provider to its SOAP support (SOAP client runtime or SOAP platform). This SOAP support sends this message to the selected address via the network protocol being used (HTTP, FTP, MQSeries, and so forth).
2. The message travels across the network to the SOAP support of the host where the service provider resides. This SOAP support is responsible for converting the XML message into programming language constructs if it is required by the application that provides the service. Within the message could be encoding schemes to control this conversion.
3. The Web service parses the message and generates a response. This response is a SOAP message as well. This time, the SOAP support in the host where the Web service resides gives the address of the service requester as the destination, and sends the SOAP message with the response to the original point.
4. The SOAP support in the host where the service requester exists receives the SOAP message and converts it into the programming language constructs expected by the requester application.

The transmission between the service requester and the service provider can be synchronous or asynchronous, and can use different types of schemas, such as one-way messaging (no response), notification (push-style response), or publish/subscribe.

The most important part of a SOAP XML document is the *SOAP envelope*, that is, the top-level element in the message. In this envelope you can send many things, so it is like a mail envelope into which you can put a letter. This envelope contains an optional *SOAP header* and a mandatory *SOAP body*. The general structure is illustrated in Example 7-3.

Example 7-3 SOAP envelope

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http:// "
     <SOAP-ENV:Header>
        ...........
        ...........
     </SOAP-ENV:Header>
     <SOAP-ENV:Body>
        ...........
        ...........
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP Header may be used to add features such as authentication, transaction management, and so forth, and the SOAP Body is where our message is going to be placed.

Going back to our primary example, we next have a WSDL document binding SOAP for a client trying to access our Web service. WSDL has extensions for SOAP 1.2 Bindings, but it has extensions for other protocols as well, like HTTP and others, as shown in Example 7-4.

Example 7-4 SOAP binding

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AddUserFlowPortTypeSOAPBinding"
    targetNamespace="http://www.add.user.flow/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.add.user.flow/">
    <import location="AddUserFlowDef.wsdl"
namespace="http://www.add.user.flow/"/>
    <binding name="AddUserFlowPortTypeSOAPBinding"
type="tns:AddUserFlowPortType">
        <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="addUser">
            <soap:operation soapAction="urn:AddUserFlowPortType" style="rpc"/>
            <input name="addUserRequest">
                <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="urn:AddUserFlowPortType"
                    parts="userName groupName" use="encoded"/>
            </input>
            <output name="addUserResponse">
                <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="urn:AddUserFlowPortType" parts="result"
use="encoded"/>
            </output>
        </operation>
    </binding>
    <service name="AddUserFlowPortTypeService">
        <port binding="tns:AddUserFlowPortTypeSOAPBinding"
name="AddUserFlowPortTypeSOAPPort">
            <soap:address
location="http://localhost:8080/flowSampleWeb/servlet/rpcrouter"/>
        </port>
    </service>
</definitions>
```

We use the tag `<soap:binding>` to specify that the protocol in use with WSDL is SOAP, since there are other possibilities. Inside the tag `<soap:binding... >`, with attribute *`style`*, we indicate that the SOAP message style is of type RPC (Remote Procedure Call), the *`transport`* attribute shows that the network (transport) protocol we are going to use is HTTP. You could use other protocols, like SMTP, JMS, and so forth.

We have created one operation for this port. For this port, the operation name must be unique. We assign the name *`addUser`*, with the *`name`* attribute.

Inside the `operation` tag, we can see a `<soap:operation... >` element declared. We need this element if we have to replace the predefined SOAP codification style or if the SOAP transport in use needs an attribute `SOAPAction`. In `SOAPAction` we specify the header URI used in HTTP messages; in our case that is the name of the Web service to be invoked, because we said that `style` is rpc.

Between the tags `input` and `output` we specify what is called the SOAP envelope, that is, the incoming/outgoing messages. Inside them you can find the `body` with `attribute` parts; a name list with the parts of the message to put in the SOAP body; a `namespace` name to be used for elements created in the codification process; `use`, which can be *`encoded`* if it is necessary to code data with a codification schema; and in this case, `encodingStyle`, which specifies the code style to be used.

Again, you can see a `port` directive, this time specifying the real location where it can find the *`rpcrouter`*, which is the component in charge of finding the remote endpoint. (It is a servlet in the remote server in charge to find the necessary components.)

Business Process Execution Language for Web Services

Web Services are self-contained business activities defined within a PortType element of a WSDL document. Usually the business functionality is implemented as well-defined transactions (that is, request/response) in the Enterprise Information Server (EIS) tier. Typically, these could be an atomic IMS or CICS transaction, a DB2 stored procedure, and so forth.

However, Web Services are too granular to deliver any business-worthy functions. On the other hand, a *Business Process*, which could be composed of one or more free-standing *Business Activities*, could be identified as an artifact of value achieving some business objectives. The value of creating business processes for an enterprise is in the intellectual assets that those processes represent.

Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short), allows specification of business processes and how they relate to Web Services. This includes specifying how a business process makes use of Web Services to achieve its goal, as well as specifying Web Services that are provided by a business process. A BPEL business process interoperates with the Web services of its partners—whether or not these Web services are implemented based on BPEL.

The role of BPEL is to define a new Web Service by composing a set of existing services. As such, it is an executable process implementation language. BPEL uses WSDL to specify actions that should take place in a business process. The interface of the composite service is described as a collection of WSDL PortTypes, just like any other Web Services. The composition (called the process) indicates how the service interface fits into the overall execution of the composition. Figure 7-6 illustrates an outer view of BPEL.

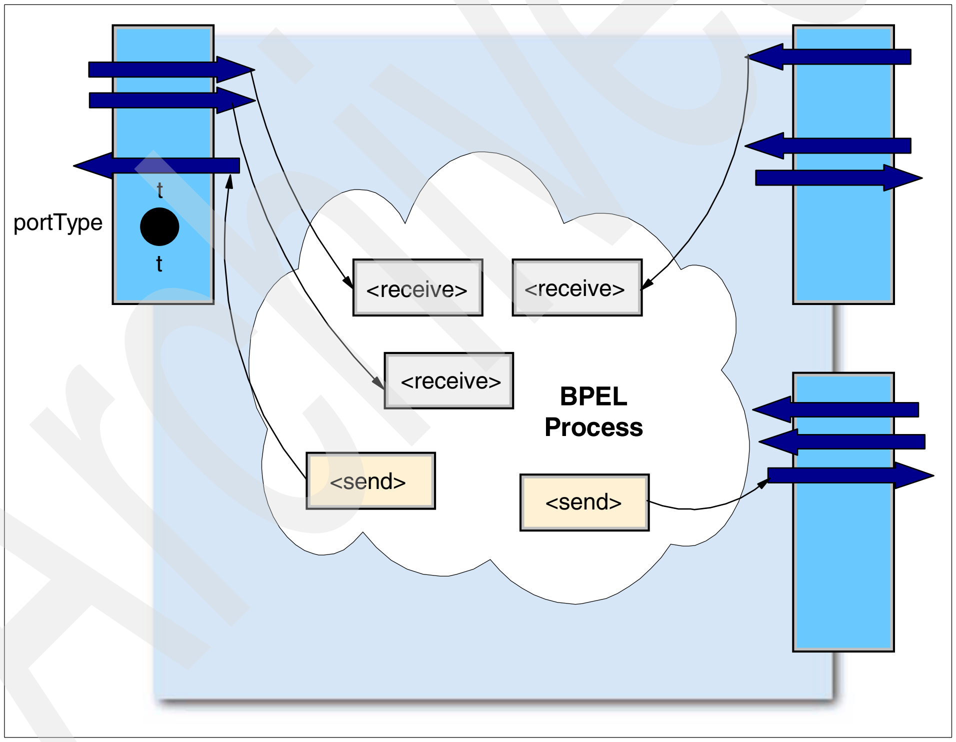


Figure 7-6 An outer view of BPEL

One of the important points to notice in the diagram is that one entry point (portType) to a process implements a single BPEL process. The process itself could be made up of multiple activities that each correspond to a specific WSDL document with its own portTypes handling a service implementation.

Specific *entry points* that correspond to external users invoking the operations of the interface are indicated within the BPEL description. These entry points consume the WSDL operation's incoming messages from input-only or input-output operations. BPEL only uses and supports input-only and input-output (request-response) operations of WSDL; output-only (notification) and output-input (solicit-response) operations are *not* supported.

The BPEL process itself is basically a flowchart-like expression of an algorithm. Each step in the process is called an *activity*. There is a collection of primitive activities, expressed as XML elements, including:

- Invoking an operation on some Web service `<invoke>`
- Waiting for a message to operation of the service's interface to be invoked by someone externally `<receive>`
- Generating the response of an input/output operation `<reply>`
- Waiting for some time `<wait>`
- Copying data from one place to another `<assign>`
- Indicating that something went wrong `<throw>`
- Terminating the entire service instance `<terminate>`
- Doing nothing `<empty>`

These primitive activities can be combined into more complex algorithms using any of the structure activities provided in the language. These include the ability to:

- Define an ordered sequence of steps `<sequence>`
- Have branching using case-statement approach `<switch>`
- Define a loop `<while>`
- Execute one of several alternative paths `<pick>`
- Indicate that a collaboration of steps should be executed in parallel `<flow>`

  Within activities executing in parallel, one can indicate execution order constraints by using the links.

BPEL allows you to recursively combine the structured activities to express arbitrarily complex algorithms that represent the implementation of the service. It can compose a set of services to a new service. This is done using the `<invoke>` activity and the `<receive>` and `<reply>` activities.

For detailed descriptions of BPEL elements, refer to the BPEL4WS specifications at:

http://www-106.ibm.com/developerworks/library/ws-bpel/

An example of a BPEL process

In this section we describe how BPEL works by examining a simple process scenario of an application for a travel agency to accept an itinerary from a customer, purchase the tickets from the airline, receive the tickets from the airline, and hand deliver them to the customer.

The activity diagram in Figure 7-7 presents the business process flow.

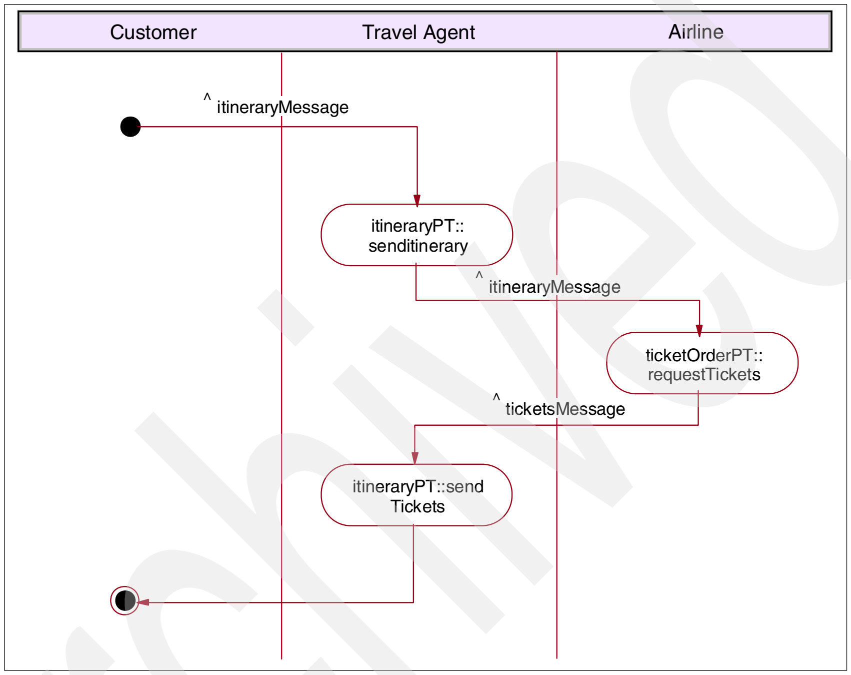


Figure 7-7 BPEL process

A travel agent specifies a business process called ticketOrder (Example 7-5, line 1). The purpose of this simplistic business process is to allow the agent to receive from a customer an itinerary (lines 20 to 23), to pass on this itinerary to an airline requesting the corresponding tickets (lines 26 to 29), and finally to receive these tickets from the airline (lines 33 to 36). To keep the example simple, it is assumed that the tickets will be picked up by the customer in person.

Example 7-5 Business process - ticketOrder

```
1 <process name="ticketOrder">

2   <partners>
3      <partner name="customer"
4      serviceLinkType="agentLink"
5      myRole="agentService"/>
```

```
6     <partner name="airline"
7     serviceLinkType="buyerLink"
8     myRole="ticketRequester"
9     partnerRole="ticketService"/>
10 </partners>

11 <containers>
12     <container name="itinerary" messageType="itineraryMessage"/>
13     <container name="tickets" messageType="ticketsMessage"/>
14 </containers>

15    <flow>
16    <links>
17    <link name="order-to-airline"/>
18    <link name="airline-to-agent"/>
19    </links>

20    <receive partner="customer"
21       portType="itineraryPT"
22       operation="sendItinerary"
23       container="itinerary"
24    <source linkName"order-to-airline"/>
25    </receive>

26 <invoke partner="airline"
27       portType="ticketOrderPT"
28       operation="requestTickets"
29       inputContainer="itinerary"
30    <target linkName"order-to-airline"/>
31    <source linkName"airline-to-agent"/>
32 </invoke>

33 <receive partner="airline"
34       portType="itineraryPT"
35       operation="sendTickets"
36       container="tickets"
37    <target linkName"airline-to-agent"/>
38 </receive>
39    </flow>
40 </process>
```

The set of partners the agent's process interacts with are defined in lines 2 to 10:

- Lines 3 to 5 introduce the partner customer.
- Lines 6 to 9 introduce the partner airline.

A partner definition involves specifying the Web services mutually used by the partner or process.

The messages that are persisted by the process are called containers (lines 11 to 14). Containers are WSDL messages that are received from or sent to partners. For example, the process stores an itineraryMessage as itinerary container.

The itineraryMessage is received from the customer (line 20) when the customer uses the sendItinerary operation of the agent's itinerary port (lines 21 and 22). This message is stored in the itinerary container (line 23) once received. The process then passes on the itinerary message to the airline (line 26) by using the requestTicket operation of the ticketOrder port (lines 27 and 28); this message is a copy of the itinerary container (line 29).

To define the order in which the activities have to be performed, the ticketOrder process structures its activities as a flow (line 15). A flow is a directed graph with the activities as nodes and so-called links as edges connecting the activities. The links required to define the flow between the ticketOrder process's activities is specified in lines 16 to 19. An activity specifies the links it is a source or target of. For example, the receive activity of line 20 is the source of the order-to-airline link (line 24); this link has the invoke activity of line 26 as target (line 30).

This example is taken from an excellent introductory paper: *“Business process in a Web Services world,”* by Frank Leymann and Dieter Roller. It can be found at:

```
http://www-106.ibm.com/developerworks/webservices/library/ws-bpelwp/
```

You can download an early implementation of the BPEL engine, as well as a BPEL modeling tool that is a plug-in for WSED, from IBM alphaworks site. It contains the Business Process Execution Language for Web Services Java Run Time (BPWS4J) package, which includes: a platform upon which business processes written in BPEL4WS can be executed, samples demonstrating the use of BPEL4WS, and a tool to validate BPEL4WS documents.

```
http://www.alphaWorks.ibm.com/tech/bpws4j?Open&ca=daw-flut-052203
```

Although a business process (a BPEL document) describes the flow of tasks, the order in which they need to be performed, the type of data shared, and how other partners are involved; there is still the need to coordinate business processes and transactions within the enterprise, and with partners and customers across heterogeneous systems and within the enterprise.

WS-Coordination and WS-Transaction specification have been developed to provide companies with a reliable and durable way of handling multiple Web services interactions, regardless of the underlying computing infrastructure. In addition, they outline how partners can interact with a collection of Web services and coordinate the outcome of those corresponding activities.

WS-Coordination

WS-Coordination provides developers with a way to manage the operations related to a business activity. A business process may involve a number of Web services working together to provide a common solution. Each service needs to be able to coordinate its activities with those of the other services for the process to succeed. Coordination involves the sequencing of operations in a process to reach an agreement on the overall outcome of the business process.

WS-Coordination provides the structure under which coordination can take place. The specification supplies standard mechanisms to create and register with transaction protocols that coordinate the execution of distributed operations in a Web services environment. WS-Coordination will help developers control operations that span interoperable Web services. For more information on WS-Coordination, visit:

```
http://www.ibm.com/developerworks/library/ws-coor/
```

WS-Transaction

WS-Transaction allows businesses to monitor the success or failure of each specific, coordinated activity in a business process. It provides businesses with a flexible transaction protocol to help enable consistent and reliable operations across distributed organizations in a Web services environment. The specification also allows the business process to react to faults detected during execution.

WS-Transaction provides for short- and long-running transactions in which resources cannot be locked for the duration of the business process. In both cases, WS-Transaction takes advantage of the structure WS-Coordination provides to enable all participating Web services to end the business process with a shared understanding of its outcome.

The use of WS-Transaction with WS-Coordination helps ensure that tasks, no matter how they are distributed across programming platforms and companies, all succeed or fail as a unit. For details about the WS-Transaction specification, see:

