DB2

Web Services Object Runtime Framework: Implementing DB2 Web Services

DB2

Web Services Object Runtime Framework: Implementing DB2 Web Services

Note
Before using this information and the product it supports, read the information in Appendix F, “Notices” on page 91.

First Edition (September, 2002)

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

Contents

About this document

The information in this document provides concepts and steps for creating Web services in DB2® using the Web services Object Runtime Framework (WORF) and optionally DB2 XML Extender. The framework uses a document access definition extension (DADX) file to generate the Web services, called DADX Web services. In this document you will find the following information:

- A description of WORF
- Set-up information for running WORF on WebSphere® Application Server Advanced Edition, Single Edition, or Extended Edition, or Apache Jakarta Tomcat
- Syntax and examples for the DADX file
- Details for generating Web Service Definition Language (WSDL) and XML schema (XSD) files, including support for UDDI Best Practices
- Details of running the DADX checker

Who should read this document

This document is written for Web application developers who want to create Web services that access DB2 and XML data that is stored in DB2. Some basic database knowledge is assumed as well as knowledge of XML Web services, Simple Object Access Protocol (SOAP), and Web Service Definition Language (WSDL). Many examples and some functions require DB2 XML Extender, and some knowledge of that product is assumed.

Related information

This document refers to the following information resources:

- *DB2 XML Extender: Administration and Programming*
- *DB2 XML Extender: DB2 XML Extender Administration and Programming, Version 7.2 Release Notes*
- *Dynamic e-business: The next stage of e-business and Web services*
- Apache Jakarta Tomcat documentation
- Simple Object Access Protocol (SOAP) 2.2 or later
- Universal Description, Discovery and Integration of Business for the Web
- *Using WSDL in a UDDI Registry 1.07*
- W3C XML Schema
- *Web Services Description Language (WSDL) 1.1*
- *Web services zone*
- WebSphere Application Server Advanced Edition Version 4 documentation
- *WebSphere Handbook*
- DB2 Developer Domain, Web Services (this is especially important for the most recent version of this document)

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this document or the WORF product, send your comments by e-mail to db2xml@us.ibm.com. Put ″WORF″ in the subject line and be sure to include the name of the book, the version of DB2 XML Extender, and, if applicable, the specific location of the text that you are commenting on (for example, a page number or table number).

Chapter 1. Web services Object Runtime Framework

Web services are XML-based application functions that can be invoked over the Internet. You can define a basic Web service by using standard SQL statements and DB2 XML Extender stored procedures. For Web Services that involve advanced transformations between XML and relational data, use the DB2 XML Extender.

Overview

WORF provides an environment to easily create simple XML based Web services that access DB2. WORF uses Apache Simple Object Access Protocol (SOAP) 2.2 or later and the Document Access Definition Extension (DADX). A DADX document specifies a Web Service using a set of operations that are defined by SQL statements or XML Extender Document Access Definition (DAD) documents. Web services, or functions invoked over the Internet, specified in a DADX file are called *DADX Web services*, also referred to as DB2 Web services.

You can create DADX documents by using a simple text editor, and tools provided in WebSphere Studio with minimal knowledge of XML or SQL.

You can use DB2 XML Extender to implement Web services using Web services Object Runtime Framework (WORF) tools with DB2. DB2 XML Extender consists of a set of stored procedures, user-defined types (UDT) and user-defined functions (UDF) that you can use to store and retrieve XML data using DB2. DB2 XML Extender allows XML documents to be stored intact, and optionally indexed in side tables, using the XML column access method, or as a collection of relational tables using the XML collection access method. DB2 XML Extender uses an XML document format called Document Access Definition (DAD) to define the mapping between XML and relational data.

Features

WORF provides the following features:

- Resource–based deployment and invocation, or using DADX and, optionally, other resources that help define the Web service
- 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

WORF is currently available on Windows NT®, Windows® 2000, AIX®, Solaris, and Linux.

WORF supports resource-based deployment of Web services. Resource-based deployment means that you define your Web service in a resource file that you place in a directory of your Web application. Then when you request that resource file, WORF loads it and makes it available as a Web service. If you edit the resource file and request it again, WORF detects the change and loads the new version automatically. This process of automatically reloading the resource file makes Web service development more productive.

WORF currently supports Web services defined with either SQL statements or XML collection statements for the XML Extender. To define a Web service with SQL statements, put the statements in a DADX resource file. The format of a DADX file is described in detail in this document.

WORF is available from the DB2 XML Extender Web site, or with DB2 Universal Database™ Version 8 and WebSphere Studio Version 4 and Version 5. When delivered with WebSphere Studio, WORF is supported with a set of tools that automate the building of DADX Web services. These tools include a wizard to create DADX files based on SQL statements or DAD files, and tools to create DAD files. WORF also works with Informix™.

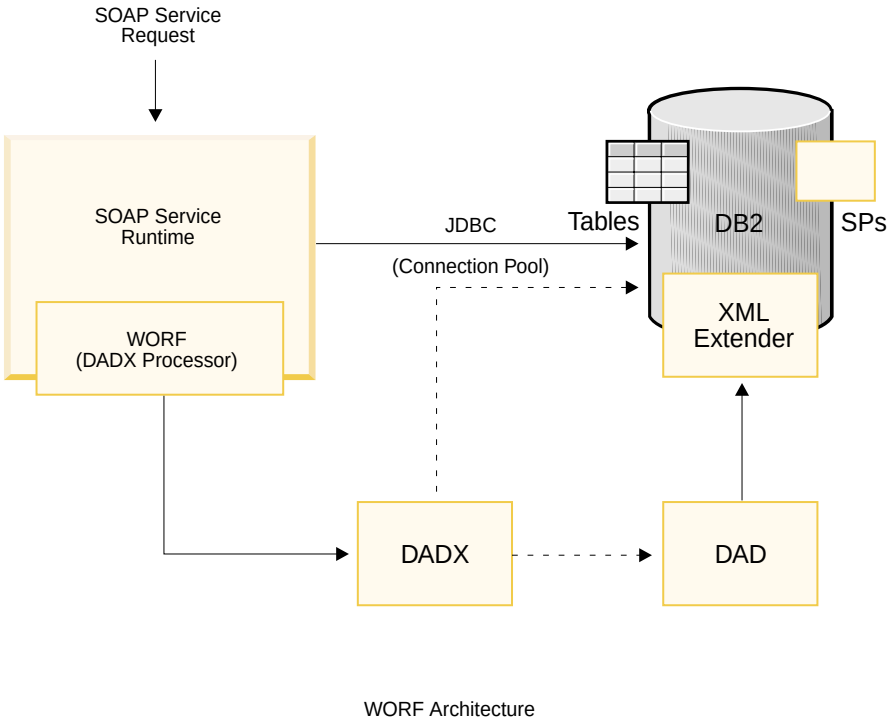

Figure 1. WORF framework and XML Extender

Figure 1 shows how WORF processes a Web service request:

1. WORF receives an HTTP SOAP, GET or POST service request.

   The URL of the Web service request includes the name of the Web service's resource file and a command. The command is either a built-in command or specifies an operation of the Web service. The built-in commands include TEST, WSDL, and XML schema files (XSD), in which case WORF generates either an HTML test page, a WSDL document, or an XML schema file. If the request is a SOAP request then the operation name is specified in the request body. See "Example of POST request" on page 73 and "Example of SOAP request" on page 74 for an example of a POST and SOAP request. Otherwise, the command is the operation name. In either case, WORF invokes the specified operation of the Web service and returns the result document. A DADX operation can also contain input parameters. WORF performs the following steps in response to a Web service request:

   a. Loads the DADX file specified in the request.

   b. Generates a response, based on the request:

      - For operations:

1) Loads a DAD file, if requested.
2) Replaces query parameters with requested values
3) Connects to DB2 and runs any SQL statements, including SQL calls.
4) Commits the database transaction.
5) Formats the result into XML, converting types as necessary.