```
http://www.ibm.com/developerworks/library/ws-transpec/
```

Conclusion

While BPEL defines a business process and the connections with customers, partners, and internal entities, WS-Coordination and WS-Transaction complement BPEL by providing a way for companies to coordinate and integrate a number of distinct Web services and business processes, consistently and reliably, across a variety of implementation environments, to ensure the right outcome.

Some service-based solution topologies

In this chapter we show how to develop an XML-based solution on z/OS and OS/390 platforms. The solution topology will be based on SOA concepts discussed previously. Some of the solution topologies are based on real-life experience.

The solution topology consists of two categories of applications:

- The first category provides an approach to develop an XML interface to an existing host application (for example, IMS or CICS) using WebSphere Application Server infrastructure on z/OS and OS/390.
- The second category provides some models to build new applications, either in a CICS/IMS TP environment or natively using the XML Toolkit for z/OS and OS/390 and the JVM built into the CICS and IMS environments.

The classification of solutions is based on a *best practice* perspective. Users can use any of these solution patterns to address their particular needs.

In the solution patterns we have put quite a bit of emphasis on J2EE Connector Architecture (JCA) in lieu of other possible design architectures for accessing host/legacy data. This was done deliberately, to encourage architects and designers to consider JCA before considering other options. The JCA-compliant connectors/resource adaptors are matured components now available on both z/OS and OS/390 platforms. Chapter 9, "JCA and WebSphere connectors" on page 209 provides a detailed discussion of JCA. As well, there are several other redbooks that deal with various alternatives for accessing legacy transactions and data, for example *Using XML on z/OS and OS/390 for Application Integration*, SG24-6285.

8.1 Solution topology for legacy systems

The solution topology presented in this section belongs in the first category. The solution architecture shows how an XML message-based approach can be adopted to implement an SOA solution. The legacy applications are existing production applications developed in COBOL, PL/1, C, or C++.

8.1.1 IMS Transaction Manager

Figure 8-1 presents an overview of the scenario.

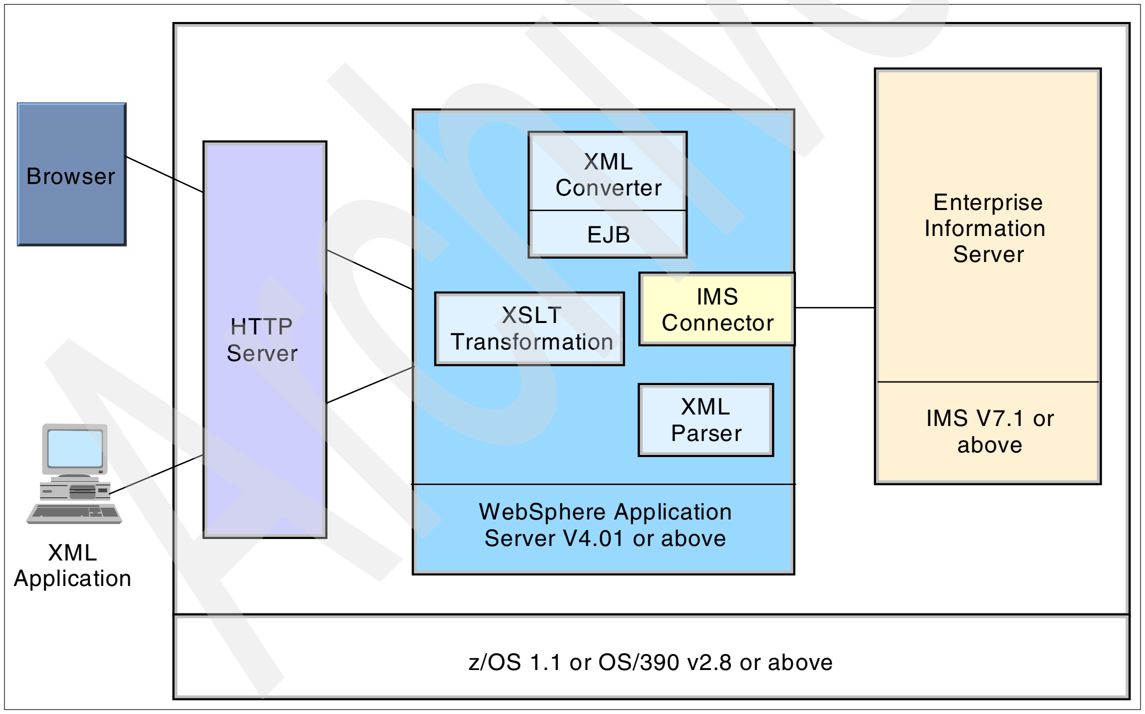


Figure 8-1 An IMS topology

The first step in solution building is to investigate the solution architecture of the legacy system. There may be quite a few surprises depending on the age of the system, whether any system documentation is available, and so forth.

One of the crucial decisions must be made at this stage:

1. Does the organization want to re-engineer the applications into a set of re-usable services—loosely coupled and highly cohesive components? This choice is ideal if the various components have potential use in other parts of the organization, in which case you might want to publish them (initially in an enterprise service catalog, and later in a UDDI).
2. Is this effort one that relates only to a specific business problem, with no likelihood of reusing the services in any other situations.

Which route you choose is critical in terms of development effort required, and determines the best choice of tools and techniques to build the intended solution.

WSAD/IE or WSED can generate all the connector components. You can use the XSLT transformation (XALAN component) capability to transform the XML stream to render it as HTML pages on the browser or to other XML formats. Again you can use WSAD or WSED to generate these converter objects.

Message flow

A high-level view of the message flow is as follows:

1. Through a Web browser, a client enters the transaction input data on an HTML page.
2. The transaction input data is transmitted from the Web browser to the Web server on OS/390 and WebSphere Application Server invokes a Java servlet.
3. The servlet invokes IMS Connect to run the IMS transaction. The Java application or servlet acts as a TCP/IP client to IMS connect. The transaction request is routed to IMS on the production system using the IMS connector for Java.
4. The transaction output is returned to IMS and then returned to the Web browser through the servlet.

The recommended versions of the software are: WebSphere Application Server V4.0.1, IMS V7.1, z/OS V1.1 or OS/390 V2.8, or above.

8.1.2 CICS Transaction Server

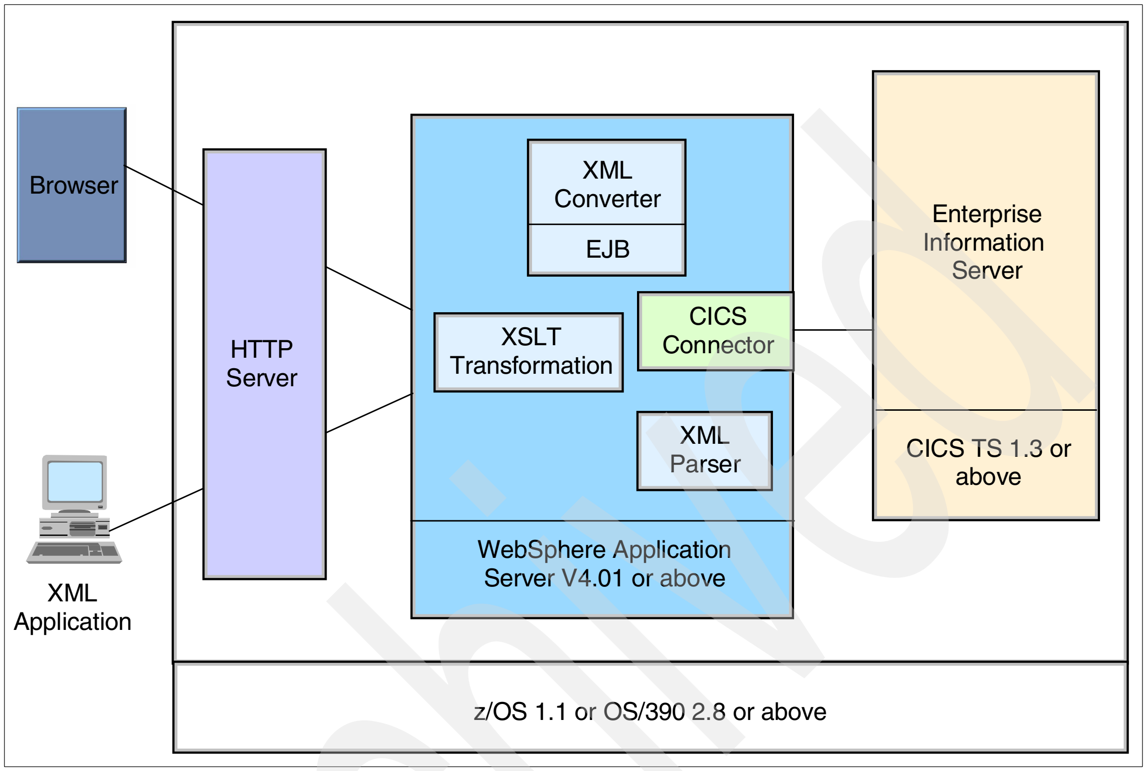


Figure 8-2 CICS TS

The J2EE Connector Architecture (JCA) specifies a standard way to access back-end enterprise systems in the J2EE environment. In JCA, the application server communicates with back-end enterprise systems through a resource adaptor or connector. A resource adaptor is specific to the back-end enterprise system and can be plugged into any application server.

The CICS resource adaptor, or connector, not only implements the J2EE connector interface, but is also RRS-compliant. It is designed specifically to work with the resource recovery services (RRS) component of z/OS. Resource recovery consists of the protocols and program interfaces that allow WebSphere for z/OS and CICS to work together to make consistent changes to multiple protected resources and participate in two-phase commit processing.

The CICS ECI resource adaptor is currently available with CICS Transaction Gateway V5.0.

From an application developer's perspective, as far as logical message flow is concerned, it is similar to the IMS scenario described previously.

For a detail CICS implementation scenario, refer to the IBM Redpaper "From code to deployment: Connecting to CICS from WebSphere v4.0.1 for z/OS," REDP0206.

8.1.3 WebSphere MQ

In this solution pattern, we present a hybrid architecture, consisting of MQ and WebSphere to access data from a *non* z/OS or OS/390 to a z/OS or OS/390 platform. Figure 8-3 provides an overview of the architecture.

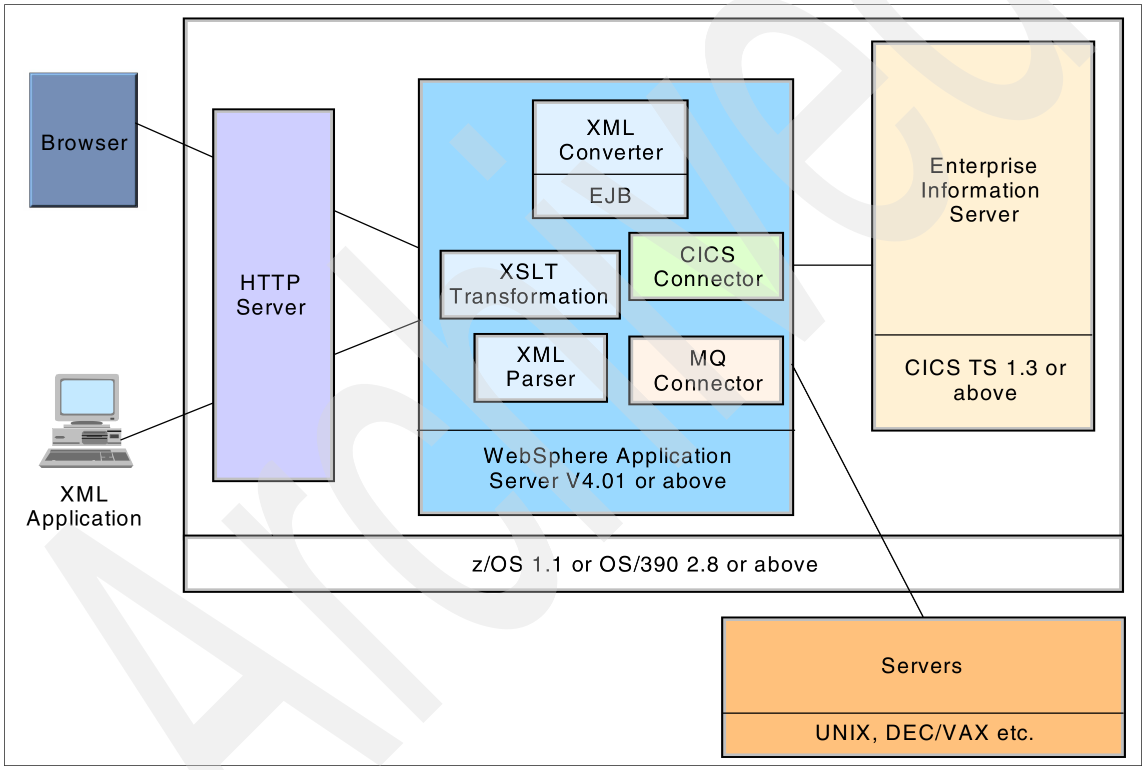


Figure 8-3 WebSphere MQ

In this solution pattern, MQ Series is utilized to access enterprise applications running on disparate platforms such as UNIX, DEC/VAX, and so forth. MQ Series is available on more than eighteen platforms, and as such, it is an ideal way to integrate enterprise applications running on disparate platforms.

MQ Series is an RRS-compliant resource manager that is able to participate in two-phase commit with other resource managers, like IMS DB and DB2. What this means is that any data put on MQ would be visible to receiving system if the data has been successfully committed by RRS.

Note: The current version of WSAD/IE (Sept.'02) does not yet provide support for MQ Connector; however, there is plan to implement such a connector in the not too distant future, based on JMS.

8.2 Solution topology for new applications

As XML capabilities are becoming more pervasive across various layers of technology platforms (z/OS, TP monitors, and the like), and within programming language domains (COBOL, for example), new possibilities are emerging for the design of applications to embed XML capabilities in solution domains. In this section we present some solution patterns for z/OS applications.

8.2.1 IMS Transaction Manager

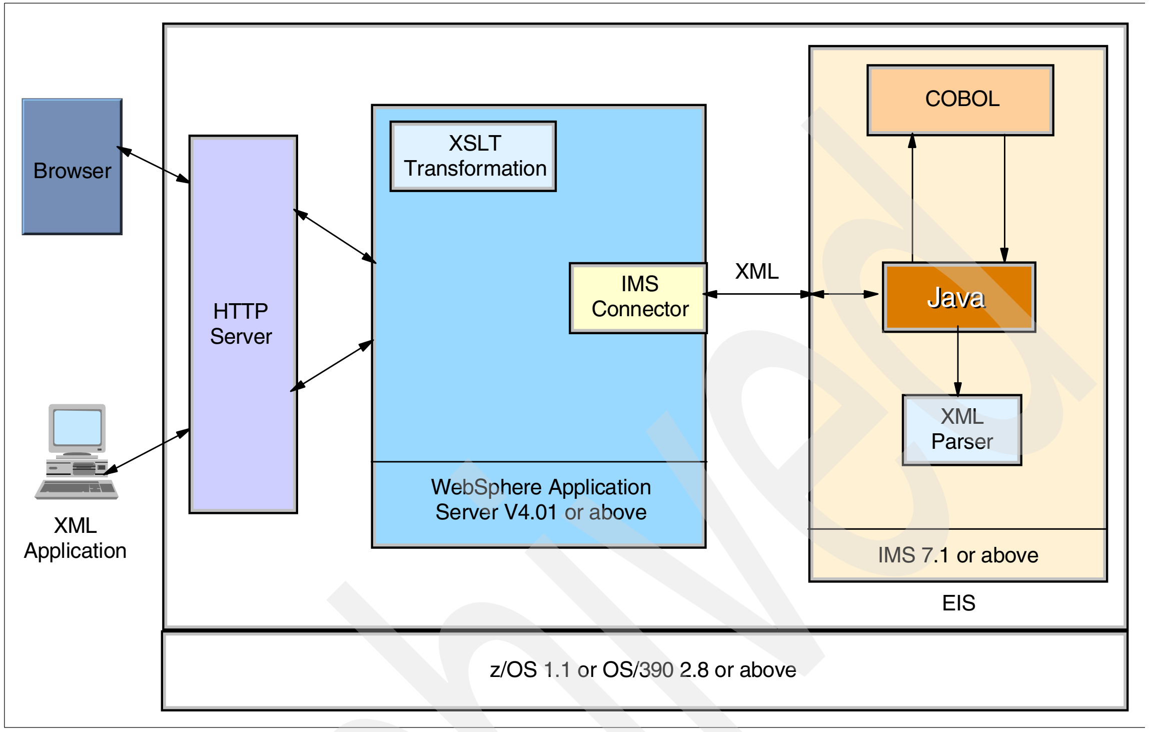


Figure 8-4 IMS Transaction Manager

The important points to note in this option are:

- There is JVM support within IMS 7.1. The Java support allows use of XML4J parser (Xerces-based).
- There is two-way support between Java and COBOL (Version 3.2 announced in September, 2002). That means Java modules can call COBOL modules and COBOL can call Java Class methods. In other words, this provides an opportunity to reuse legacy transactions.

Message flow

A high-level view of the message flow is as follows:

1. A client enters the transaction input data on an HTML page on a Web browser.
2. The transaction input data is transmitted from the Web browser to the Web server on z/OS and WebSphere Application Server invokes a Java servlet.

3. The servlet can invoke the XML parser to validate the XML input, then it invokes IMS connector to the EIS Server. The servlet acts as client to the IMS connector. All this happens under Struts (MVC) framework control.
4. On the host, an IMS Java “driver” program receives the transaction from the IMS Q. It parses the XML data component of the message using XML4J to validate the XML data component, converts the XML document to a data stream, and puts back the converted transaction into an IMS Q for invoking the legacy COBOL transaction.
5. The Java driver program converts the response data stream into an XML document and puts it back onto an IMS Q for IMS Connect to pick up and deliver to the WebSphere platform.
6. The servlet (controller) can either invoke a JSP (View) or deliver the XML output over an enterprise-preferred transport TCP/IP or MQ.

8.2.2 CICS TS

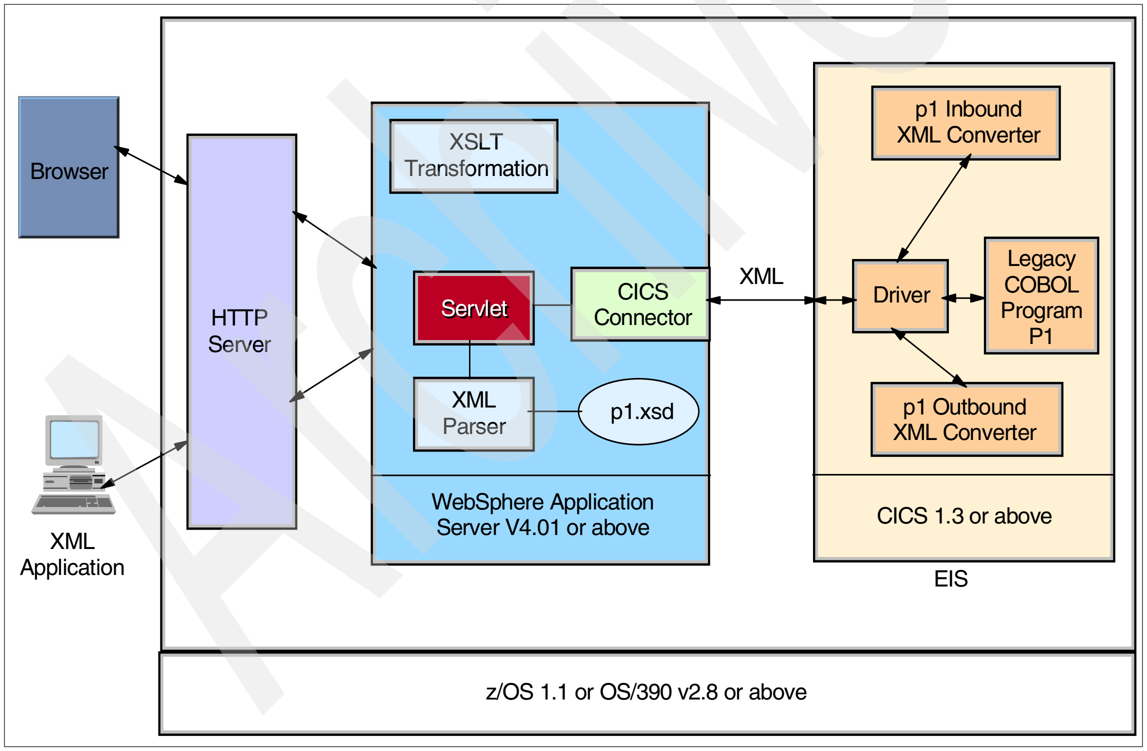


Figure 8-5 CICS TS

The most innovative aspect of this option is the XML Converter component provided with WebSphere Studio Enterprise Developer. From a COBOL program's source code, WSED generates the following components:

1. An input XML converter to map an XML instance document to a data stream based on a COBOL data structure. The XML converter is a generated COBOL program, which can be executed in CICS address space. The tool generates all the appropriate COBOL data types corresponding to all XML element definitions.
2. An output XML converter to map a COBOL data stream, based on a COBOL data structure, to an XML Instance document.
3. A sample driver program that contains guidelines on how to invoke converters and the existing application (p1) in the CICS environment. The programmer needs to modify this sample program to suit his needs.
4. A generated XML Schema (for example, p1.xsd) to validate both the input and output XML message instance.

For details on how to generate the XML Converters, refer to 5.4.2, "XML converters for traditional COBOL programs" on page 121.

Message flow

A high-level view of the message flow is as follows:

1. A client enters the transaction input data on an HTML page on a Web browser.
2. The transaction input data is transmitted from the Web browser to the Web server on OS/390 and WebSphere Application Server invokes the Java servlet.
3. The servlet first invokes the XML parser to validate the XML instance, then it invokes CICS connector to the EIS Server. The Java application or servlet acts as client to the CICS connector. All this happens under Struts (MVC) framework control.
4. On the host, the driver program receives the XML instance, invokes the input converter to generate the COBOL-compliant data stream, and then hands it over to DFHCOMMAREA for processing by the p1 (the legacy COBOL program). When p1 finishes processing, the driver picks up the output from DFHCOMMAREA, invokes the output converter, generates the output XML document instance, and hands it over to the ECI gateway to deliver it to WebSphere.
5. The servlet on WebSphere can optionally invoke the parser to validate the output document using the Schema (p1.xsd) generated by WSED. Depending on the interaction model chosen, the servlet (controller) can either invoke a

JSP (View) or deliver the XML instance over an enterprise-preferred transport TCP/IP or MQ.

Similar model for IMS

The schematic in Figure 8-6 presents an overview of how the model can be adapted for IMS.

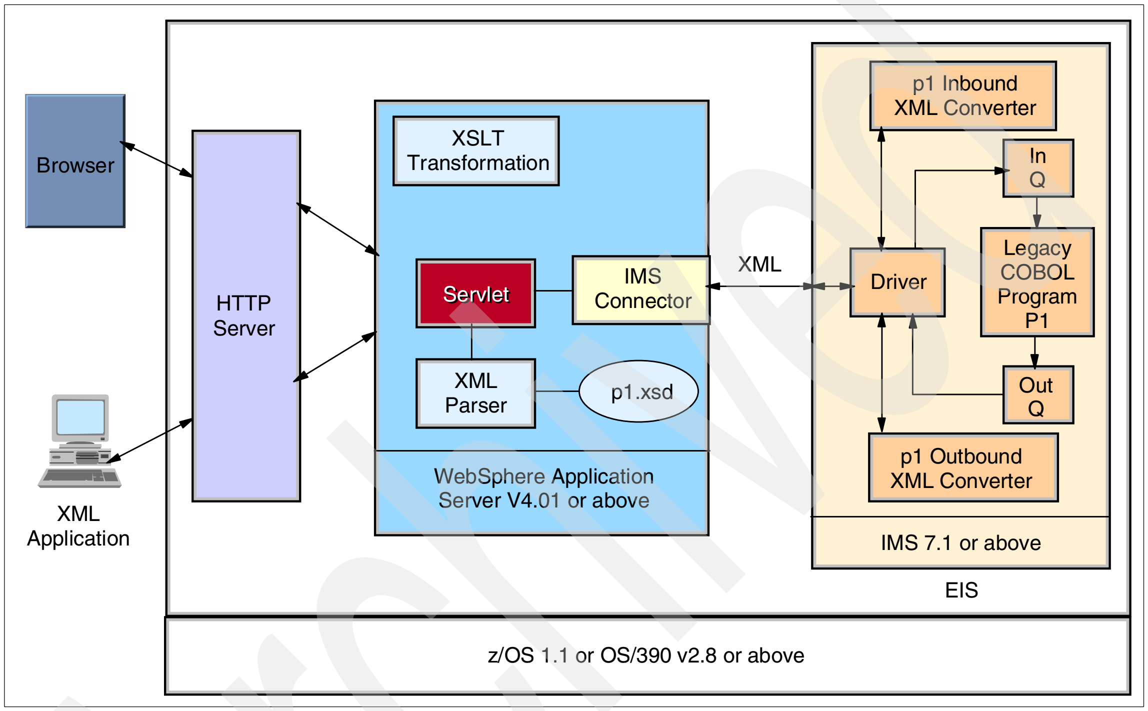


Figure 8-6 IMS Transaction Manager

The only difference in this model is that the *driver*, instead of handing over the input data stream after conversion to DFHCOMMAREA, puts it into an IMS Q. Similarly, the driver picks up the response data stream from an IMS Queue for handing over to the output XML converter. Again the WSED will generate all the appropriate components, such as Driver template, Input/Output XML converters, and so forth.

8.2.3 Web services for DB2

DB2 Version 7.2 supports all the open Internet standards, such as XML, UDDI and SOAP, required by Web Services infrastructure with the DB2 XML Extender. This support is known as the *Web Services Object Runtime Framework (WORF)*. The DB2 XML Extender is an integral feature of DB2 Version 7.

More information on DB2 and Web services is available from the DB2 Developer Domain Web site at:

http://www7b.software.ibm.com/dmdd/zones/webservices/

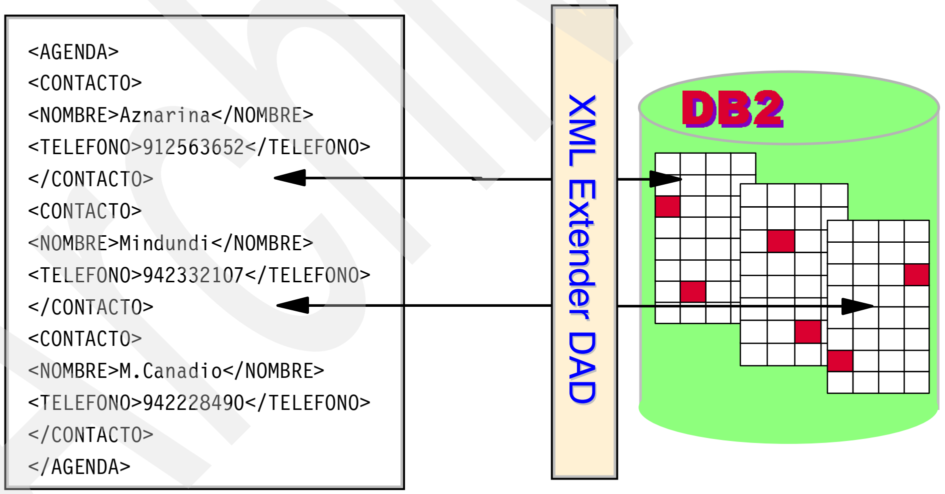
In addition, see the tutorial on Web services with WORF and DB2 extender at:

http://www7b.software.ibm.com/dmdd/library/tutorials/0304balani/index.html

The DB2 XML Extender provides the support for the XML-based operations that enable XML documents to be stored or retrieved from a DB2 database. This support is provided through tools and runtime software. The following types of Web service operations are supported:

- **XML-based query** or storage: An XML document is stored in DB2 relational tables and composed again on retrieval. This method of operation requires the presence of DB2 XML Extender.

  This query allows you to compose XML documents from relational data, or break an XML document down into its component parts and store it in relational tables. This is supported partly by XML Extender and partly by special stored procedures shipped with DB2 XML Extender. This operation is illustrated in Figure 8-7.