- For commands: generates necessary files, test pages, or other responses required.

c. Returns the response to the service requestor.

This model implies that you can only have one transaction per Web service invocation. If you want multiple SQL operations in one transaction, write stored procedures executing these statements and use those stored procedures as your Web service operation.

The DADX file

A DADX file specifies how to create a Web service using a set of operations that are defined by SQL statements (including stored procedure calls) and, optionally, DAD files. These Web services store or retrieve XML documents managed by DB2 XML Extender. WORF provides the run-time support for invoking DADX documents as Web services in the Apache SOAP 2.2 (or later) engine supported by WebSphere Application Server and Apache Jakarta Tomcat.

In addition to specifying storage and retrieval operations on XML data, WORF allows stored procedures and SQL statements to be exposed as invokable Web service operations. You can expose any database stored procedure. WORF assumes that your stored procedures have result sets with fixed metadata (a fixed number and a fixed shape). The operation signature includes the input and output parameters. You can also specify SQL statements to select, insert, update and delete data. And, simple mapping of XML schema to SQL data types is provided. These features do not require the XML Extender.

DADX files support two kinds of Web service operations:

XML collection operations (requires DB2 XML Extender)
These storage and retrieval operations help you to map XML document structures to DB2 tables so that you can either compose XML documents from existing DB2 data, or decompose (store untagged element or attribute content) XML documents into DB2 data. This method is useful for data interchange applications, particularly when the contents of XML documents are frequently updated.

There are two elements that make up the XML collection operation type:

<retrieveXML>
generates XML documents

<storeXML>
stores XML documents

The DAD file provides fine-grained control over the mapping of XML documents to a DB2 database for both storage and retrieval.

SQL operations
SQL-based querying is the ability to send SQL statements, including stored procedure calls, to DB2 and to return results with a default tagging. The

data is returned using only a simple mapping of SQL data types, using column names as elements. There are three elements that make up the SQL operations type:

<query>
queries the database

<update>
inserts into a database, deletes from a database, or updates a database

<call> calls stored procedures with multiple result sets

Chapter 2. Installing and configuring WORF

The following sections describe how to set up WORF on the Web server. They describe:

Software requirements for UNIX® and Windows

You set up WORF in any of the following operating systems:

- Windows NT
- Windows 2000
- Linux
- AIX
- Solaris

You can use the following database environments:

- IBM DB2 Universal Database Version 7.2, FixPak 7 or later (http://www.ibm.com/software/data/db2)
  - IBM DB2 XML Extender Version 7.2, FixPak 7 or later (http://www.ibm.com/software/data/db2/extenders/xmlext/index.html) Required for store and retrieve operations
- IBM DB2 Universal Database Version 8 (http://www.ibm.com/software/data/db2). This includes DB2 XML Extender.
- Informix Dynamic Server (IDS) Version 9.3

Additionally, use the following software:

- Java™ JDK Version 1.2 or 1.3 (http://java.sun.com, or http://www.ibm.com/java)
- One of the following Web servers
  - Websphere Application Server Advanced Edition Version 4.01 or Version 5, Single or Extended Edition (http://www-4.ibm.com/software/webservers/appserv/)
    - Apache SOAP 2.2 or later binary, which is included with WebSphere Studio (http://xml.apache.org/ (requires Document Object Model, level 2 (DOM 2), which is supported by Xerces Java 1.4.4 or later.))
  - Apache Web server:
    - Apache Jakarta Tomcat Version 3.3.1 through 4.0.3 (http://www.apache.org/)
      - Apache Jakarta Tomcat Version 4 standard comes with the appropriate Xerces
      - For Apache Jakarta Tomcat versions earlier than Version 4, you must add Xerces to your CLASSPATH to use it as the XML parser

- SOAP 2.2 or later binary (http://xml.apache.org/ (requires Document Object Model, level 2 (DOM 2), which is supported by Xerces Java 1.4.4 or later.))
- Xerces Java parser Version 1.4.4 (http://xml.apache.org/)
- JavaMail Version 1.2 (http://java.sun.com/)
- JavaBeans™ Activation Framework Version 1.0.1 (http://java.sun.com/). This is required by Apache SOAP.

Downloading and accessing WORF

For DB2 Universal Database Version 7.2 FixPak 7, you can download the Web Object Runtime Framework (WORF) from the DB2 Developer Domain, *http://www.ibm.com/software/data/developer*.

WORF also comes with DB2 Universal Database Extended Server Edition, Version 8. WORF is found in the following path in DB2 UDB Version 8: *sqllib\samples\java\Websphere\dxxworf.zip*.

Configuring WORF for WebSphere Application Server on UNIX and Windows

You can run DB2 Web services on WebSphere Application Server (WAS) Advanced Edition. WORF provides the run-time support for invoking DADX documents as Web services over HTTP with Apache SOAP 2.2 (or later), which is supported by WebSphere Application Server 4.0.1 or higher and other servlet engines. WebSphere lets you secure your SOAP Web services. See the WebSphere documentation on securing SOAP services for more information. The following sections describe:

Installing the software requirements

Ensure that you have the required software installed. See "Software requirements for UNIX® and Windows" on page 5 to verify your installation. The DB2 XML Extender is required for advanced mapping control between XML and relational data. Verify the installation by creating the DB2 SAMPLE database. WORF requires JDBC 2.0, which is the default in DB2 Universal Database Version 8. The procedures to prepare the WORF environment are as follows:

1. Stop any services that use DB2 (such as Web Application Server)
2. Stop DB2.
3. For DB2 Universal Database versions earlier than Version 8, select JDBC 2.0 by running the `C:\SQLLIB\java12\usejdbc2.bat` file, assuming that you installed DB2 in C:\SQLLIB\ using a Windows environment.
4. Restart DB2.
5. Start WebSphere Application Server Advanced Edition 4.01 or 5.0 (WAS). For the rest of this document it is assumed that you installed WAS in C:\WebSphere\Appserver.
6. Install Apache SOAP 2.2 or later.
   a. Download, and unzip the `soap-bin-2.2.zip` file.

b. WebSphere Application Server Version 4.01 or later comes with soap.jar in *WebSphere\AppServer\lib* directory. Verify that you have soap.jar in directory *WebSphere\AppServer\lib*. If not, copy the extracted file `soap-2_2\lib\soap.jar` into `WebSphere\AppServer\lib\soap.jar`.

Installing WORF on WAS Version 4.01 or Version 5.0

To install WORF, complete the following steps:

1. Download and unzip `dxxworf.zip` to a directory, such as `C:\worf`. The directory has the following contents:
   - `readme.html`
   - `lib\services.war` - sample Web application containing Web services that use WORF
   - `lib\worf.jar` - WORF library which must be installed on the CLASSPATH of the servlet engine
   - `schemas\` - XML schemas for the DADX and NST XML files
   - `tools\`
2. Copy `worf.jar` to `C:\WebSphere\AppServer\lib`.
3. Start the Websphere Adminstration Server.

To install the WORF examples, complete the following steps:

1. Open the Administrator's console and install `services.war` as an enterprise application. You can use the Wizard menu to configure your application.
2. Give the Web application a context name of your choice. For example, `services`. The Wizard also asks you to select a virtual host (for example: `default_host`) and select an application server (for example: `Default Server`). You can leave the defaults on the remaining pages of the wizard to complete the installation. Figure 2 on page 8 shows the WAS Administrator's Console during the installation of the application:

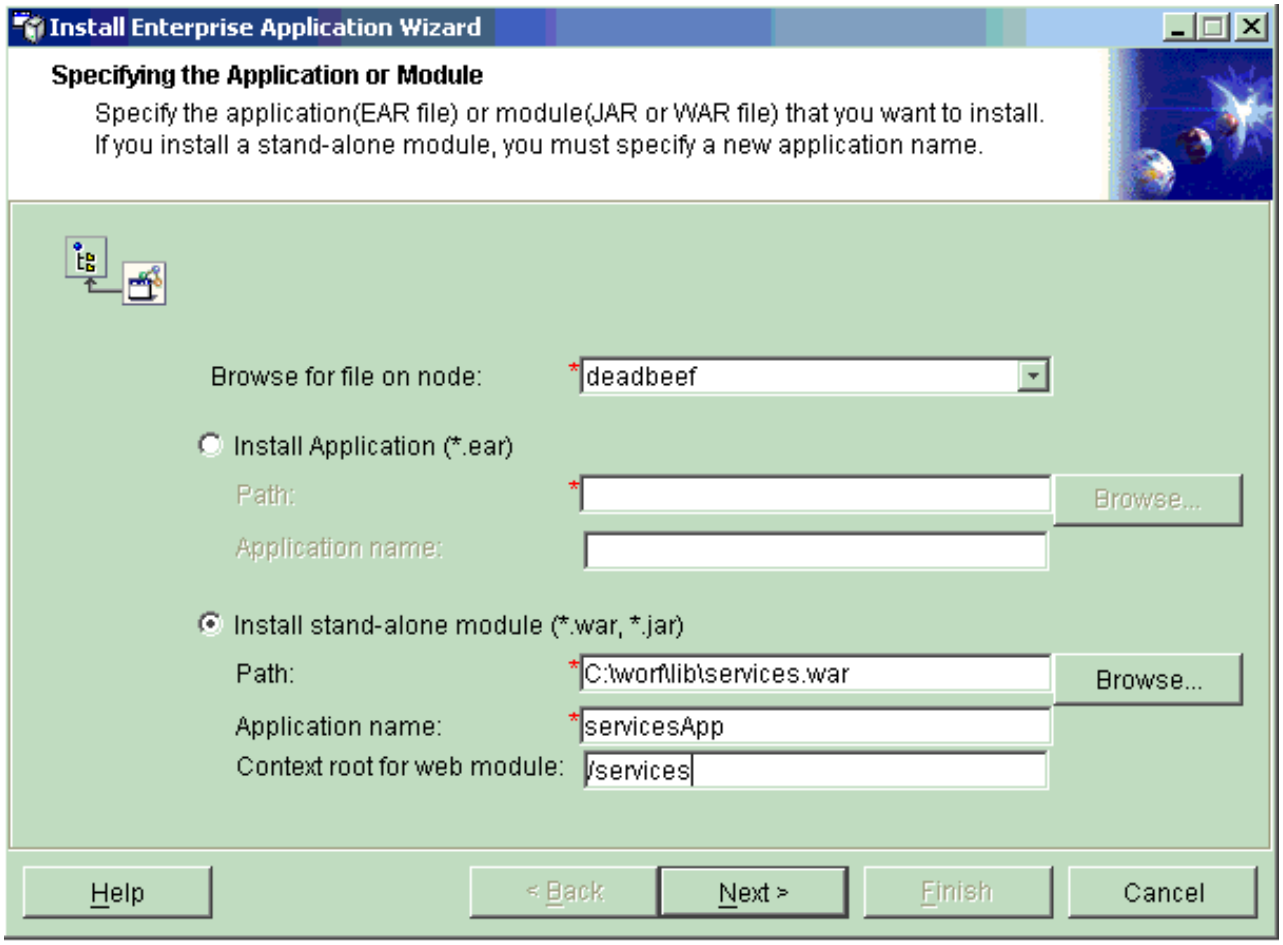

Figure 2. Specifying the application or module

3. Start the application server containing the `services` Web application
4. Verify that the database settings are correct (especially user ID and password) in the group.properties files.
5. Test the installation by opening your browser on the services Web application welcome page. The specific port number varies according to the WAS configuration. If you have used the defaults, the services Web application welcome page might be `http://localhost:9080/services`. The page should look like the screen shown in Figure 3 on page 9:

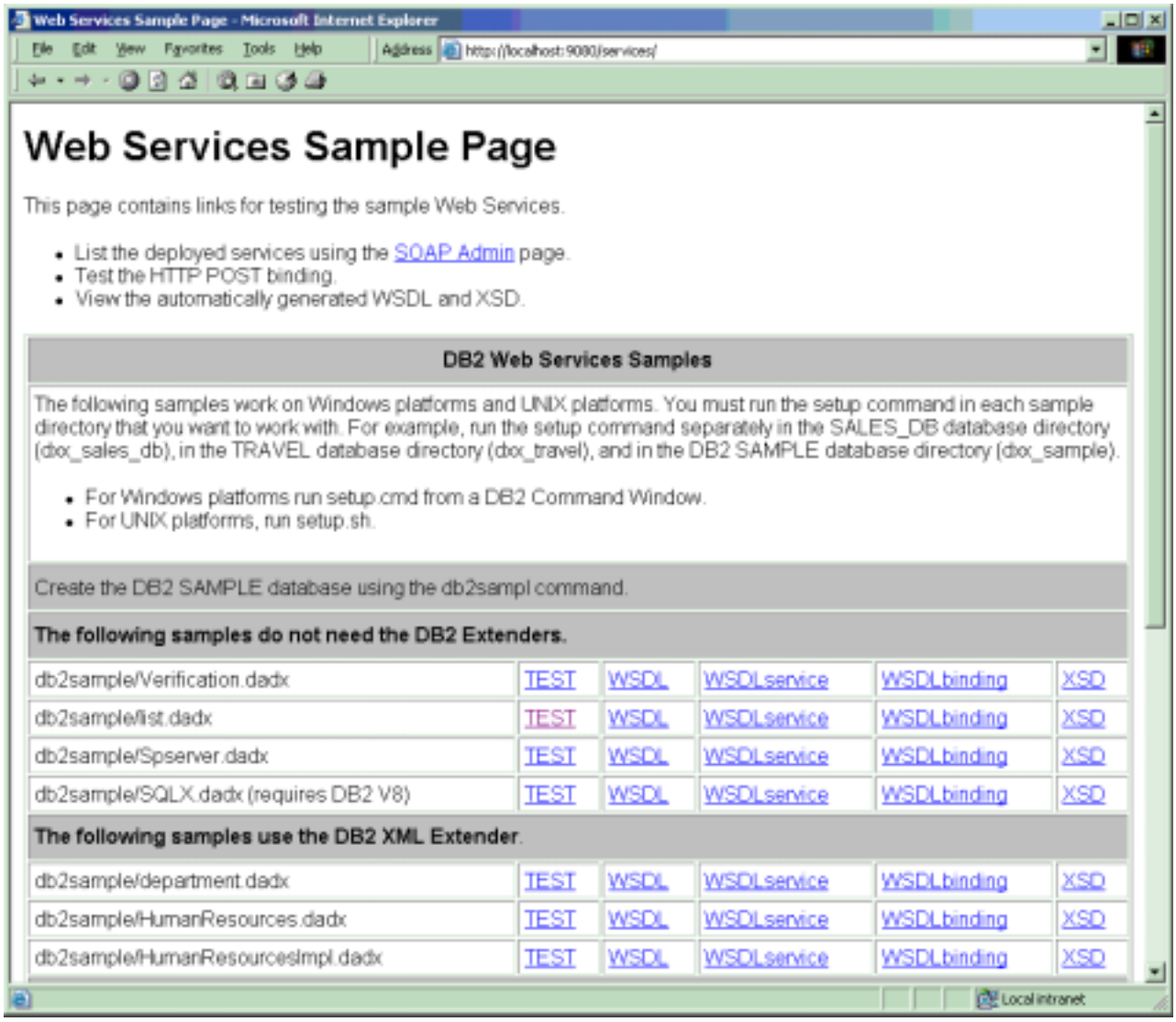

Figure 3. Web services sample page

6. Click on some of the TEST links to see if the sample services work. To learn more about the Web services Sample Page, see “Testing the Web service” on page 49.
7. Run `setup.cmd` in a Windows environment, or *setup.sh* in a UNIX environment in each of the database directories to create the database. For example, in the `dxx_sales_db` directory to set up the SALES_DB database using DB2 XML Extender.

   Now you should be able to use all the TEST links in the SALES_DB Database Samples.
8. You can use either the Apache configuration manager (the default) or the IBM configuration manager (XMLDrivenConfigManager). The IBM configuration manager disables the automatic deployment of Web services when adding new DADX files, so you have to deploy them manually (see “Generating deployment descriptors” on page 47). WORF provides a configuration file for deploying in the services.war file. If you want to use the IBM configuration manager, rename `soap-ibm.xml` to `soap.xml` and `dds-example.xml` to `dds.xml`. The dds-example.xml is a deployment descriptor file for all the DADX files in the examples. Restart the application server to use the

XMLDrivenConfigManager, in the `installedApps\servicesApp.ear\services.war` subdirectory.

The Web application is configured to use the default ConfigManager, which stores the deployed services in a serialized Java file named DeployedServices.ds. The XMLDrivenConfigManager uses an XML format instead. A sample `dds.xml` file that deploys a few services is included. If you invoke a service that is not deployed the server will report that the service is unknown.

Using connection pooling to improve performance

Connection pooling can help improve performance. This step is optional.

You can read more about this topic in the redbook, *WebSphere Handbook* Chapter 10.

1. Create a data source; you can use the Create Data Source Wizard from the Administrative Console, as shown in Figure 4:

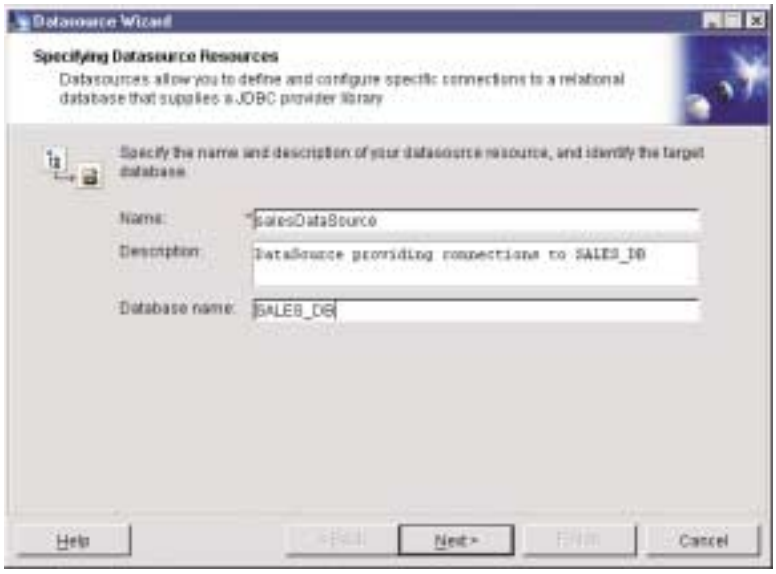

Figure 4. Specifying datasource resources

2. On the **Specifying DataSource Resources** window, specify the name, description and database name of the data source, and choose a JDBC provider.
3. Edit the `group.properties` file in the groups subdirectory and add the following lines of text:

   ```
   initialContextFactory=<your context factory>
   datasourceJNDI=<your DataSource>
   ```

   For example:

   ```
   initialContextFactory=com.ibm.ejs.ns.jndi.CNInitialContextFactory
   datasourceJNDI=jdbc\salesDataSource
   ```

4. Restart the Web application if you have made any changes to the group.properties file so they will take effect. See “Defining the web.xml and group.properties files” on page 23 for more information on group.properties.

Troubleshooting

Table 1 on page 11 describes problems that can occur when you use WORF on WAS 4.0. Recommended solutions are provided.