```
<AGENDA>
<CONTACTO>
<NOMBRE>Aznarina</NOMBRE>
<TELEFONO>912563652</TELEFONO>
</CONTACTO>
<CONTACTO>
<NOMBRE>Mindundi</NOMBRE>
<TELEFONO>942332107</TELEFONO>
</CONTACTO>
<CONTACTO>
<NOMBRE>M.Canadio</NOMBRE>
<TELEFONO>942228490</TELEFONO>
</CONTACTO>
</AGENDA>
```

Figure 8-7 XML based query

How does DB2 know the correct mapping and the attributes to build the XML document from DB2 data?

One of the inputs into both storage and retrieval is the user-specified mapping file that creates the association between relational data and XML document structure. This mapping file is called a *document access definition (DAD)*. It provides a way that you can create an XML document with the XML elements and attributes named as you please and with the shape that you want. The focus of this approach is on moving and manipulating XML documents.

- **SQL-based query -** This is simply the ability to send SQL statements, including stored procedure calls, to DB2 and to return results with a default tagging.

  The focus of this approach is actually getting the data in and out of the database, not on shaping the results in a particular way. SQL-based query does not require the use of DB2 XML Extender because there is no use-defined mapping of SQL data to XML elements and attributes. Instead, the data is returned using only a simple mapping of SQL data types, using column names as elements.

Both the XML-based and the SQL-based forms of querying are controlled by a file called a *document access definition extension* (DADX). The DADX defines the operations that can be performed by the Web services. In other words, it defines what the XML extender DAD document and SQL-based stored procedures will provide from the DB2 database. In Example 8-1 we show a sample of a DADX file.

Example 8-1 A DADX file

```
<?xml version="1.0"?>
<DADX xmlns="urn:ibm:.com:dadx"....>
<SQL_call>CALL Myproc(:query1, :query2, :query3)</SQL_call>
<SQL_query>select * from order_tab .... </SQL_query>
<SQL_update>insert into order_tab ... </SQL_update>
......
<retrieveXML>
   <DAD_ref>getstart_xcollection.dad</DAD_ref>
   <SQL_override>
   select o.order_key, customer_name,
   customer_email...
   .....
   </SQL_override>
</retrieveXML>
```

You can write DADX files using any text editor. After you create a DADX file, the DB2 Web services samples provide support to automatically generate WSDL files, including support for UDDI Best Practices. We used an internal version of the WORF sample, modified for z/OS. For more information on the WORF sample, see the Web Services Object Runtime Framework for DB2 download Web page at:

http://www7b.software.ibm.com/dmdd/zones/webservices/worf/

In addition, with WebSphere Studio Application developer, support is provided to generate the deployment descriptor needed to deploy the Web service into WebSphere, and to generate a documentation and test page, which you can use as a basis for building the client part of your Web application.

WORF uses SOAP 2.2 and DADX. A DADX document specifies a Web Service using a set of operations that are defined by XML Extender DAD documents or SQL statements. Web services specified in a DADX file are called DADX Web services.

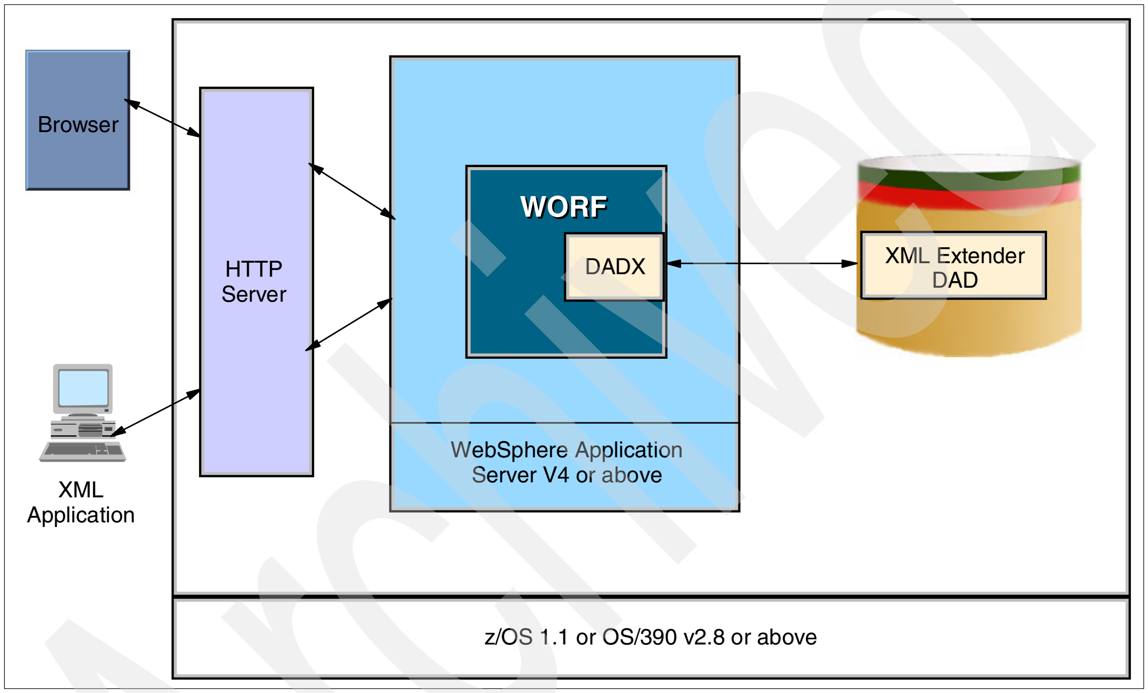

Browser
HTTP Server
WORF
DADX
XML Extender DAD
WebSphere Application Server V4 or above
XML Application
z/OS 1.1 or OS/390 v2.8 or above

Figure 8-8 Web Services Object Runtime Framework

The framework provides the following features:

- Resource-based deployment and invocation
- Automatic service redeployment, at development time, when defining resource changes
- HTTP GET and POST bindings, in addition to SOAP
- Automatic WSDL and XSD generation, including support for UDDI best practices
- Automatic documentation and test page generation

Figure 8-8 provides an overview of the WORF framework. WORF receives an HTTP SOAP GET or POST service request. The URL of the request specifies a DADX or DTD file, and the requested action, which can be a DADX operation or a command, such as TEST, WSDL, or XSD. A DADX operation can also contain input parameters. In response to a Web service request, WORF loads the DADX file specified in the request, and generates a response based on the request, as follows:

- For operations: Loads a DAD file, if requested. It replaces parameters with requested values; connects to DB2 and runs any SQL statements, including SQL calls; and finally, formats the result into XML, converting types as necessary.
- For commands: Generates necessary files, test pages, or other responses required, and returns the response to the service requestor.

System requirements

- IBM DB2 Universal Database™ Version 7, with the fix to APAR PQ56655 installed (PTF UQ65774).
- IBM DB2 XML Extender Version 7 is required for store and retrieve operations.
- Java JDK Version 1.3.1.
- WebSphere Application Server Advanced Edition Version 4.01 with maintenance level W401400.

The Web Services WORF sample

The process to install the sample Web Services application using WORF is similar to other J2EE applications, as described in “Application deployment” on page 142.

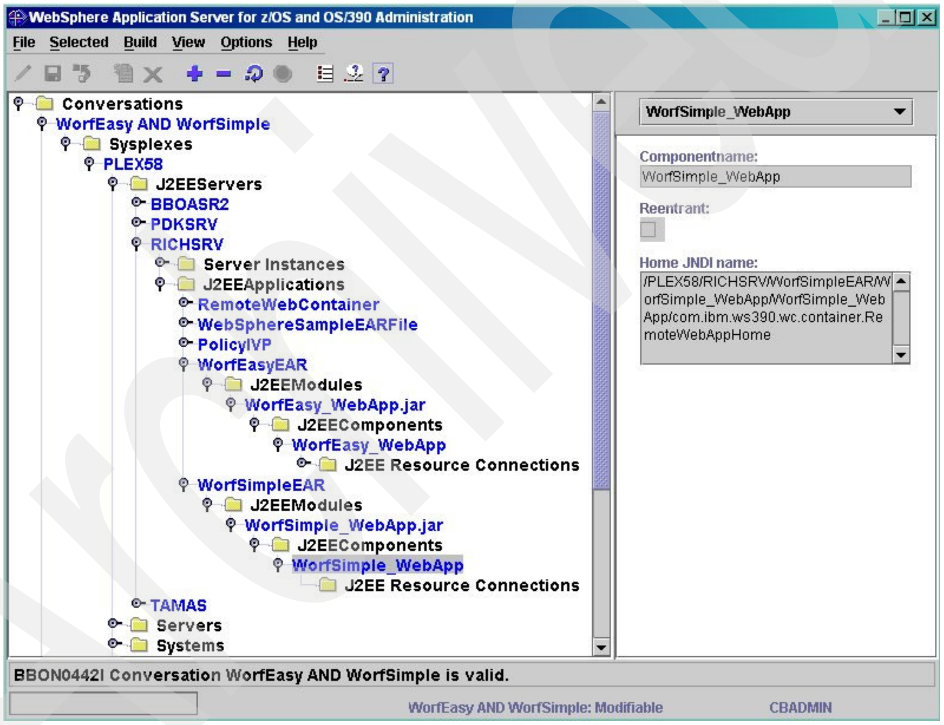
Note: If your WebSphere server does not contain Service Level 4 (PTFs UQ90051 and UQ90052) you still need to go through the Application Assembly Tool (AAT) to assemble and validate the application.

If WebSphere service level 4 is installed on your system, you can process any ear file created from WebSphere Studio Application Development directly into a SMEUI conversation. The Administration and Operations applications automatically run the 390fy program to resolve your ear files, and you have no need to run the 390fy command.

However, if you deploy an application through some other method, you must run the 390fy command to resolve your ear files for use on z/OS and OS/390. The same 390fy command ships with both the Administration and Operations applications and the WebSphere Application Server for z/OS and OS/390 runtime.

How you build your WORF Web Services sample application will determine whether you have to run the 390dfy command. Once you have built and assembled a WORF Web Services ear file, you need to deploy it on z/OS using a SMEUI conversation. Figure 8-9 illustrates the deployment of a WorfSimpleEAR.ear onto a WebSphere server named RICHSRV.

As discussed in Chapter 6, “WebSphere Application Server on z/OS and OS/390” on page 135, you should not rely on the level of SOAP and XML parser jar files provided with WebSphere Application Server; it is recommended that you include the level needed by your application.



Componentname: WorfSimple_WebApp

Reentrant: ☐

Home JNDI name: /PLEX58/RICHSRV/WorfSimpleEAR/WorfSimple_WebApp/WorfSimple_WebApp/com.ibm.ws390.wc.container.RemoteWebAppHome

BBON0442I Conversation WorfEasy AND WorfSimple is valid.

WorfEasy AND WorfSimple: Modifiable CBADMIN

Figure 8-9 Deploying WORF Web Services samples using a SMEUI conversation

You have a choice of where you want to place the SOAP and XML parser jar files.

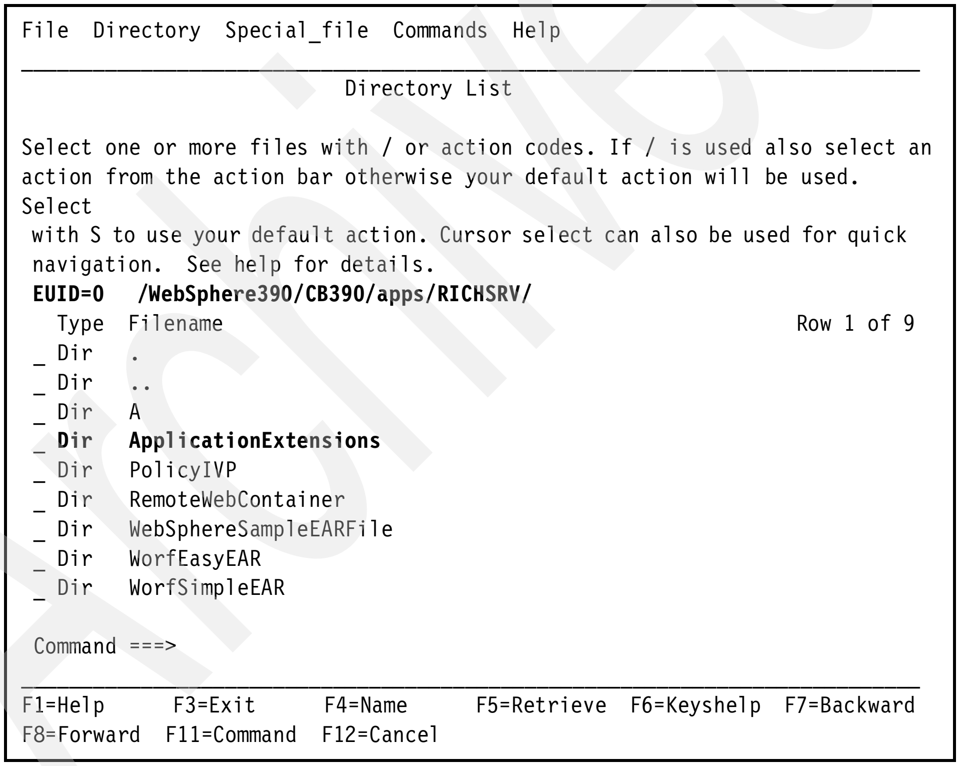
If included at development time, the jar files will be placed into the WEB-INF/lib and will be specific to the Web application you deploy. In this case the files will be uploaded by SMEUI to the WebSphere server on z/OS as part of the deployment process. If you have multiple Web services applications that use the same set of

SOAP and XML parser jar files, this is obviously not a recommended solution, since it will make application maintenance extremely difficult to manage.

You may decide to consolidate one single copy of the jar files to a WebSphere application directory and make these files accessible to all your Web services applications. In our case, we used ftp to place the common jar files into an application directory pointed to by the `APP_EXT_DIR` variable in the WebSphere server and, to indicate that we were using application mode, in the server `jvm.properties` file we specified:

```
com.ibm.ws390.server.classloadermode=2
com.ibm.ws.classloader.ejbDelegationMode=false
```

Now all Web services applications running in that server consistently refer to a common set of SOAP, WORF, and XML parser jar files.



```
File  Directory  Special_file  Commands  Help
___________________________________________________________________________
                           Directory List

Select one or more files with / or action codes. If / is used also select an
action from the action bar otherwise your default action will be used.
Select
 with S to use your default action. Cursor select can also be used for quick
 navigation.  See help for details.
 EUID=0   /WebSphere390/CB390/apps/RICHSRV/
   Type  Filename                                                  Row 1 of 9
 _ Dir   .
 _ Dir   ..
 _ Dir   A
 _ Dir   ApplicationExtensions
 _ Dir   PolicyIVP
 _ Dir   RemoteWebContainer
 _ Dir   WebSphereSampleEARFile
 _ Dir   WorfEasyEAR
 _ Dir   WorfSimpleEAR

 Command ===>
___________________________________________________________________________
F1=Help      F3=Exit      F4=Name      F5=Retrieve  F6=Keyshelp  F7=Backward
F8=Forward  F11=Command  F12=Cancel
```

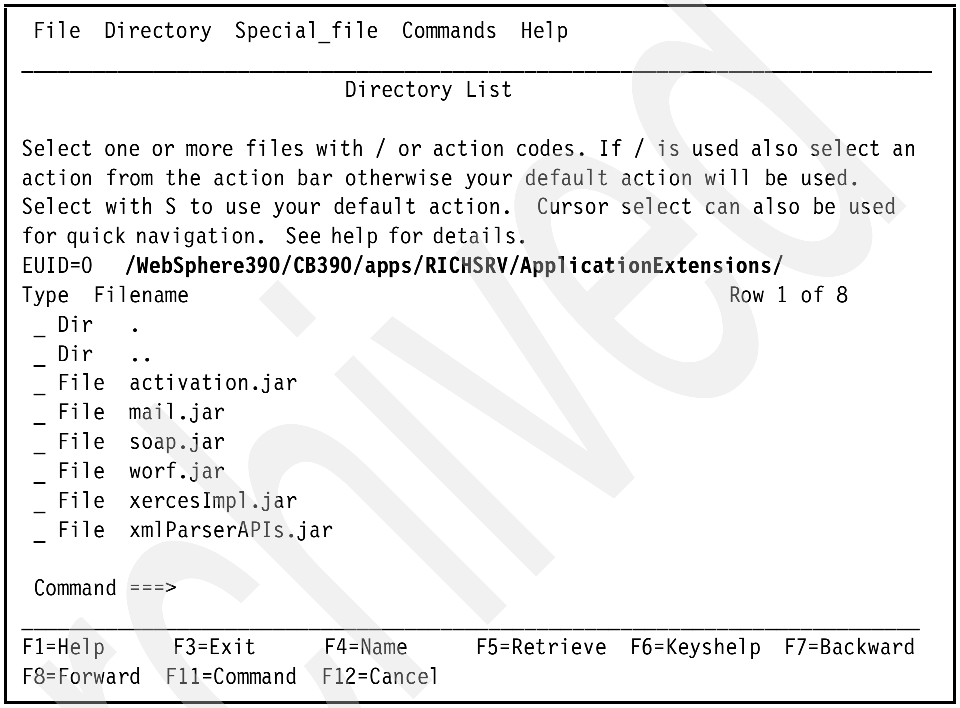
Figure 8-10   ApplicationExtensions directory specified in APP_EXT_DIR

For the Web services sample, we placed the following files in the server application directory:

– activation.jar

– mail.jar
– soap.jar
– worf.jar
– xercesImpl.jar
– xmlParserAPIs.jar

This is illustrated in Figure 8-11.



```
 File  Directory  Special_file  Commands  Help
______________________________________________________________________________
                           Directory List

Select one or more files with / or action codes. If / is used also select an
action from the action bar otherwise your default action will be used.
Select with S to use your default action.  Cursor select can also be used
for quick navigation.  See help for details.
EUID=0   /WebSphere390/CB390/apps/RICHSRV/ApplicationExtensions/
Type  Filename                                            Row 1 of 8
 _ Dir   .
 _ Dir   ..
 _ File  activation.jar
 _ File  mail.jar
 _ File  soap.jar
 _ File  worf.jar
 _ File  xercesImpl.jar
 _ File  xmlParserAPIs.jar

 Command ===>
_____________________________________________________________________________
F1=Help      F3=Exit      F4=Name      F5=Retrieve  F6=Keyshelp  F7=Backward
F8=Forward  F11=Command  F12=Cancel
```

Figure 8-11 JAR files in ApplicationExtensions directory

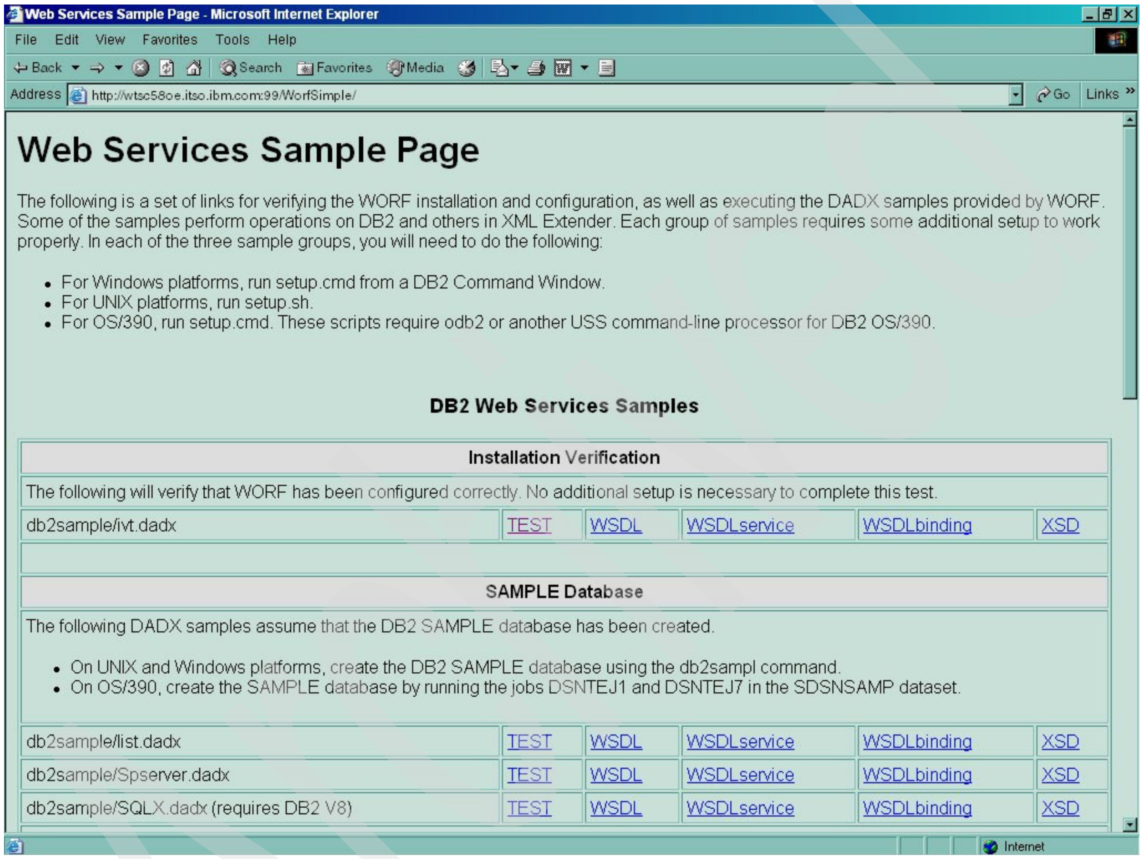
Note that the jar files associated with SOAP and XML parsers, in this example soap.jar, xercesImpl.jar, and xmlParserAPIs.jar, must be consistent and placed in the same directory. The classloader would detect a violation if there is an inconsistency between levels of SOAP and XML parser, or if you have SOAP and the XML parser in different directories.

Note that if you misplace any of the SOAP or XML parser jar files, or inadvertently load inconsistent levels, at runtime you will get an error message from the classloader, such as:

```
Error Message: Class org/xml/sax/InputSource violates loader constraints:
parent and child already loaded different classes
```

```
Error Message: Class org/xml/sax/InputSource violates loader constraints:
definition mismatch between parent and child loaders
```

Once the WORF Web services sample has been deployed on your WebSphere Application Server on z/OS, the initial Web Services Sample Page can be displayed, as shown on Figure 8-12.



Web Services Sample Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wtsc58oe.itso.ibm.com:99/WorfSimple/
Go Links
Internet

Web Services Sample Page

The following is a set of links for verifying the WORF installation and configuration, as well as executing the DADX samples provided by WORF. Some of the samples perform operations on DB2 and others in XML Extender. Each group of samples requires some additional setup to work properly. In each of the three sample groups, you will need to do the following:

- For Windows platforms, run setup.cmd from a DB2 Command Window.
- For UNIX platforms, run setup.sh.
- For OS/390, run setup.cmd. These scripts require odb2 or another USS command-line processor for DB2 OS/390.

DB2 Web Services Samples

Installation Verification

The following will verify that WORF has been configured correctly. No additional setup is necessary to complete this test.

db2sample/ivt.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD

SAMPLE Database

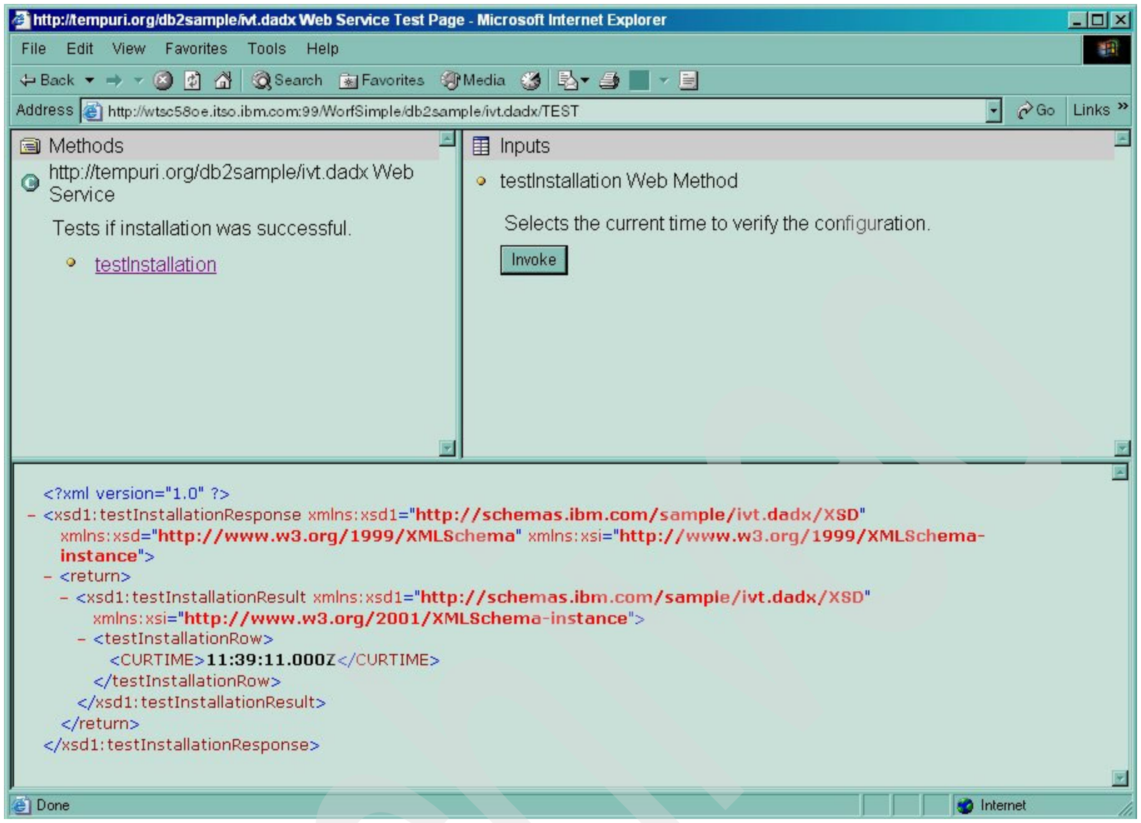
The following DADX samples assume that the DB2 SAMPLE database has been created.

- On UNIX and Windows platforms, create the DB2 SAMPLE database using the db2sampl command.
- On OS/390, create the SAMPLE database by running the jobs DSNTEJ1 and DSNTEJ7 in the SDSNSAMP dataset.

db2sample/list.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/Spserver.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/SQLX.dadx (requires DB2 V8)	TEST	WSDL	WSDLservice	WSDLbinding	XSD

Figure 8-12 WORF Sample index page

We also ran the installation verification provided with the sample to verify the deployment. This part requires no additional installation or customization. It displays the time as shown in Figure 8-13.


http://tempuri.org/db2sample/ivt.dadx Web Service Test Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://wtsc58oe.itso.ibm.com:99/WorfSimple/db2sample/ivt.dadx/TEST
Go Links
Methods
http://tempuri.org/db2sample/ivt.dadx Web Service
Tests if installation was successful.
testInstallation
Inputs
testInstallation Web Method
Selects the current time to verify the configuration.
Invoke
Done
Internet