Table 1. Errors and solutions

Problem	Solution
Error 500: Server caught unhandled exception from servlet [isd_demos]: org.apache.soap.rpc. SOAPContext: method setClassLoader (java\lang\ClassLoader;) not found	SOAP 2.2 or later (soap.jar) is missing. See item 6 on page 6 on how to install SOAP 2.2 or later in WAS.
Clicking on the **Invoke** button from the Web services test page in Internet Explorer results in a 'The page cannot be found' error.	To view a more helpful error message use Netscape to debug the problem. Or, edit the Internet Explorer environment by doing the following steps: 1. Open the **Tools** menu from the Internet Explorer menu bar. 2. Select **Internet Options** from the menu to open the Internet Options window. 3. Click on the **Advanced** tab. 4. Clear the check box next to **Show friendly HTTP error messages**
Error 400: service 'http://tempuri.org /***/***.dadx' unknown	You have to generate a deployment descriptor from the DADX file and restart your Web application before invoking the service (see "Generating deployment descriptors" on page 47).

Configuring WORF for Apache Jakarta Tomcat on UNIX and Windows

You can run DB2 Web services on Apache Jakarta Tomcat. The following sections describe:

- "Installing the software requirements"
- "Installing WORF on Apache Jakarta Tomcat" on page 12

Installing the software requirements

Ensure that you have the required software installed. See "Software requirements for UNIX® and Windows" on page 5 to verify your installation. The DB2 XML Extender is required for advanced mapping control between XML and relational data. The DB2 XML Extender is required for advanced mapping control between XML and relational data. Verify the installation by creating the DB2 SAMPLE database. WORF requires JDBC 2.0, which is the default in DB2 Universal Database Version 8. The procedures to prepare the WORF environment are as follows:

1. Stop DB2.
2. If you are not running DB2 Universal Database Version 8, select JDBC 2.0 by running `C:\SQLLIB\java12\usejdbc2.bat`, assuming that you installed DB2 in C:\SQLLIB\ in a Windows environment.
3. Restart DB2
4. Install the following Internet software:
   - From Apache:
     - Apache Jakarta Tomcat Version 4.0.3 or earlier binary from http://jakarta.apache.org/site/binindex.html. (Apache Jakarta Tomcat Version 4 standard comes with the appropriate Xerces; for earlier versions you must add Xerces to your CLASSPATH to use it as the XML parser.)

  - SOAP 2.2 or later binary from http://xml.apache.org/soap
  - Xerces 1.4.4 from http://xml.apache.org/
- From Sun (http://java.sun.com/products):
  - JavaMail 1.2
  - JavaBeans Activation Framework (JAF) 1.0 1

Installing WORF on Apache Jakarta Tomcat

To install WORF on Apache Jakarta Tomcat,

- Add the following jar files to the classpath: `soap.jar`, `xerces.jar`, `mail.jar`, `activation.jar`, and `worf.jar`. On the Windows platform, add those files to the:*setClasspath* section of the `tomcat\bin\tomcat.bat` file.

  With Apache Jakarta Tomcat version 3.2.4 or earlier, add the following **bold** lines to the `tomcat\bin\tomcat.bat` file:

  Note: Replace each%MYPATH% by the path to the jar file mentioned and C:\SQLLIB\java\db2java.zip by your own db2-jdbc driver path. Assume DB2 is installed in C:\SQLLIB.

```
.... rem ----- Set Up The run-time Classpath -------------------------------------
:setClasspath set CP=%TOMCAT_HOME%\classes
rem --- WORF set up (To add)
set CP=%CP%;%MYPATH%\soap.jar
set CP=%CP%;%MYPATH%\xerces.jar
set CP=%CP%;%MYPATH%\mail.jar
set CP=%CP%;%MYPATH%\activation.jar
set CP=%CP%;%MYPATH%\worf.jar
set CP=%CP%;C:\SQLLIB\java\db2java.zip
rem --- End WORF set up
```

  With Apache Jakarta Tomcat version 3.3.1,

  1. Add the following **bold** lines to the `tomcat\bin\tomcat.bat` file:

     Note: Replace C:\SQLLIB\java\db2java.zip by your own db2-jdbc driver path. Here DB2 is installed in C:\SQLLIB.

```
:setClasspath

set CLASSPATH=%TOMCAT_INSTALL%\lib\tomcat.jar

rem --- WORF set up (To add)
set CLASSPATH=%CLASSPATH%;C:\SQLLIB\java\db2java.zip
set CLASSPATH=%CLASSPATH%;D:\WebServices\xerces-1_4_4\xerces.jar
rem --- End WORF set up
...
```

  2. Put all the installation jar files (soap.jar, activation.jar, mail.jar, worf.jar, xerces.jar) in the tomcat\webapps\services\WEB-INF\lib directory.

     Note: This directory does not exist until you install services.war and start Apache Jakarta Tomcat. Apache Jakarta Tomcat automatically unzips the services.war file and creates this directory.

  With Apache Jakarta Tomcat version 4.0.3,

  1. Add the following **bold** lines to the `tomcat\bin\setclasspath.bat` file:

     Note: Replace C:\SQLLIB\java\db2java.zip by your own db2-jdbc driver path. Here DB2 is installed in C:\SQLLIB.

```
....
rem Set standard CLASSPATH
rem Note that there are no quotes as we do not want to introduce random
rem quotes into the CLASSPATH
set CLASSPATH=%JAVA_HOME%\lib\tools.jar

rem --- Start WORF set up
set CLASSPATH=%CLASSPATH%;C:\SQLLIB\java\db2java.zip
rem --- End WORF set up....
```

2. Put all the installation jar files (soap.jar, activation.jar, mail.jar, worf.jar, xerces.jar) in the tomcat\webapps\services\WEB-INF\lib directory.

To install the WORF examples, do the following,

1. Copy the `services.war` into your `tomcat\webapps` directory.

   If you already have a `services.war` file installed:

   a. Stop Apache Jakarta Tomcat.

   b. Delete the `services` subdirectory under `webapps` and all of its contents.

      CAUTION:
      Any of your previously deployed Web services will be lost with this action, so make sure this is acceptable.

   c. Restart Apache Jakarta Tomcat.

2. Stop and start Apache Jakarta Tomcat (unless you deleted the `services` directory in the previous step).

   The services context starts:

   ```
   ContextManager: Adding context Ctx(\services)
   ```

3. Verify the installation by entering the following URL:

   ```
   http://localhost:8080/services/
   ```

   The specific port address might vary depending on your environment. You should get a page that looks like Figure 3 on page 9. To learn more about the Web services Sample Page, see “Testing the Web service” on page 49.

4. Verify that your database settings are correct, especially user ID and password, in group.properties. Try the verification.dadx on your system (the dynamic test page and the WSDL).

5. To display the XML document, use Internet Explorer Version 5 or a text editor.

6. List the deployed SOAP services in your services context in your system. WORF automatically deploys the services, for each test you run. Click on the SOAP administration link from the Web services Sample Page.

Chapter 3. Creating a DB2 Web service with WORF

This chapter contains the following sections:

Scenario

This section provides an overview of WORF using the SAMPLE database that comes with DB2. As a starting point, we assume that you have set up your application server (see Chapter 2, "Installing and configuring WORF" on page 5 for more details) and are now ready to create a Web service that accesses the SAMPLE database. This scenario assumes that you installed the WORF samples as a Web application named *services* and *services* is configured on your application server.

WORF supports the creation of Web services by using the DADX, which extends the functions of the DAD file used by XML Extender. The DADX file contains necessary information to create a Web service and can reference the DAD file. This scenario will use a simple DADX file, called `HelloSample.dadx`:

```
<?xml version="1.0" encoding="UTF-8"?>
 <DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx">

    <operation name="listDepartments">
      <query>
        <SQL_query>SELECT * FROM DEPARTMENT</SQL_query>
      </query>
    </operation>
 </DADX>
```

Figure 5. Simple DADX file: HelloSample.dadx

To deploy the Web service defined in the DADX file, copy it to the application server in the directory defined by the group *db2sample* in directory dxx_sample.

`HelloSample.dadx` defines a Web service with a single operation named `listDepartment`, which lists the contents of the DEPARTMENT table. The child tag <query> specifies the type of operation. See "The DADX file" on page 3 for an explanation of the elements that comprise an operation type.

Testing the Web service

If you have added the HelloSample.dadx in the correct directory (\WEB-INF\classes\groups\dxx_sample) and if you use the WAS configuration manager that you deployed using the instructions at Using a configuration manager on page 9, then you can test the new service. Access the following URL to begin the test (remember that the *your WebAppServer* identifier depends on your Web server configuration):

`http://<your WebAppServer>/services/db2sample/HelloSample.dadx/TEST`

When you type the address, you see the following automatically generated documentation and test page:

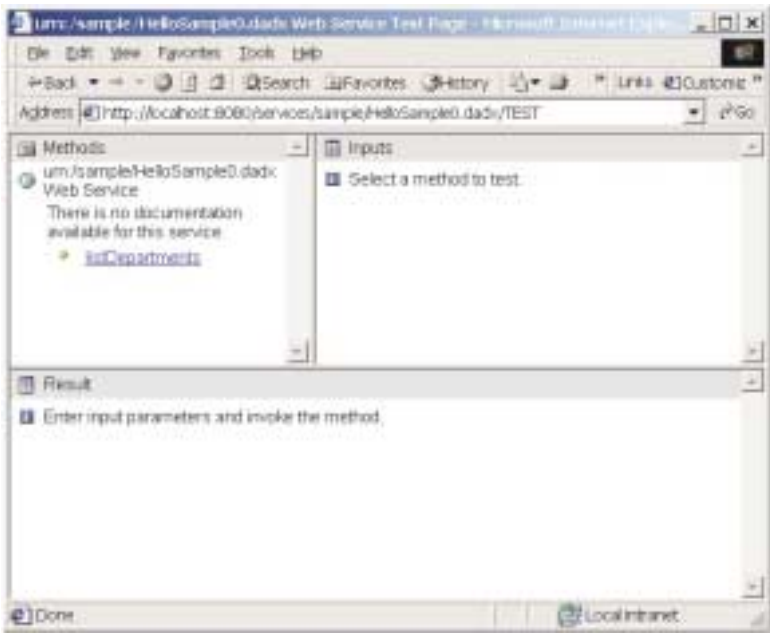

Figure 6. The WORF test page

To test the `listDepartments` operation:

1. Click the `listDepartments` link in the **Methods** frame.
2. Click the **Invoke** push button in the **Inputs** frame.
   The XML result of the operation is displayed in the Result frame:

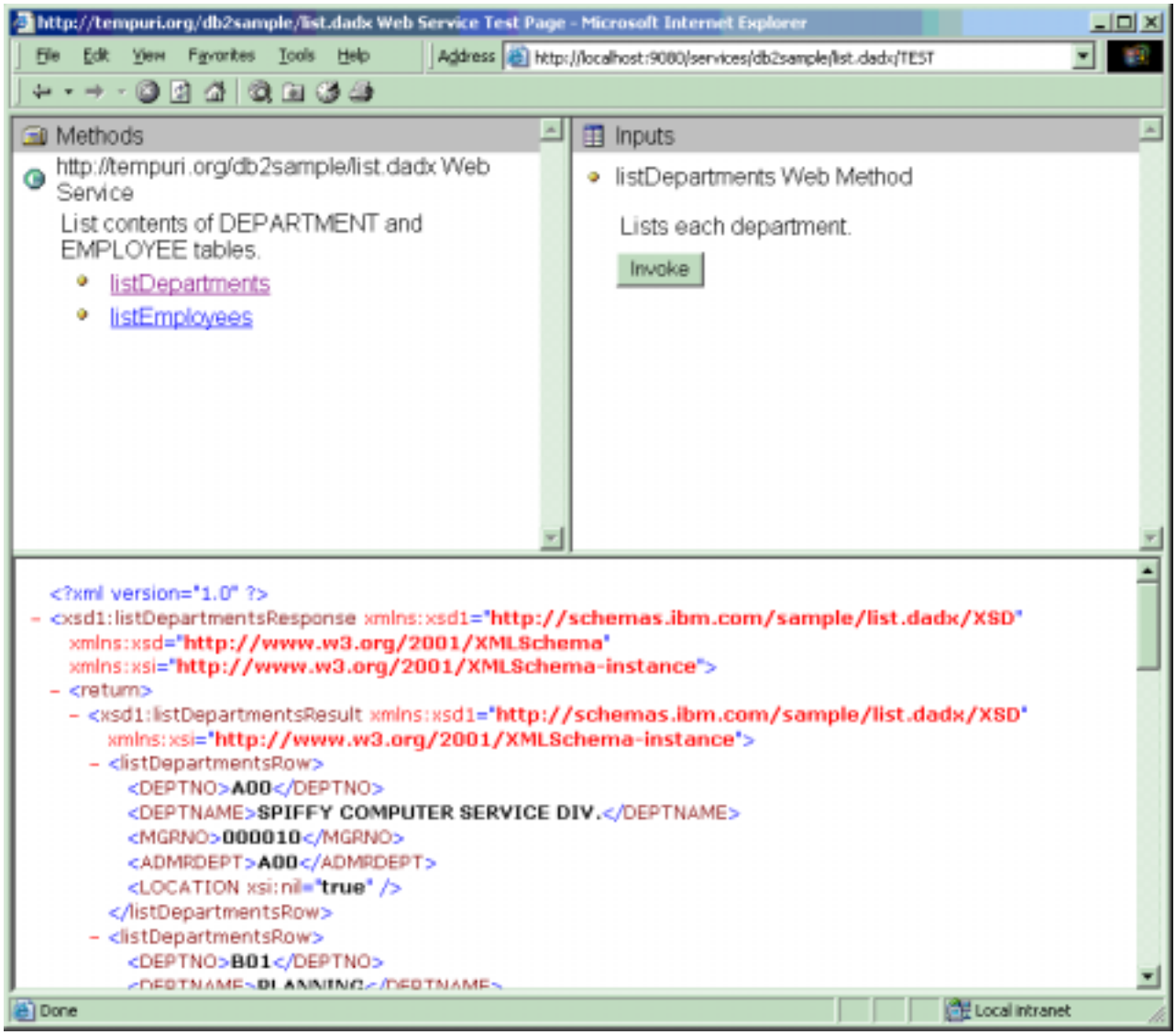

Figure 7. Result of the query

Accessing the Web service

The test page acts as a simple HTML client of the Web Service and uses the HTTP POST binding. The Web Service can also be accessed using HTTP GET and SOAP bindings. The `listDepartments` operation can be invoked by the HTTP GET and POST bindings using the following URL (the localhost port number depends on your own current machine configuration):

```
http://localhost:8080/services/db2sample/HelloSample.dadx/listDepartments
```