```
  <?xml version="1.0" ?>
- <xsd1:testInstallationResponse xmlns:xsd1="http://schemas.ibm.com/sample/ivt.dadx/XSD"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema" xmlns:xsi="http://www.w3.org/1999/XMLSchema-
    instance">
  - <return>
    - <xsd1:testInstallationResult xmlns:xsd1="http://schemas.ibm.com/sample/ivt.dadx/XSD"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      - <testInstallationRow>
          <CURTIME>11:39:11.000Z</CURTIME>
        </testInstallationRow>
      </xsd1:testInstallationResult>
    </return>
  </xsd1:testInstallationResponse>
```

Figure 8-13 WORF sample, installation verification

9

JCA and WebSphere connectors

This chapter provides an overview of J2EE Connector Architecture (JCA) and the WebSphere connector facility.

9.1 J2EE Connector Architecture overview

The J2EE Connector Architecture specifies a standard way to access back-end enterprise systems in a J2EE environment. In JCA, the application server communicates with the back-end enterprise system through a resource adapter or connecter. A resource adapter is specific to the back-end enterprise system and can be plugged into any application server.

The JCA addresses the key issues and requirements of EIS integration by defining a set of scalable, secure, and transactional mechanisms that enable the integration of EIS with application servers and enterprise applications.

An application server and resource adapter (and its underlying EIS) collaborate to keep all system-level mechanisms—transaction, security, and connection pooling—transparent from the application. As a result, an application developer can focus on the development of business and presentation logic for the application components and does not need to get involved in the system-level issues related to EIS integration.

To accomplish this goal, the connector architecture (illustrated in Figure 9-1) defines two types of contracts:

- A system-level contract between an application server and a resource adapter
- An application contract between an application component and a resource adapter

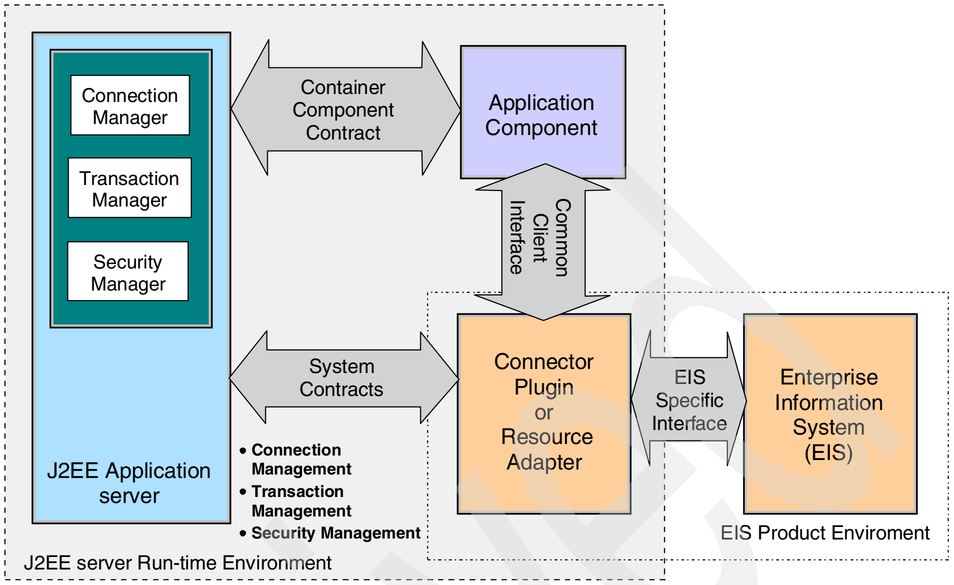


Figure 9-1 Contracts

9.1.1 System-level contracts

The connector architecture's system-level contracts define a "pluggability" standard between application servers and EIS. By adhering to the terms of these contracts when developing their components, an application server and an EIS know that connecting the two platforms will be a straightforward operation of plugging in the resource adapter.

A resource adapter is a system library that is specific to the EIS and designed to provide connectivity to the EIS. The resource adapter is the component that plugs in to an application server. A resource adapter is a library used within the address space of the application server. It abstracts the details of both the interface and communication between the underlying resource adapter library and the EIS. Typically, the EIS and the resource adapter communicate over some EIS-specific protocol. A resource adapter can also use a native library as part of its implementation.

In JCA 1.0 specification, the following three contracts are defined:

- Connection management contract: This contract enables an application server to pool connections to an underlying EIS, while at the same time it enables application components to connect to an EIS. Pooling connections is important to create a scalable application environment, particularly when large number of clients require access to the underlying EIS.

- Transaction management contract: This contract is between the transaction manager that is provided with the application server and an EIS that supports transactions. It gives an application server's transaction manager the ability to manage transactions across multiple EIS resource managers.
- Security contract: The security contract enables secure access to an EIS. It provides support for a secure application environment and protects the EIS-managed resources.

The JCA 2.0 specification will cover thread management and asynchronous communication with EIS.

9.1.2 Application contract

This contract defines the client API that an application component uses for EIS access. The client API may be the Common Client Interface (CCI) or it may be an API specific to the particular type of resource adapter and the underlying EIS. The CCI defines a common client API for accessing multiple heterogeneous EIS. It is well suited for enterprise application integration (EAI).

For a more detail treatment of JCA, refer to *J2EE Connector Architecture and Enterprise Application Integration,* by Rahul Sharma, Beth Stearns, and Tony Ng.

9.2 WebSphere connectors

WebSphere for z/OS supports JCA as well as J2EE 1.2 standards. WebSphere for z/OS supports the following CICS or IMS resource adaptors:

- CICS Transaction Gateway External Call Interface (ECI) Connector
- IMS Connector for Java 1.2.2
- IMS JDBC Connector

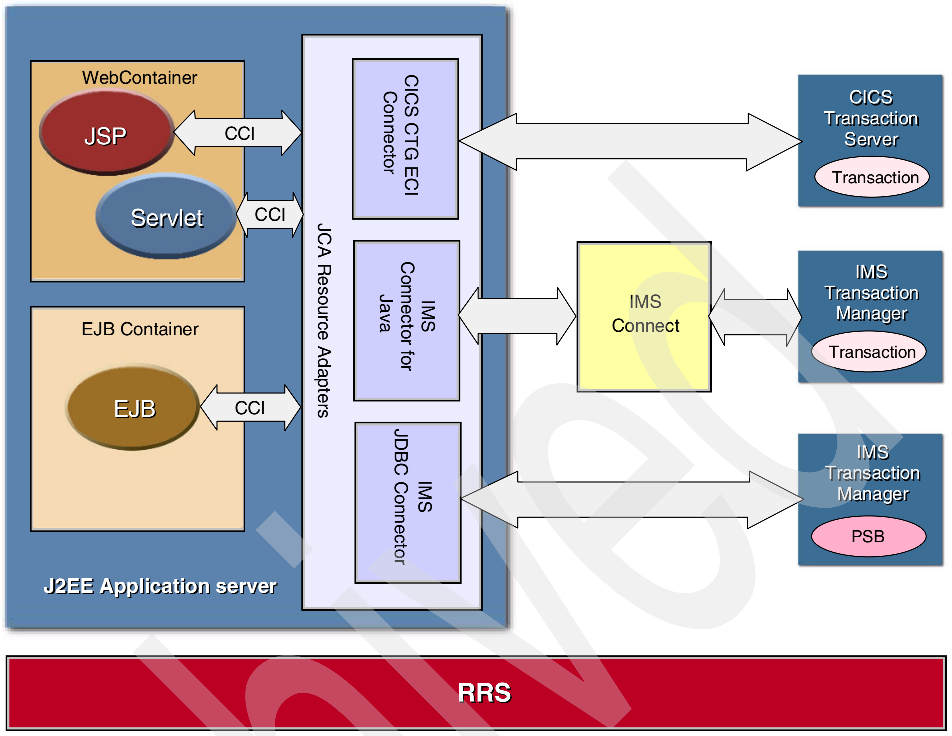


Figure 9-2 JCA connectors

These resource adapters or connectors not only implement the J2EE connector interface, but also are RRS-compliant. They are designed specifically to work with the resource recovery services (RRS) component of z/OS or OS/390 (V2.8 or above). Resource recovery consists of the protocols and program interfaces that allow WebSphere for z/OS, the RRS component of z/OS, and CICS or IMS to work together to make consistent changes to multiple protected resources.

Because of their design, WebSphere for z/OS, the RRS component of z/OS, CICS or IMS subsystems, and these RRS-compliant connectors can participate in two-phase commit processing, which enables z/OS or OS/390 to restore critical resources to their original state if they become corrupted because of a hardware or software failure, human error, or a catastrophe. These J2EE connectors are shipped as part of separate CICS or IMS products.

For its supported connectors, WebSphere for z/OS also provides additional advantages, specifically:

- The ability for system administrators to define connection management at a sysplex level. Connection management support is a configuration extension available through the WebSphere for z/OS Administration application (SMEUI).

- The ability for application assemblers to specify:
  - Connection management policy, which is a quality of service issue for applications using connectors. This ability allows finer control of the management of valuable back-end resources, which is especially useful to prevent a misbehaving application from tying up system-wide resources, thereby making the system unusable.
  - Resource authentication for applications using connectors. This ability determines which user identities WebSphere for z/OS will pass to back-end products (such as CICS and IMS) through connectors.

Connection management policies and resource authorization are set through the WebSphere for z/OS Application Assembly tool.

These configuration and application extensions are functions that WebSphere for z/OS provides in addition to the implementation of the J2EE interfaces. Use of these extensions does not cause any loss of function provided for J2EE compliance at the current level.

WebSphere for z/OS also extends its connection management capabilities to its JDBC resources, so J2EE application components that use JDBC to access DB2 also benefit from additional qualities of service.

As specified by the J2EE Connector Architecture, the application server is responsible for managing physical connections, or *managed connections*, to a back-end resource, and for providing qualities of service related to the use of connectors. These qualities of service include connection pooling, transaction management, and security management. In a WebSphere for z/OS J2EE server configuration, these Managed Connections are known as J2EE resources.

9.3 Transaction management

WebSphere for z/OS supports only two types of connectors: non-transactional and RRS-transactional. Connector transaction processing varies for each type.

Restriction: WebSphere for z/OS V4 does not support XA transaction or local transaction support defined by the J2EE Connector Architecture.

9.3.1 RRS-transactional

This type of connector is configured to work with the resource recovery service component of z/OS to participate in two-phase commit processing. For RRS-transactional connectors, the type of transaction processing performed is determined at the time an interaction is executed on a connection to send a request to the target Enterprise Information System (EIS). There are two ways a given interaction may be handled:

- If processing under the current thread is running under a global transaction, WebSphere for z/OS propagates the current transaction context across the interface to the back-end EIS, and two-phase commit processing or rollback processing of the transaction is coordinated using z/OS resource recovery services (RRS).
- If processing under the current thread is not running under a global transaction, WebSphere for z/OS sends the request to the back-end EIS, indicating that processing performed for the request should be committed before returning (this type of processing is known as sync-on-return).

Usually, the legacy application is wrapped in a stateless session EJB. The transaction policy of the EJB dictates whether or not processing is running under a global transaction. If the transaction policy dictates processing under a global transaction, then any connector processing will also do global transaction processing. Similarly, if the transaction policy dictates processing without a transaction, then any connector processing will be performed as a sync-on-return request.

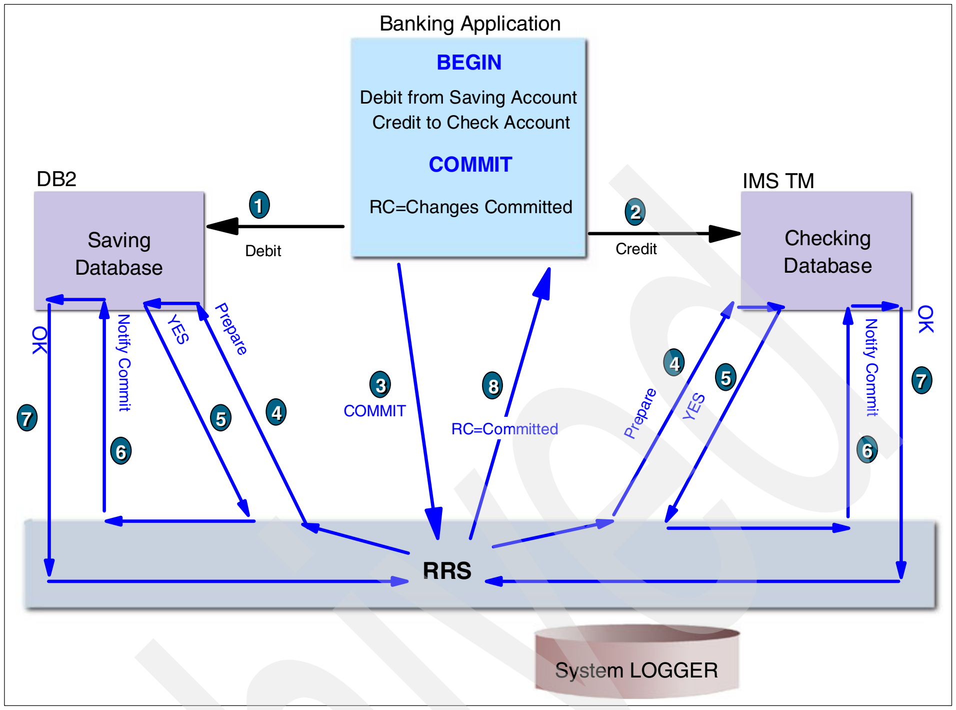


Figure 9-3

Figure 9-3 shows an example of a global transaction, where two atomic transactions on two different Resource Managers are being coordinated by RRS. As is apparent from this scenario, global transactions are more expensive in terms of response time.

9.3.2 Non-transactional

In the case of a connector that has been configured as a non-transactional connector, all requests to the back-end EIS (for example, a CICS or IMS subsystem) are performed as sync-on-return requests. In other words, any changes made by the EIS are committed by the time control is returned to the EJB that made the request. Sync-on-return processing is performed regardless of the transaction policy specified by the EJB.

Application model

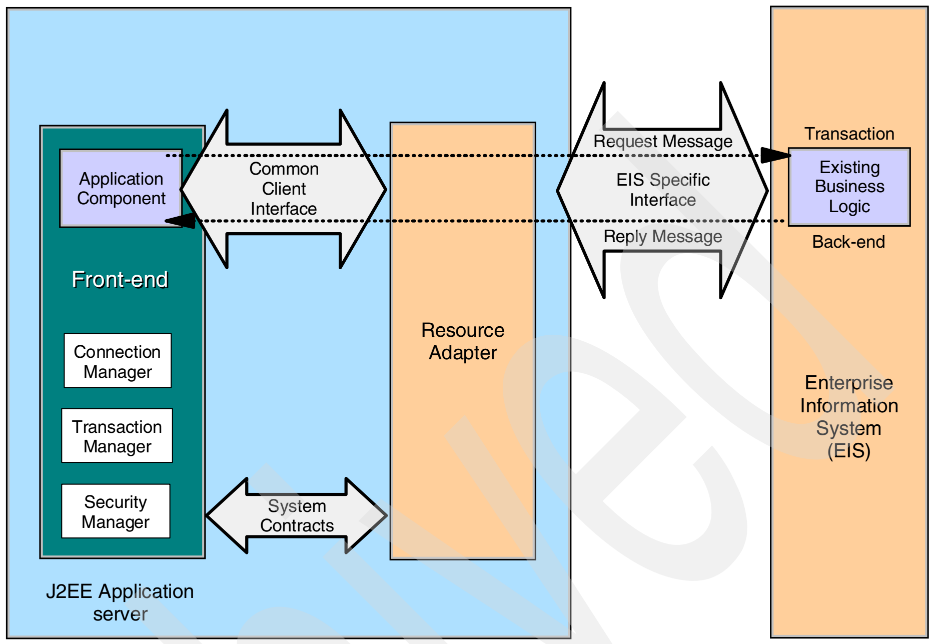


Figure 9-4 Application-to-application interaction

Figure 9-4 illustrates an application-to-application interaction. From the developer's point of view, the complexities of the System Contract and RRS co-ordination for two phase commits are totally transparent, these features being container managed. Application developers are provided with a set of APIs to utilize the services provided by the WebSphere connector facility; thus, they can control the functionality they require by setting appropriate values in the parameters of these APIs.

10

Some key design guidelines

In this chapter we describe some important design concepts that can enable architects and designers to build SOA-based solutions more rapidly and robustly. These guidelines are based on practical experience implementing e-business solutions and some of the new technologies that we used in the lab.

Some of the important aspects of SOA-based application design are:

- Patterns for e-business
- XML-based message design
- Principles of “Design by Contract” and “Service Design”

10.1 Patterns for e-business

Patterns for e-business enable architects and designers to implement successful e-business solutions through the re-use of assets from proven successful experience. The patterns for e-business are based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in such a way that each level of detail builds on the previous one. These assets include the following:

- **Business patterns** identify the interaction between users, businesses, and data. Business patterns are used to create simple, end-to-end e-business applications.
- **Integration patterns** connect business patterns together to create applications with advanced functionality. Integration patterns are used to combine business patterns in advanced e-business applications.
- **Composite patterns** are combinations of business patterns and integration patterns that have then become commonly used types of e-business applications. Composite patterns are advanced e-business applications.
- **Custom designs** are similar to composite patterns, in that they combine business patterns and integration patterns to form an advanced, end-to-end solution. These solutions, however, have not been implemented to the extent of composite patterns, but are instead developed to solve the e-business problems of one specific company, or perhaps several enterprises with similar problems.
- **Application and Runtime patterns** are driven by the customer's requirements and describe the shape of applications and the supporting runtime needed to build the e-business application.
- **Product mappings** are the mechanisms for populating the solution. Product mappings are based on proven implementations.
- **Guidelines** define best practices for the design, development, deployment, and management of e-business applications.

These assets and their relationships to each other are shown in Figure 10-1.

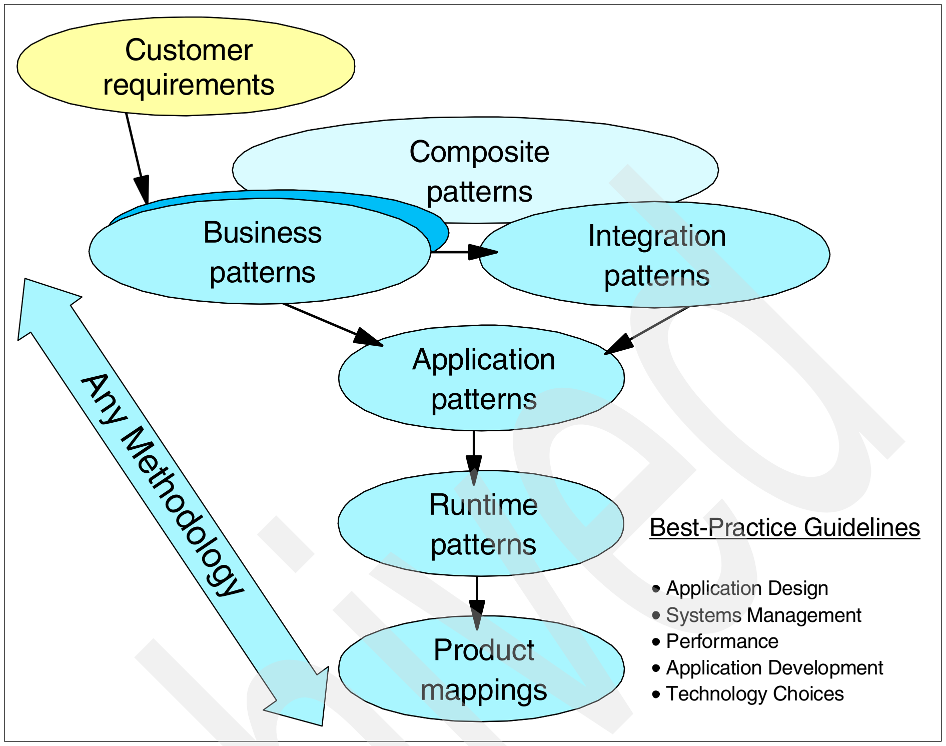


Figure 10-1 Patterns for e-business assets

The IBM patterns for e-business help facilitate the reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven. The information captured by the patterns is applicable to the majority of business situations.

Many methodologies for developing applications exist; however, most are merely variations of one of the two generic methodologies:

- Structured Analysis/Design Methodology: Data-flow, process/data centric emphasis.
- Object-Oriented Analysis and Design Methodology: This methodology encompasses a vast number of variations; UML (Unified Modeling Language) is emerging as the most widely accepted technique.

Figure 10-2 presents a classification of the patterns from a development life cycle perspective.

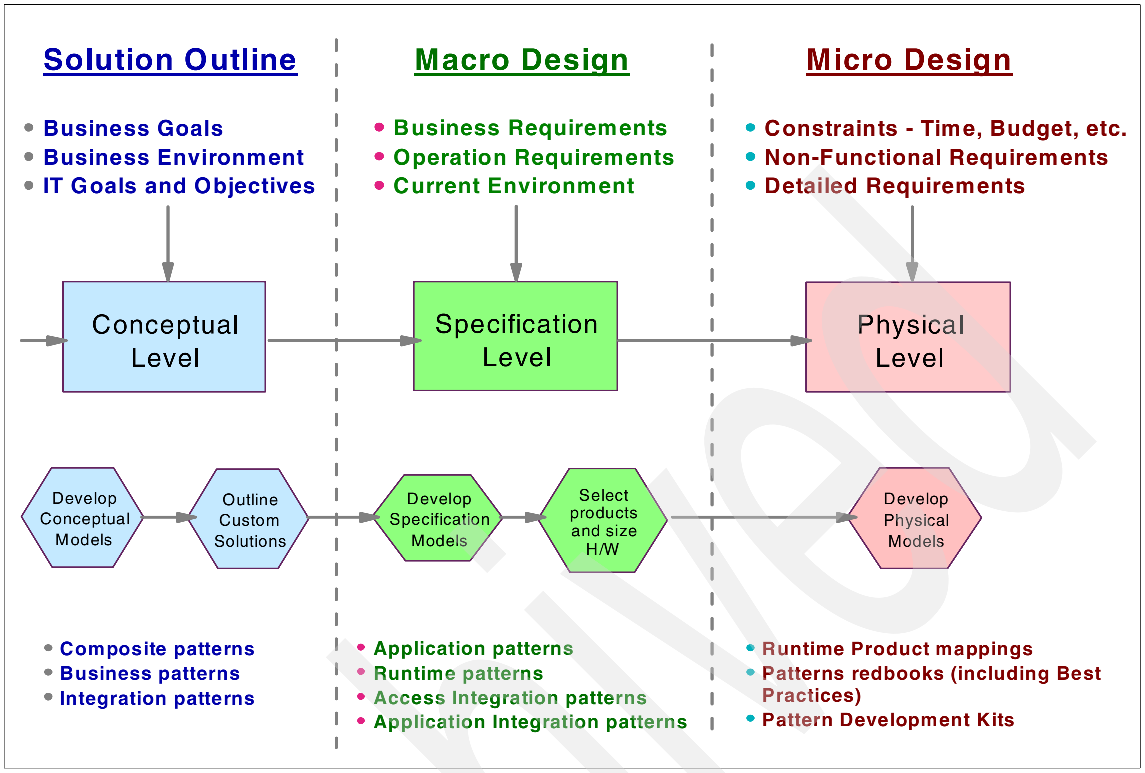


Figure 10-2 Patterns for e-business and solutions methodologies

The patterns in different phases of development life cycle help architects and designers to quickly articulate a solution model. Patterns for e-business are distilled from a rich set of experiences of many architects and designers around the world.

The patterns can be effectively used to create solutions quickly, whether for a small local business or a large multinational enterprise. As shown in the previous figure, customer requirements are quickly translated through the different levels of pattern assets to identify a final solution design and product mapping appropriate for the application being developed. Think in terms of patterns, build on the shoulders of giants!

There are several detailed treatments of how to use patterns in developing solutions. Among them are:

- The patterns discussion on the IBM Web site at:

  `http://www.ibm.com/developerworks/patterns/`

- *Patterns for e-business: A Strategy for Reuse*, Jonathan Adams, et. al.

- *Patterns on z/OS: Connecting Self Service Applications to the Enterprise*, SG24-6827

10.2 XML-based message design

From a user perspective, a *service* represents a free-standing and functionally cohesive module. And the way to look at a *Service interface* is from a data perspective. One of the recommended ways to present this service interface is through a data-centric XML document instance.

A service consists of a pair of messages, a request and a response, and this pair can be represented as a pair of XML instance documents. Once a binary message data stream is represented as an XML document instance, it can be delivered to any other disparate platform that can handle an XML document. And there is hardly any platform that can't handle XML documents.

All these concepts are well understood and this is what is achieved through a Web services model. However, the point we are making here is that, while the components of Web services (SOAP, WSDL, and so forth) are well understood, and great tools and techniques have been developed, not much attention has been paid to how to design the "Business Data" component (or as the cliché would have it, the "payload") of the message. In a data-exchange situation (using messages), the predominant tendency in the industry is to use XML as a simple data encoding mechanism, meaning one that merely assigns a pair of start and end tags to every data element of a message, mimicking an underlying COBOL or any other source system data structure.

This is a very narrow view of the capability of XML. Data-centric XML instances based on XML Schema can be as powerful as Relational Schema-based views of data. If data-centric XML is going to form the basis of e-business applications, organizations need to put some thought and effort into designing proper data-centric XML-based service interfaces. In the next section we present a three-tier architecture for modelling an XML-based service interface.

10.2.1 Architecture for XML messages

Before we describe the architecture for XML messages, we introduce the concept of a *Vocabulary* (or *Vocab*).

A vocabulary is a list of terms used in communication. The terms can be simple elements, like Name, or complex terms composed of simple elements and (optionally) other complex elements, for example, Address.

In the context of enterprise services, a vocabulary usually represents terms used in a particular business domain, hence it should capture at least some of the constraints the underlying corporate data model has implemented, meaning the EIS systems over a period of time. Some of these constraints might be data type, size of a character element, and valid values of a term or enumeration. In case of a complex term or aggregate, there might as well be cardinality constraints, for example: optional, one or more, zero or more, attributes like ID, IDRef, and so forth.

The content of a message (that is, a service interface) should belong to a business domain vocabulary, for example, Customer. Each of these domains will be a Namespace. And an XML instance document can refer to elements in multiple Namespaces.

A domain-specific vocab is similar to a *conceptual schema* in traditional information management parlance. One of the fundamental drivers behind creation of a vocab is to institute a “data discipline” into XML-based message definitions, a discipline akin to what we find in database design parlance. A conceptual view of the process is shown in Figure 10-3.

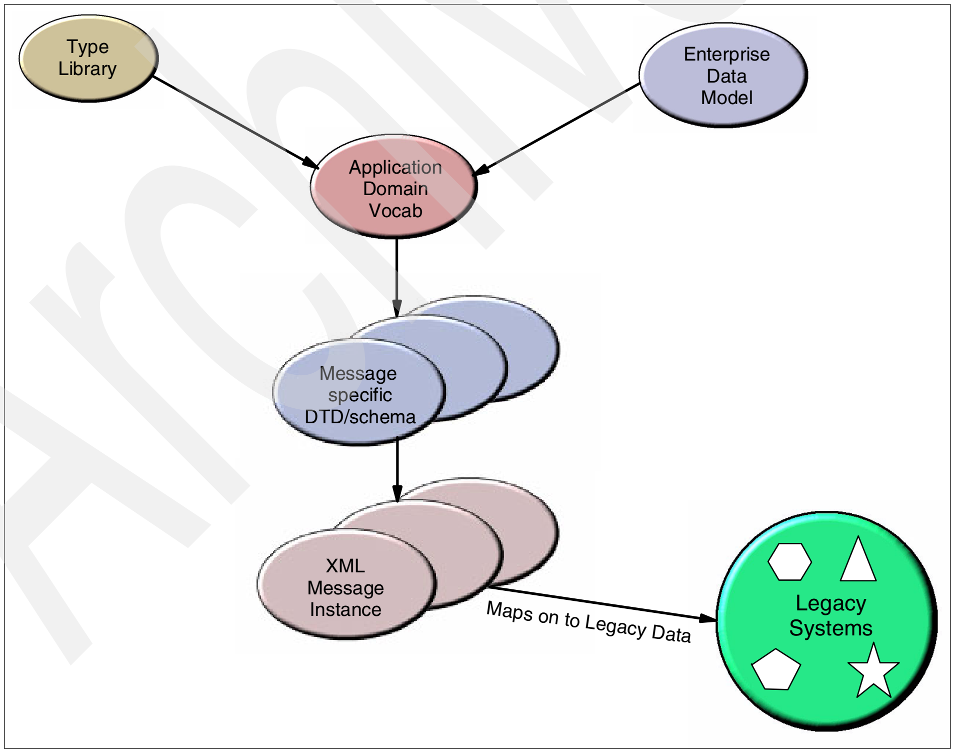


Figure 10-3

The following sections discuss in detail the components shown in the diagram.

10.2.2 Type library

The primary purpose of a type library is to increase the quality of data exchanged across the service interfaces. An abstract data type library is used to provide type constraints on the elements that constitute an XML message. If you are using XML Schema, most of the basic types are available as part of the Schema. However, if you are using DTD, you have to develop one yourself using XML's attribute definition capability. As you progress through the development of service interfaces, you can develop more abstract data types (for example, Date, CustomerId, PhoneNumber, and so forth), which will ensure uniform implementation data representation across all services of an organization. For a good example, visit the IFX Forum at:

`http://www.ifxforum.org.`

10.2.3 Enterprise data model

An enterprise data model (EDM) is inherent in every large organization's data management process. However, depending on an organization's maturity and technological sophistication, the EDM can vary greatly in form, complexity, and level of granularity. It can range from the abstract (in some long-term staffer's head), to the ad hoc (a bunch of file boxes, for instance), all the way to an elaborate enterprise repository.

WebSphere Studio Asset Analyzer holds the promise of delivering the infrastructure needed for a robust enterprise repository. In the meantime, if you don't have a model, there are some industry-specific ones available, from IBM and others. For example, IBM offers two major enterprise models: Information FrameWork (IFW) for banking, and Insurance Application Architecture (IAA) for the insurance industry.

Coming back to why we need an EDM, its role is to act as a *reference model* to ensure that the services we are developing are meaningful and reusable (in other words, from a business perspective these services make sense). We are concerned about business usability perspective. Unless the data content of a service interface is validated against an EDM, there is every likelihood that the implementation could be a point solution at best. Of course we can employ additional techniques, such as Use Cases and the like, to validate the functionality and usefulness of a service from business users, but EDM occupies a central spot in the development of reusable enterprise services.

At this point we want to emphasize that services development should not be viewed as a mechanical process of quickly slapping XML structure onto a transaction interface (which a tool like WSED lets you do with a mere click of a button). If we go down this quick and easy path, from a data management perspective, we will be propagating existing “data anarchy” one level up, to the world of XML.

In the following sections we outline a path on which organizations can start this journey in a more methodical manner.

10.2.4 Domain-specific vocabulary

This is an XML artifact (DTD/Schema) that defines all the elements that constitute a particular domain of business interest—a Customer, for example. The elements defined in the vocab are constrained syntactically through the type library and semantically through an EDM.

From an enterprise management perspective, a particular business unit will own these domains. It goes almost without saying that these domain-specific vocabs constitute the single most important asset in the e-business initiative of an enterprise. This is also a critical asset for enterprise portal and personalization efforts.

So, how do you start building a vocab? There is no single, right way; you can approach it from the bottom up as well as from the top down. From our experience, we found working from the bottom up is practical and delivers immediate results. The EDM should be consulted constantly to ensure that definitions are correct, and that in the case of complex elements (also sometimes known as aggregates) the scope of functionality is well understood and represented.

The problem with trying to define the vocab from the top down is that if you stray in your definitions, it may become very cumbersome to connect to the applications in production. While the intention of defining a vocab in a top-down fashion is very noble (that is, it would result in truly generic elements, and an interface defined based upon such definitions would lay the foundation of a highly reusable Service) the goal could be quite elusive to achieve, particularly because of the difficulty of mapping to implemented systems in production.

The main reason a top down approach is difficult is that EDM is usually an *abstraction* of implemented enterprise data stores, and any attempt to map to an implemented system would invariably lead to a tangled web of DBMS accesses, which would make the services completely unusable.

Hence we need to adopt a more patient and evolutionary approach to developing a vocab. Abstracting business data from implemented systems and gradually extending the vocab to include additional simple and complex elements will ultimately deliver the desired goal of providing a vocab that will allow us to build highly reusable services.

10.2.5 Message-specific DTD/Schema

As we have seen already, each Service is represented by a Request and Response pair service interface. And each of these Service interfaces is described in terms of a DTD/Schema that contains simple or complex elements defined in one or more domain-specific Vocabs. At runtime, each request and response message represents a well-formed and valid XML document instance. In a way, this is similar to a polymorphic Class Interface.

The multiple occurrences of the DTD indicate that many Services-specific DTDs can be defined over one or more domain vocabs. These services interface DTDs/Schemas are nothing but a composition mechanism to present a useful business function. The domain-specific vocabs hold all the definitions of the contents of the interface.

10.2.6 Message instances

There could be multiple XML message instances, depending on the cardinality of data in request and reply. These multiple XML message instances are well-formed and valid XML documents based on the corresponding Service interface DTD/Schema.

10.2.7 A final note about XML-based message design

We want to reiterate that currently developers tend to use XML as an encoding mechanism, and they are satisfied with an XML message that is a well formed document. We would like to remind them that this is not even half the story.

To take full advantage of the capabilities of data-centric XML, you have to model the service interfaces from a data perspective, and capture them in Service-specific interfaces. By doing this you are not only cutting down on data exceptions on both the client side and the server side (and hence unnecessary exception messages over the wire), you are also laying the foundation of a solid enterprise metadata repository. This is also a great opportunity for organizations to clean up their legacy applications by properly designing the service interface. Achievement of this objective alone will provide a major milestone in legacy modernization efforts.

10.3 Design by Contract and Service Design

The principal idea behind Design by Contract (DbC) is that a *Service provider* (supplier) and a *Service requestor* (client) have a contract with one another: The requestor must guarantee certain conditions (*precondition*) before calling for a service to be provided by the service provider, and the service provider has to guarantee certain properties are held true (*postcondition*) after the service is delivered. This is the basic idea of DbC, formally presented by Bertrand Meyer in the programming language he designed, called Eiffel.

The notion of contract is one of the most common in human dealings, for example, when one party (the supplier) performs some task for another party (the client). Each party expects some *benefits* from the contract and accepts certain *obligations* in return.

The DbC-based software development technique ensures high-quality software by guaranteeing that every component of a system lives up to its expectations, as identified in Table 10-1. As a developer using DbC, you specify component *contracts* as part of the component's interface. The contract specifies what the component expects of clients, and what clients can expect of it.

Table 10-1 Benefits and obligations of DbC

	Benefit	**Obligation**
Client	No need to check output values Result guaranteed to comply with Post conditions	Satisfy precondition
Supplier	No need to check input values Input guaranteed to comply with precondition	Satisfy Post conditions

The idea behind DbC is fairly straightforward, but it is a very powerful technique to ensure robustness from the software engineering perspective. Central to DbC is the notion of an *assertion* - a Boolean expression about the state of a software system. At runtime, we evaluate the assertions at specific checkpoints during the system's execution. In a valid software system, all assertions evaluate to true. In other words, if any assertion evaluates to false, we consider the software system invalid or broken.

This ability to evaluate addresses the goal of producing reliable software, but it is important to remember that *correctness* is not a property of the software: a software system is correct (or incorrect) with respect to a certain specification.

Assertions are meant to express such a specification. What DbC does is to take care of the behavioral aspects of the specification of a software component or service. In this way, DbC allows enforcement of trustability in the component.

It is interesting to note that if the Service interface is implemented as a data-centric XML document as suggested in 10.2, “XML-based message design” on page 223, the parsing process will implement the DbC principle at a service level. WSDL is a contract between a client and supplier and, as such, its specification should be based on DbC principles.

Formally (except for Eiffel and a few others), there are few programming languages that have implemented DbC as part of the language construct; however, third party extensions are available for Java (iContract from Reto Kramer, for example). As for those languages, such as COBOL and PL/1, that do not have such support, at least DbC should become part of the software specification and should form part of the program module documentation.

In conclusion, in service-oriented application solutions, DbC should be part of every interacting software component.

For more details about DbC, consult the following:

1. Bertrand Meyer, *Object-Oriented Software Construction*.
2. *Building bug-free O-O Software: An Introduction to Design by Contract*

   http://www.eiffel.com/doc/manuals/technology/contract/
3. Grady Booch, *The Illusion of Simplicity*
4. Clements Szyperski, *Components and Web Services*
5. Reto Kramer, “*iContract – The Java Design by Contract Tool*”

Glossary

address space. A range of virtual storage pages identified by a number (ASID) and a collection of segment and page tables which map the virtual pages to real pages of the computer's memory.

address space connection. The result of connecting an allied address space to DB2. Each address space containing a task connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See allied address space and task control block.

Advanced Program-to-Program communication (APPC). (1) The general facility characterizing the LU6.2 architecture and its implementation in different SNA products. (2) Sometimes used to refer to an LU6.2 product feature in particular, such as an APPC application programming interface.

allied address space. An area of storage external to DB2 that is connected to DB2 and is therefore capable of requesting DB2 services.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

ANSI. See *American National Standards Institute*.

APAR. See *authorized program analysis report*.

API. See *Application Program Interface.*

applet. See *Java Applet*.

application. (1) A program or set of programs that perform a task; for example, a payroll application. (2) In Java programming, a self-contained, stand-alone Java program that includes a static main method. It does not require an applet viewer. Contrast with applet.

application plan. The control structure produced during the bind process and used by DB2 to process SQL statements encountered during statement execution.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester (AR). See *requester.*

Application Service Provider (ASP). An ASP is an agent or broker that aggregates, facilitates and brokers IT services to deliver IT-enabled business solutions across a network via subscription-based pricing.

application-owning region (AOR). A CICS region in an MRO environment that “owns” the CICS applications, and invokes them on behalf of remotely attached terminal (or Web) users. See also *TOR* and *listener region.*

AR. Application requester. See *requester*.

ASCII. (1) American Standard Code for Information Interchange. A standard assignment of 7-bit numeric codes to characters. See also *Unicode*. (2) An encoding scheme used to represent strings in many environments, typically on PCs and workstations. Contrast with *EBCDIC*.

attribute. In XML, a name="value" pair that can be placed in the start tag of an element. The value must be quoted with single or double quotes.

authorization ID. A string that can be verified for connection to DB2 and to which a set of privileges are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

automatic bind. (More correctly automatic rebind). A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

base table. (1) A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with result table and temporary table. (2) A table containing a LOB column definition. The actual LOB column data is not stored along with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with *auxiliary table*.

basic mode. A S/390 central processing mode that does not use logical partitioning. Contrast with logically partitioned (LPAR) mode.

bean. A definition or instance of a JavaBeans component. See *JavaBeans*.

bind. The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

browser. (1) In VisualAge for Java, a window that provides information on program elements. There are browsers for projects, packages, classes, methods, and interfaces. (2) An Internet-based tool that lets users browse Web sites.

bytecode. Machine-independent code generated by the Java compiler and executed by the Java interpreter.

call level interface (CLI). A callable application program interface (API) for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 CLI does not require the user to precompile or bind applications, but instead provides a standard set of functions to process SQL statements and related services at run time.

Cascading Style Sheet (CSS). CSS defines a stylesheet language for HTML 4.0. CSS allows a Web page designer to separately specify style elements of a Web page, such as colors, fonts and font styles.

case-sensitive. Indicates whether an application, processor, or operating system distinguishes between upper and lower case. If it does, it is case-sensitive. XML tags are case-sensitive, but HTML tags are not.

casting. Explicitly converting an object or primitive's data type.

catalog. In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table. Any table in the DB2 catalog.

CGI. The Common Gateway Interface (CGI) is a means of allowing a Web server to execute a program that you provide rather than to retrieve a file. A number of popular Web servers support the CGI. For some applications (for example, displaying information from a database), you must do more than simply retrieve an HTML document from a disk and send it to the Web browser. For such applications, the Web server has to call a program to generate the HTML to be displayed. The CGI is not the only such interface, however.

channel-attached. (1) Pertaining to attachment of devices directly by data channels (I/O channels) to a computer. (2) Pertaining to devices attached to a controlling unit by cables rather than by telecommunication lines.

character large object (CLOB). See CLOB.

class. An encapsulated collection of data and methods to operate on the data. A class may be instantiated to produce an object that is an instance of the class.

class hierarchy. The relationships between classes that share a single inheritance. All Java classes inherit from the Object class.

class method. Methods that apply to the class as a whole rather than its instances (also called a *static method*).

class variable. Variables that apply to the class as a whole rather than its instances (also called a *static field*).

CLASSPATH. In your deployment environment, the environment variable keyword that specifies the directories in which to look for class and resource files.

CLI. See *call level interface*.

client. (1)A networked computer in which the IDE is connected to a repository on a team server. (2) See requester.

CLOB. A sequence of bytes representing single-byte characters or a mixture of single and double-byte characters where the size can be up to 2 GB - 1. Although the size of character large object values can be anywhere up to 2 GB - 1, in general, they are used whenever a character string might exceed the limits of the VARCHAR type.

codebase. An attribute of the <APPLET> tag that provides the relative path name for the classes. Use this attribute when your class files reside in a different directory than your HTML files.

column function. An SQL operation that derives its result from a collection of values across one or more rows. Contrast with scalar function.

commit. The operation that ends a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes.

Common Connector Framework. In the Enterprise Access Builder, interface and class definitions that provide a consistent means of interacting with enterprise resources (for example, CICS and Encina® transactions) from any Java execution environment.

connection. In the VisualAge for Java Visual Composition Editor, a visual link between two components that represents the relationship between the components. Each connection has a source, a target, and other properties.

connection handle. The data object that contains information associated with a connection managed by DB2 CLI. This includes general status information, transaction status, and diagnostic information.

content model. In XML, the expression specifying what elements and data are allowed within an element.

cookie. (1) A small file stored on an individual's computer; this file allows a site to tag the browser with a unique identification. When a person visits a site, the site's server requests a unique ID from the person's browser. If this browser does not have an ID, the server delivers one. On the Wintel platform, the cookie is delivered to a file called 'cookies.txt,' and on a Macintosh platform, it is delivered to 'MagicCookie.' Just as someone can track the origin of a phone call with Caller ID, companies can use cookies to track information about behavior. (2) Persistent data stored by the client in the Servlet Builder.

cursor. A named control structure used by an application program to point to a row of interest within some set of rows, and to retrieve rows from the set, possibly making updates or deletions.

Customer relationship management (CRM). CRM includes the systems and infrastructure required to analyze, capture and share all parts of the customer's relationship with the enterprise. From a strategy perspective, it represents a process to measure and allocate organizational resources to those activities that have the greatest return and impact on profitable customer relationships.

Data Access Bean. In the VisualAge for Java Visual Composition Editor, a bean that accesses and manipulates the content of JDBC/ODBC-compliant relational databases.

Data Access Builder. A VisualAge for Java Enterprise tool that generates beans to access and manipulate the content of JDBC/ODBC-compliant relational databases.

data source. A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver which supports the ODBC APIs. In the case of DB2 for OS/390, the data sources are always relational database managers.

database management system (DBMS). A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

DB2 thread. The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources and services.

DBCLOB. A sequence of bytes representing double-byte characters where the size can be up to 2 gigabytes. Although the size of double-byte character large object values can be anywhere up to 2 gigabytes, in general, they are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

DBMS. Database management system.

direct access storage device (DASD). A mass storage medium on which a computer stores data.

distributed relational database architecture (DRDA®). A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

DLL (dynamic link library). A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously. The DLL's Enterprise Access Builders also generate platform-specific DLLs for the workstation and OS/390 platforms.

Document Object Model (**DOM)**. This allows the representation and manipulation of an XML document in memory as a programming object. DOM is defined by the World-Wide Web Consortium.

Document Type Definition (DTD). A DTD is a definition of which Elements and Attributes are acceptable in a specific XML file. The DTD therefore defines a subset of XML which may be used for a particular application.

DOM. (see Document Object Model).

DOM Tree. A DOM Tree is an in-memory representation of an XML Document.

double precision. A floating-point number that contains 64 bits. See also *single precision*.

double-byte character large object (DBCLOB). See DBCLOB.

DRDA. Distributed relational database architecture.

duplex. Pertaining to communication in which data or control information can be sent and received at the same time. Contrast with *half duplex.*

dynamic bind. A process by which SQL statements are bound as they are entered.

Dynamic I/O Reconfiguration. A S/390 function that allows I/O configuration changes to be made non-disruptively to the current operating I/O configuration.

dynamic SQL. SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

EBCDIC. Extended binary coded decimal interchange code. An encoding scheme used to represent character data in the MVS, VM, VSE, and OS/400® environments. Contrast with ASCII.

EBNF. Extended Backus-Naur Form. A formal set of production rules that comprise a grammar defining another language, such as XML.

Electronic data interchange. The automatic machine-to-machine transfer of trading documents (for example, invoices and purchase orders) using electronic networks such as the Internet. Originally conducted only through value-added networks, EDI is gradually moving to the Internet.

element. In XML, a start tag and its end tag, plus the content between the tags. An empty tag is also an element.

embedded SQL. SQL statements coded within an application program. See static SQL.

embeddedJava. An API and application environment for high-volume embedded devices, such as mobile phones, pagers, process control, instrumentation, office peripherals, network routers and network switches. EmbeddedJava applications run on real-time operating systems and are optimized for the constraints of small-memory footprints and diverse visual displays.

empty declaration. In XML, the DTD declaration for an empty tag. For example, if <foo/> is an empty tag, the empty declaration looks like: <!ELEMENT foo EMPTY>.

empty tag. In XML, a start and end tag combined in one tag. The tag has a trailing slash, so an XML parser can immediately recognize it as an empty tag and not bother looking for a matching end tag. For example, if foo is an empty tag, it looks like <foo/>.

Enterprise Java. Includes Enterprise JavaBeans as well as open API specifications for: database connectivity, naming and directory services, CORBA/IIOP interoperability, pure Java distributed computing, messaging services, managing system and network resources, and transaction services.

Enterprise JavaBeans. A cross-platform component architecture for the development and deployment of multi-tier, distributed, scalable, object-oriented Java applications.

Enterprise JavaBeans (EJB). The Enterprise JavaBeans specification defines a way of building transactionally aware business objects in Java.

Enterprise Systems Architecture/390® (ESA/390). An IBM architecture for mainframe computers and peripherals. Processors that follow this architecture include the S/390 Server family of processors.

entity. In XML, an entity declaration provides the ability to have constants or replacement strings, which are expanded by a pre-processor. An entity declaration maps some token to a replacement string. Later the token can be prefixed with the '&' character and the replacement string is put in its place.

environment handle. In DB2 ODBC, the data object that contains global information regarding the state of the application. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

ESA/390. See *Enterprise Systems Architecture/390*.

exception. An exception is an object that has caused some sort of new condition, such as an error. In Java, *throwing* an exception means passing that object to an interested party; a signal indicates what kind of condition has taken place. *Catching* an exception means receiving the sent object. *Handling* this exception usually means taking care of the problem after receiving the object, although it might mean doing nothing (which would be bad programming practice).

executable content. Code that runs from within an HTML file (such as an applet).

extends. A subclass or interface extends a class or interface if it add fields or methods, or overrides its methods.

external function. A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with sourced function and built-in function.

Extranet. In some cases intranets have connections to other independent intranets. An example would be one company connecting its intranet to the intranet of one of its suppliers. Such a connection of intranets is called an extranet. Depending on the implementation, they may or may not be fully or partially visible to the outside.

factory. A bean that dynamically creates instances of beans.

FastCGI. FastCGI is a way of combining the advantages of CGI programming with some of the performance benefits you get by using the GWAPI. FastCGI, written by Open Market, Inc., is an extension to normal Web server processing. It requires server-specific API support, which is available for AIX®, Sun Solaris, HP-UX, and OS/390. With FastCGI you can start applications in independent address spaces and pass requests for these applications from the Web server. The communication is through either the TCP/IP sockets interface or UNIX Domain socket bind path in the Hierarchical File System (HFS).

fibre channel standard. An ANSI standard for a computer peripheral interface. The I/O interface defines a protocol for communication over a serial interface that configures attached units to a communication fabric. The protocol has four layers. The lower of the four layers defines the physical media and interface, the upper of the four layers defines one or more logical protocols (for example, FCP for SCSI command protocols and FC-SB-2 for FICON™ for ESA/390). Refer to ANSI X3.230.1999x.

FICON. (1) An ESA/390 computer peripheral interface. The I/O interface uses ESA/390 logical protocols over a FICON serial interface that configures attached units to a FICON communication fabric. (2) An FC4 proposed standard that defines an effective mechanism for the export of the SBCON command protocol via fibre channels.

field. A data object in a class; for example, a variable.

File Transfer Protocol (FTP). In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

first tier. The client; the hardware and software with which the end user interacts.

foreign key. A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

form data. A generated class representing the HTML form elements in a visual servlet.

FTP. See *File Transfer Protocol*.

function. A specific purpose of an entity or its characteristic action, such as a column function or scalar function. (See column function and scalar function.). Furthermore, functions can be user-defined, built-in, or generated by DB2. (See user-defined function, external function, sourced function.)

garbage collection. Java's ability to clean up inaccessible unused memory areas ("garbage") on the fly. Garbage collection slows performance, but keeps the machine from running out of memory.

GWAPI. Because CGI has some architectural limitations, most Web servers provide an equivalent mechanism that is optimized for their native environment. Domino™ Go Web Server, IBM's strategic Web server, offers the Domino Go Web Server Application Programming Interface (GWAPI), optimized for a given environment, such as OS/390. The GWAPI enables you to create dynamic content similar to the CGI, but in a more specialized way than the generalized CGI. The GWAPI process is similar to OS/390 exit processing. There is an exit point for various server functions that can be exploited.

half duplex. In data communication, pertaining to transmission in only one direction at a time. Contrast with *duplex.*

handle. In DB2 CLI, a variable that refers to a data structure and associated resources. See connection handle, environment handle.

hard disk drive. (1) A storage media within a storage server used to maintain information that the storage server requires. (2) A mass storage medium for computers that is typically available as a fixed disk or a removable cartridge.

hierarchy. The order of inheritance in object-oriented languages. Each class in the hierarchy inherits attributes and behavior from its superclass, except for the top-level Object class.

HTTPS. HTTPS is a de facto standard developed by Netscape for making HTTP flows secure. Technically, it is the use of HTTP over SSL.

Hypertext Markup Language (HTML). A file format, based on SGML, for hypertext documents on the Internet. Allows for the embedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are embedded using URLs, enabling readers to jump directly to the referenced document.

Hypertext Transfer Protocol (HTTP). The Internet protocol, based on TCP/IP, used to fetch hypertext objects from remote hosts.

IDE. See *Integrated Development Environment*.

Identifier. A unique name or address that identifies things such as programs, devices or systems.

initial program load (IPL). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs.

Integrated Development Environment (IDE). In VisualAge for Java, the set of windows that provide the user with access to development tools. The primary windows are the Workbench, Log, Console, Debugger, and Repository Explorer.

Internet. The vast collection of interconnected networks that use TCP/IP and evolved from the ARPANET of the late 1960s and early 1970s. The number of independent networks connected into this vast global net is growing daily.

Internet Protocol (IP). In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical network. However, this protocol does not provide error recovery and flow control, and does not guarantee the reliability of the physical network.

interpreter. A tool that translates and executes code line-by-line.

Intranet. A private network inside a company or organization that uses the same kinds of software that you would find on the Internet, but that are only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks; for example, many companies have Web servers that are available only to employees.

IP. See *Internet Protocol*.

IPL. See *initial program load*.

JAR file format. JAR (Java Archive) is a platform-independent file format that aggregates many files into one. Multiple Java applets and their requisite components (.class files, images, sounds and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction.

Java. An object-oriented programming language for portable, interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated. The Java environment consists of the JavaOS, the Virtual Machines for various platforms, the object-oriented Java programming language, and several class libraries.

Java applet. A small Java program designed to run within a Web browser. It is downloadable and executable by a browser or network computer.

Java beans. Java's component architecture, developed by Sun, IBM, and others. The components, called Java beans, can be parts of Java programs, or they can exist as self-contained applications. Java beans can be assembled to create complex applications, and they can run within other component architectures (such as ActiveX and OpenDoc).

Java Development Kit (JDK). The set of Java technologies made available to licensed developers by Sun Microsystems. Each release of the JDK contains the following: the Java Compiler, Java Virtual Machine, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools.

Java Naming and Directory Interface (JNDI). A set of APIs that assist with the interfacing to multiple naming and directory services. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Native Interface (JNI). A native programming interface that allows Java code running inside a Java Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C and C++.

Java Platform. The Java Virtual Machine and the Java Core classes make up the Java Platform. The Java Platform provides a uniform programming interface to a 100% Pure Java program regardless of the underlying operating system. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Remote Method Invocation (RMI). Java Remote Method Invocation is method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java. Included in JDK 1.1.

Java Runtime Environment (JRE). A subset of the Java Development Kit for end users and developers who want to redistribute the JRE. The JRE consists of the Java Virtual Machine, the Java Core Classes, and supporting files. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Server Page (JSP). Java Server Pages are Web pages that include dynamic tags which are executed on the server. JSPs are the presentation layer for Web-based applications built in Java.

Java Virtual Machine (JVM). A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

JavaDoc. Sun's tool for generating HTML documentation on classes by extracting comments from the Java source code files.

JavaScript. A scripting language used within an HTML page. Superficially similar to Java but JavaScript scripts appear as text within the HTML page. Java applets, on the other hand, are programs written in the Java language and are called from within HTML pages or run as standalone applications.

JDBC (Java Database Connectivity). In the JDK, the specification that defines an API that enables programs to access databases that comply with this standard.

JIT. See *Just-In-Time compiler.*

JNDI. See *Java Naming and Directory Interface.*

JNI. See *Java Native Interface.*

JRE. See *Java Runtime Environment*.

Just-In-Time compiler (JIT). A platform-specific software compiler often contained within JVMs. JITs compile Java bytecodes on-the-fly into native machine instructions, thereby reducing the need for interpretation.

JVM. See *Java Virtual Machine*.

LAN. See *local area network*.

large object (LOB). See *LOB*.

licensed internal code (LIC). Microcode that IBM does not sell as part of a machine, but instead, licenses to the customer. LIC is implemented in a part of storage that is not addressable by user programs. Some IBM products use it to implement functions as an alternate to hard-wire circuitry.

linker. A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses. In Java, the linker creates an executable from compiled classes.

load module. A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

LOB (large object). A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB -1 bytes in length. See also CLOB, DBCLOB.

local area network (LAN). A computer network located in a user's premises within a limited geographic area.

logical partition (LPAR). A set of functions that create a programming environment that is defined by the ESA/390 architecture. ESA/390 architecture uses this term when more than one LPAR is established on a processor. An LPAR is conceptually similar to a virtual machine environment except that the LPAR is a function of the processor. Also, LPAR does not depend on an operating system to create the virtual machine environment.

logical switch number (LSN). A two-digit number used by the I/O Configuration Program (IOCP) to identify a specific ESCON® Director.

logically partitioned (LPAR) mode. A central processor mode, available on the Configuration frame when using the PR/SM™ facility, that allows an operator to allocate processor hardware resources among logical partitions. Contrast with *basic mode.*

LPAR. See *logical partition*.

megabyte (MB). (1) For processor storage, real and virtual storage, and channel volume, 2^{20} or 1 048 576 bytes. (2) For disk storage capacity and communications volumes, 1 000 000 bytes.

method. A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

middle tier. The hardware and software that resides between the client and the enterprise server resources and data. The software includes a Web server that receives requests from the client and invokes Java servlets to process these requests. The client communicates with the Web server via industry standard protocols such as HTTP and IIOP.

middleware. A layer of software that sits between a database client and a database server, making it easier for clients to connect to heterogeneous databases.

multithreading. Multiple TCBs executing one copy of code concurrently (sharing a processor) or in parallel (on separate central processors).

National Committee for Information Technology Standards. NCITS develops national standards, and its technical experts participate on behalf of the United States in the international standards activities of ISO/IEC JTC 1, information technology.

native class. Machine-dependent C code that can be invoked from Java. For multi-platform work, the native routines for each platform need to be implemented.

NCITS. See *National Committee for Information Technology Standards.*

NUL terminator. In C, the value that indicates the end of a string. For character strings, the NUL terminator is X'00'.

null. A special value that indicates the absence of information.

NUL-terminated host variable. A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

object. The principal building block of object-oriented programs. Objects are software programming modules. Each object is a programming unit consisting of related data and methods.

ODBC. See Open Database Connectivity.

ODBC driver. A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

Open Database Connectivity (ODBC). A Microsoft database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called database drivers that link the application to their choice of database management systems at runtime. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

open system. A system whose characteristics comply with standards made available throughout the industry and that therefore can be connected to other systems complying with the same standards.

original equipment manufacturers information (OEMI). A reference to an IBM guideline for a computer peripheral interface. More specifically, refer to IBM S/360 and S/370™ Channel to Control Unit Original Equipment Manufacturer's Information. The interface uses ESA/390 logical protocols over an I/O interface that configures attached units in a multi-drop bus environment.

package. A program element that contains classes and interfaces.

persistence. In object models, a condition that allows instances of classes to be stored externally, for example in a relational database.

Persistence Builder. In VisualAge for Java, a persistence framework for object models, which enables the mapping of objects to information stored in relational databases and also provides linkages to legacy data on other systems.

plan. See *application plan*.

plan name. The name of an application plan.

precompilation. A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

prepare. The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

prepared SQL statement. A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

primary key. A unique, non-null key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

process. A program executing in its own address space, containing one or more threads.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM in a current unaltered release of a program.

property. An initial setting or characteristic of a bean, for example, a name, font, text, or positional characteristic.

PTF. See *program temporary fix*.

RDBMS. Relational database management system.

reentrant. Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. Re-entrancy is a compiler and operating system concept, and re-entrancy alone is not enough to guarantee logically consistent results when multithreading.

reference. An object's address. In Java, objects are passed by reference rather than by value or by pointers.

relational database management system (RDBMS). A relational database manager that operates consistently across supported IBM systems.

remote. Refers to any object maintained by a remote DB2 subsystem; that is, by a DB2 subsystem other than the local one. A remote view, for instance, is a view maintained by a remote DB2 subsystem. Contrast with local.

Remote Method Invocation (RMI). RMI is a specific instance of the more general term RPC. RMI allows objects to be distributed over the network; that is, a Java program running on one computer can call the methods of an object running on another computer. RMI and java.net are the only 100% pure Java APIs for controlling Java objects in remote systems.

Remote Object Instance Manager. In Remote Method Invocation, a program that creates and manages instances of server beans through their associated server-side server proxies.

Remote Procedure Calls (RPC). RPC is a generic term referring to any of a series of protocols used to execute procedure calls or method calls across a network. RPC allows a program running on one computer to call the services of a program running on another computer.

requester. Also application requester (AR). The source of a request to a remote RDBMS, the system that requests the data.

RMI (Remote Method Invocation). See *Remote Method Invocation.*

rollback. The process of restoring data changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with commit.

RPC. See *Remote Procedure Calls*.

runtime system. The software environment where compiled programs run. Each Java runtime system includes an implementation of the Java Virtual Machine.

sandbox. A restricted environment, provided by the Web browser, in which Java applets run. The sandbox offers them services and prevents them from doing anything naughty, such as doing file I/O or talking to strangers (servers other than the one from which the applet was loaded). The analogy of applets to children led to calling the environment in which they run the "sandbox."

scalar function. An SQL operation that produces a single value from another value and is expressed as a function name followed by a list of arguments enclosed in parentheses. See also column function.

Secure Socket Layer (SSL). SSL is a security protocol that allows communications between a browser and a server to be encrypted and secure. SSL prevents eavesdropping, tampering, or message forgery on your Internet or intranet network.

security. Features in Java that prevent applets downloaded off the Web from deliberately or inadvertently doing damage. One such feature is the digital signature, which ensures that an applet came unmodified from a reputable source.

serialization. Turning an object into a stream, and back again.

server. The computer that hosts the Web page that contains an applet. The .class files that make up the applet, and the HTML files that reference the applet reside on the server. When someone on the Internet connects to a Web page that contains an applet, the server delivers the .class files over the Internet to the client that made the request. The server is also known as the originating host.

server bean. The bean that is distributed using RMI services and is deployed on a server.

servlet. See *Java servlet*.

SGML. See *Standardized Generalized Markup Language.*

Shell. The user interface of UNIX system softwares. In z/OS, an xpg4.2-compliant shell is used. Very often OMVS is used as an interface for z/OS shells.

single precision. A floating-point number that contains 32 bits. See also double precision.

Small Computer System Interface (SCSI). (1) An ANSI standard for a logical interface to computer peripherals and for a computer peripheral interface. The interface uses a SCSI logical protocol over an I/O interface that configures attached targets and initiators in a multi-drop bus topology. (2) A standard hardware interface that enables a variety of peripheral devices to communicate with one another.

SmartGuide. In IBM software products, an active form of help that guides you through common tasks.

source type. An existing type that is used to internally represent a distinct type.

sourced function. A function that is implemented by another built-in or user-defined function already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with external function and built-in function.

SQL. Structured Query Language. A language used by database engines and servers for data acquisition and definition.

SSL. See *secure socket layer*.

Standardized Generalized Markup Language. An ISO/ANSI/ECMA standard that specifies a way to annotate text documents with information about types of sections of a document.

static bind. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time. Contrast with dynamic bind.

static SQL. SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables specified by the statement might change).

stored procedure. A user-written application program, that can be invoked through the use of the SQL CALL statement.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

Sysout. The regular output for a program on z/OS is SYSOUT. It is the functional equivalent of stdout on UNIX. In batch, there can be multiple SYSOUTs.

System. A single instance of the z/OS or OS/390 operating system in a sysplex.

System Management End User Interface (SMEUI). A Windows-based tool that makes it possible to perform administrative tasks for WebSphere Application Server from a Windows workstation. The SMEUI tool is used to deploy a new application to WebSphere on z/OS.

task control block (TCB). A control block used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. See *address space connection*.

TCB. Task Control Block; manages dispatchable tasks. Each UNIX thread is assigned to a TCB.

Telnet. Telnet provides a virtual terminal facility that allows users of one computer to act as if they were using a terminal connected to another computer. The Telnet client program communicates with the Telnet daemon on the target system to provide the connection and session.

temporary table. A table created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with result table.

thin client. Thin client usually refers to a system that runs on a resource-constrained machine or that runs a small operating system. Thin clients don't require local system administration, and they execute Java applications delivered over the network.

third tier. The third tier, or back end, is the hardware and software that provides database and transactional services. These back-end services are accessed through connectors between the middle-tier Web server and the third-tier server. Though this conceptual model depicts the second and third tier as two separate machines, the NCF model supports a logical three-tier implementation in which the software on the middle and third tier is on the same box.

thread. A separate flow of control within a program.

timestamp. A seven-part value that consists of a date and time expressed in years, months, days, hours, minutes, seconds, and microseconds.

trace. A facility that provides the ability to monitor and collect monitoring, auditing, performance, accounting, statistics, and serviceability data.

Trading communities. Trading communities bring together buyers and sellers in a central online location to trade, using various online mechanisms including auctions and exchanges, in addition to industry content and application services. Trading communities are owned and operated by both large industry players in closed trading networks and by neutral parties in more fragmented open communities.

transaction. (1) In a CICS program, an event that queries or modifies a database that resides on a CICS server. (2) In the Persistence Builder, a representation of a path of code execution. (3) The code activity necessary to manipulate a persistent object. For example, a bank application might have a transaction that updates a company account.

UDF. See *user-defined function.*

UDT. See *user-defined data type*.

Unicode. A 16-bit international character set defined by ISO 10646. See also *ASCII*.

Uniform Resource Locator (URL). The unique address that tells a browser how to find a specific Web page or file.

URI/URL. A Uniform Resource Identifier (URI) and Uniform Resource Locator (URL) uniquely define a location on the Web. URLs are familiar to anyone who browses the Web (for example http://www.ibm.com®), and the term URI is a more general term which also incorporates other schemes for identifying resources.

URL. See *Uniform Resource Locator.*

user-defined data type (UDT). See *distinct type*.

user-defined function (UDF). A function defined to DB2 using the CREATE FUNCTION statement that can be referenced thereafter in SQL statements. A user-defined function can be either an external function or a sourced function. Contrast with built-in function.

valid. An XML document is valid if its content conforms to the rules in its DTD.

variable. (1) An identifier that represents a data item whose value can be changed while the program is running. The values of a variable are restricted to a certain data type. (2)A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with constant.

vi. A popular UNIX editor. It can only be used from an ASCII Telnet connection.

virtual machine. A software or hardware implementation of a central processing unit (CPU) that manages the resources of a machine and can run compiled code. See *Java Virtual Machine*.

visual bean. In the Visual Composition Editor, a bean that is visible to the end user in the graphical user interface.

WAP. Wireless Application Protocol. Offers Internet browsing from wireless handsets.

Web. See *World Wide Web.*

Web Application. A WebSphere Web application is a collection of static pages, JSPs, and Servlets that share a common URL prefix, and together make a complete application.

Web browser. The Web uses a client/server processing model. The Web browser is the client component. Examples of Web browsers include Mosaic, Netscape Navigator, and Microsoft Internet Explorer. The Web browser is responsible for formatting and displaying information, interacting with the user, and invoking external functions, such as Telnet, or external viewers for data types that it does not directly support. Web browsers are fast becoming the universal client for the GUI workstation environment, in much the same way that the ability to emulate popular terminals such as the DEC VT100 or IBM 3270 allows connectivity and access to character-based applications on a wide variety of computers. Web browsers are available for all popular GUI workstation platforms and are inexpensive (often included with operating systems or related products for no additional charge.)

Web server. Web servers are responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server's local disk or generated by a program called by the server to perform a specific application function. Web servers are sometimes referred to as httpd servers or daemons. A number of Web servers are available for most platforms including most UNIX variants, OS/2® Warp, OS/390, and Windows NT.

well-formed. An XML document is well-formed if there is one root element, and all its child elements are properly nested within each other. Start tags must have end tags, and each empty tag must be designated as such with a trailing slash. Also, all attributes must be quoted, and all entities must be declared.

white-space. In XML, characters that are not visible, but used in formatting documents or programs. These characters include the SPACE, TAB, NEWLINE, and CARRIAGE-RETURN characters.

World Wide Web. A network of servers that contain programs and files. Many of the files contain hypertext links to other documents available through the network.

WWW. See *World Wide Web*.

XML. The Extensible Markup Language (XML) is an important new standard emerging for structured documents on the Web. XML extends HTML beyond a limited tag set and adapts SGML, making it easy for developers to write programs that process this markup and providing for a rich, more complex encoding of information.

XSL Stylesheet. The eXtensible Stylesheet Language defines stylesheets for XML Documents. It is composed of two parts: the formatting objects, and XSLT (see below). XSL is defined by the WorldWide Web Consortium.

XSLT. eXtensible Stylesheet Language Transformations. This defines the part of the XSL specification which allows the stylesheet to reformat and reorganize the XML data. It is most often used to transform XML into XSL.

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 248. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns on z/OS: Connecting Self Service Applications to the Enterprise*, SG24-6827
- *Legacy Modernization with WebSphere Studio Enterprise Developer*, SG24-6586
- *Using XML on z/OS and OS/390 for Application Integration*, SG24-6285
- “From code to deployment: Connecting to CICS from WebSphere v4.0.1 for z/OS,” REDP0206
- *Enterprise COBOL for z/OS and OS/390 Programming Guide* , SC27-1412
- *WebSphere Application Server V4.0.1 for z/OS and OS/390 Assembling Java 2 Platform, Enterprise Edition (J2EE) Applications,* SA22-7836

Other resources

These publications are also relevant as further information sources:

- *Patterns for e-business: A Strategy for Reuse,* by Jonathan Adams, et al, ISBN 1-931182-02-7
- *Building Web Services with Java* by Steve Graham, et al ISBN 0-672-32181-5
- *Object-Oriented Software Construction,* by Bertrand Meyer
- *Building bug-free O-O Software: An Introduction to Design by Contract*

  http://www.eiffel.com/doc/manuals/technology/contract/
- *The Illusion of Simplicity,* by Grady Booch
- *Components and Web Services,* by Clements Szyperski
- *iContract – The Java Design by Contract Tool*, by Reto Kramer

Referenced Web sites

These Web sites are also relevant as further information sources:

- Patterns for e-business

  http://www.ibm.com/developerworks/patterns/
- XML Schema

  http://www.w3.org/XML/Schema
- Tree Regular Expressions for XML

  http://www.thaiopensource.com/trex/
- Schema for Object-oriented XML

  http://www.w3.org/TR/NOTE-SOX
- Schema for Object-oriented XML

  http://www.ascc.net/xml/resource/schematron/schematron.html

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

IBM

Redbooks

XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture

Redbooks

Leverage XML and XSL-based applications on z/OS and OS/390

Design concepts for Web services architectures on z/OS

Implement solutions based on practical examples

This IBM Redbook describes the use of XML on IBM servers running z/OS or OS/390, and how it can be extended to modernize legacy applications. It provides both a high-level discussion of service-oriented architecture along with practical, detailed information about XML.

In addition to an overview of XML concepts, the first part of the book provides detailed instructions for installing the XML Toolkit for z/OS and OS/390 V1.4 and running the sample programs bundled with it. It describes how to use various tools that are part of the services development environment, details the support for XML in Enterprise COBOL, and provides an overview of the IBM WebSphere Application Server. This material is of interest mainly to system programmers and application programmers.

The second part of the book is geared more to the needs of application developers and architects. It provides a comprehensive introduction to service-oriented architecture (SOA) and Web services, and describes in detail some service-based topologies for both legacy systems and new applications. Finally, this book presents some important design concepts to enable the reader to build robust SOA-based solutions rapidly. This includes an introduction to the IBM Patterns for e-business, as well as XML-based message design, and the principles of design by contract and service design.

SG24-6826-00 ISBN 0738426156