This WORF operation returns an XML response that you can save to a file.

```
<?xml version="1.0" ?>
<xsd1:listDepartmentsResponse
  xmlns:xsd1="http://schemas.ibm.com/sample/department.dadx/XSD"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<return>
 <xsd1:listDepartmentsResult
  xmlns:xsd1="http://schemas.ibm.com/sample/department.dadx/XSD"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<listDepartmentsRow>
  <DEPTNO>A00</DEPTNO>
  <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
</listDepartmentsRow>
<listDepartmentsRow>
  <DEPTNO>B01</DEPTNO>
  <DEPTNAME>PLANNING</DEPTNAME>
</listDepartmentsRow>
<listDepartmentsRow>
  <DEPTNO>C01</DEPTNO>
  <DEPTNAME>INFORMATION CENTER</DEPTNAME>
</listDepartmentsRow>
<listDepartmentsRow>
  <DEPTNO>D01</DEPTNO>
  <DEPTNAME>DEVELOPMENT CENTER</DEPTNAME>
</listDepartmentsRow>
         ...
<listDepartmentsRow>
  <DEPTNO>E21</DEPTNO>
  <DEPTNAME>SOFTWARE SUPPORT</DEPTNAME>
</listDepartmentsRow>
</xsd1:listDepartmentsResult>
</return>
</xsd1:listDepartmentsResponse>
```

Figure 8. XML response document

HTTP GET and POST bindings are generally the same as any other HTTP GET and POST requests. For the HTTP GET binding, any input parameters to the operation are added to the URL. But, for the HTTP POST binding, the parameters are sent in the request body.

SOAP binding

The SOAP binding also uses HTTP POST but the operation name, input parameters, and other information are sent as an XML request body.

Use the following URL to access the SOAP binding (remember that the *your WebAppServer* identifier depends on your Web server configuration):

`http://<your WebAppServer>/services/db2sample/HelloSample.dadx/SOAP`

Note: Consider using the SOAP binding for Java and JavaScript clients. WebSphere Studio has the functionality to generate Java Web service clients.

Web services definition language

Request a Web services Definition Language (WSDL) document for the service, `HelloSample.wsdl`, with the following URL:

`http://localhost:8080/services/db2sample/HelloSample.dadx/WSDL`

WORF automatically generates the WSDL document, from DADX. Refer to the WSDL documentation for information on SQL statements and Web services response structure.

UDDI business registries

When you register a Web Service in a Universal Discovery, Description, and Integration (UDDI) business registry, the recommended practice is to split the WSDL document into a service instance document and a binding document. To learn more about UDDI and best practices, see *Using WSDL in a UDDI Registry 1.07*.

The service instance document contains the address at which the service is deployed and it imports the binding document. Many service instances might refer to a common binding document. The binding document is registered in UDDI as a reusable `tModel`. The `tModel` is the information about a specification for a Web service.

Request the WSDL service instance document with the URL:

`http://localhost:8080/services/db2sample/HelloSample.dadx/WSDLservice`

Request the WSDL binding document with the URL:

`http://localhost:8080/services/db2sample/HelloSample.dadx/WSDLbinding`

XML schema definitions

The data types used in the Web service interface are defined using an XML schema. Request the XML schema definitions for the service by the URL:

`http://localhost:8080/services/db2sample/HelloSample.dadx/XSD`

WORF generates an XML schema file similar to the following XML schema example:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace="http://localhost:8080/services/sample/HelloSample.dadx/XSD"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://localhost:8080/services/sample/HelloSample.dadx/XSD">
  <element name="listDepartmentsResult">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="listDepartmentsRow">
          <complexType>
            <sequence>
              <element name="DEPTNO" type="string"/>
              <element name="DEPTNAME" type="string"/>
              <element name="MGRNO" nillable="true" type="string"/>
              <element name="ADMRDEPT" type="string"/>
              <element name="LOCATION" nillable="true" type="string"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Figure 9. The XML schema definition file

The DB2 XML Extender can use DTDs to define the schema of XML documents, so the WORF run-time automatically translates the DTD into an XML Schema. For example, if an XML document is defined using the DTD `order.dtd`, then you can use the following URL to request the translation into XML Schema:

```
http://localhost:8080/services/db2sample/order.dtd/XSD
```

Documentation

You can include documentation in the DADX for the service as a whole and for each operation. `HelloSample1.dadx` illustrates how to add documentation:

```
<?xml version="1.0" encoding="UTF-8"?>
<DADX
  xmlns="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    Simple DADX example that accesses the SAMPLE database.
  </wsdl:documentation>  <operation name="listDepartments">
    <wsdl:documentation>
      Lists the departments.
    </wsdl:documentation>
    <query>
      <SQL_query>SELECT * FROM DEPARTMENT</SQL_query>
    </query>
  </operation>
</DADX>
```

Figure 10. HelloSample1.dadx

The documentation can contain any valid XML. For proper display in a browser, you should use XHTML. Now when you request the test page, the documentation is included:

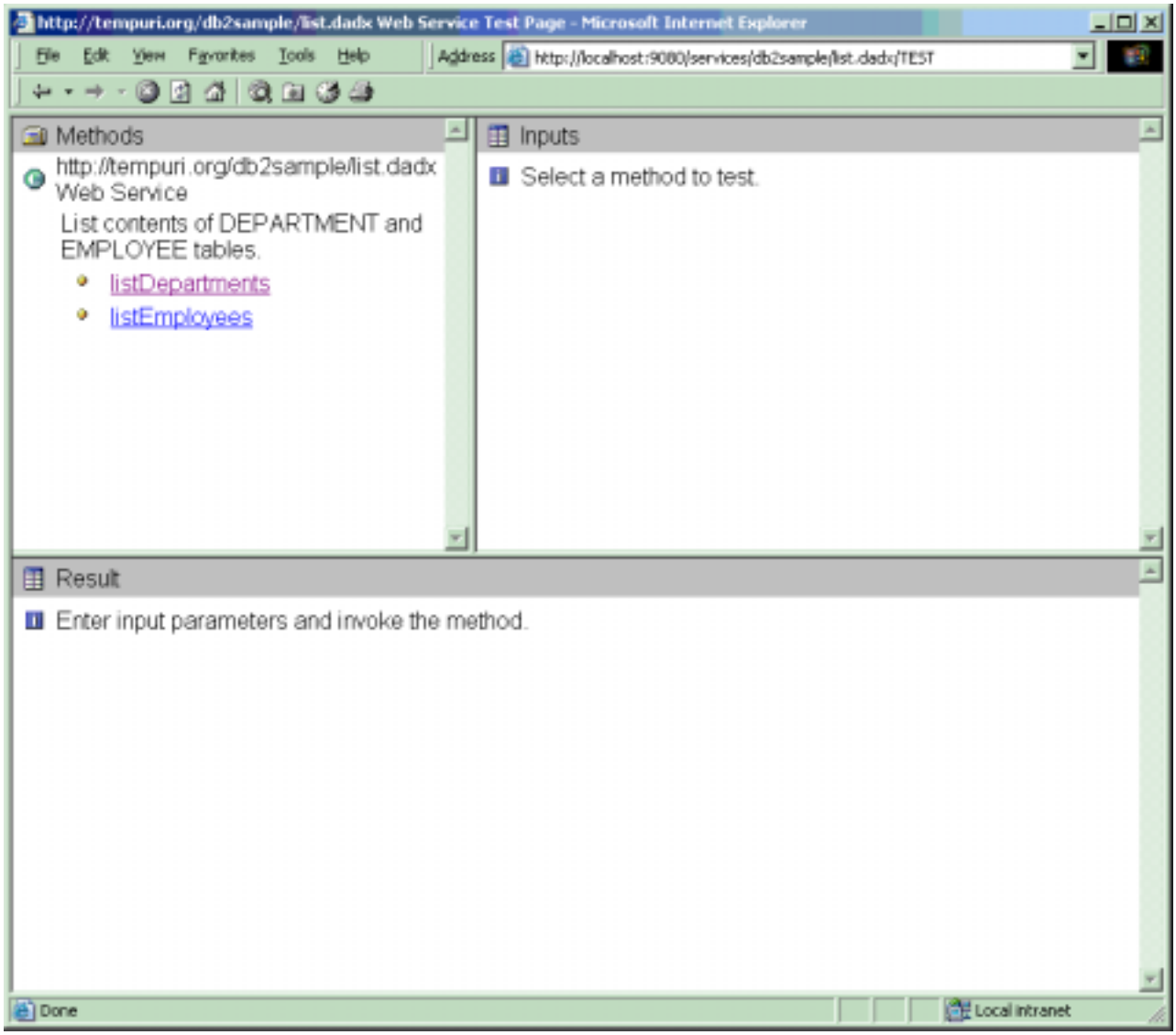

Figure 11. WORF test page with documentation

Automatic Reloading

During the course of development, you are likely to make frequent changes to your DADX files. WORF allows you to make changes to your DADX files while the application server is running, and automatically reloads the DADX file with the new updates. Automatic reloading makes developing DADX Web services as simple as developing Java Server Pages. You can turn off automatic reloading when you deploy your DADX Web services to a production server. See also information on group.properties in “Defining a group of Web services” on page 22.

Overview of the process

Overview of the process:

1. The database administrator sets up the databases.
2. The database administrator optionally enables the databases for DB2 XML Extender. This is required when using the retrieveXML and storeXML

operations or when using the XML column. See the administration chapters of *DB2 XML Extender: Administration and Programming* to learn how to enable the databases for XML Extender.

3. A Web application developer creates a Web application at the Web server. A Web application is a folder on the server that is made up of a collection of related files and tools. Web applications include the interfaces, program flow, program logic, and data access information to create an infrastructure for doing business over the Internet. See the following documentation to learn more:
   - WebSphere Application Server Advanced Edition Version 4 documentation
   - Apache Jakarta Tomcat documentation
4. Within the Web application, the Web application developer creates a database connection configuration for each connection to a database. The connection configuration is stored in the group.properties file, which is a file that contains information about the database connection and other related information used by WORF.
5. The database developer optionally creates the DAD to map XML and relational data conversion (required when using XML Extender stored procedures.) See *DB2 XML Extender: Administration and Programming*.
6. The Web service developer creates the DADX document, which defines a set of operations and contains information used to create the Web service. See "Defining the Web service with the DADX file" on page 26.
7. When using the WAS ConfigManager for Windows or UNIX, the Web service developer creates and deploys a deployment descriptor for the Web service. A deployment descriptor is an isd file, that identifies configuration and deployment information. Each Web service has an `*.isd` file. All of the isd files must be copied into the dds.xml file. (This step is automatic for Apache Jakarta Tomcat users.)
8. The Web service developer verifies the Web service using the DADX test page.

Defining a group of Web services

The resources for all DADX Web service groups are stored in the *groups* directory created when the WORF Web application was configured. On UNIX and Windows, this directory is in the *WEB-INF\classes\groups* subdirectory of the Web application's base directory.

DADX files contain the implementation of the Web services and are therefore similar to Java classes. The `classes` directory is part of the Java CLASSPATH for the Web application. This means that your DADX files can be loaded by the Java class loader and that your Web application can execute directly from its WAR file.

Within the `groups` directory, each group of DADX Web services is stored in a directory with the same name as its servlet instance. For example, the DxxInvoker servlet determines where to find DADX files by looking for a directory that matches its servlet name. In our example, the DADX files are stored in the `WEB-INF\classes\groups\dxx_sample` directory.

Before you begin

Ensure that the database administrator has set up any databases or subsystems required for the application, and enables them for use by XML Extender (if XML Extender is used). The following table is a partial list of some of the files to modify if you use a directory other than *c:\dxx*, which is the default XML Extender install directory.

Table 2. XML Extender samples reference the following DTDs

Platform	Location of DTDs
DB2 Version 7.2 FixPak 7 or later Windows	• c:\dxx\samples\dtd\getstart.dtd • c:\dxx\dtd\dad.dtd.
DB2 Version 8 Windows	• c:\program files\IBM \SQLLIB\samples\db2xml\dtd\ getstart.dtd • c:\program files\IBM \SQLLIB\samples\db2xml\dtd\ dad.dtd
DB2 Version 8 on Solaris	/opt/IBMdb2/V8.1/samples /db2xml/dtd/dad.dtd
DB2 Version 8 on AIX	/usr/opt/db2_08_01/samples /db2xml/dtd/dad.dtd
DB2 Version 8 on Linux	/usr/IBMdb2/V8.1/samples /db2xml/dtd/dad.dtd

Table 3. Some files referencing dad.dtd

department.dad	getstart.xml
department2.dad	order.dad
departmentStd.dad	order-public.dad
sales_db.nst	order-10.xml

Defining the web.xml and group.properties files

In order to define a new group of DADX Web services, complete the following steps:

1. Choose a group name for the DADX group that reflects your application. For these instructions, we will use the name `myapp_group`.
2.
   - For UNIX and Windows, in the *WEB-INF* directory, edit the `web.xml` file to define the group name. You can have multiple group names in the same `web.xml` file. The following figure shows an example of the `web.xml` file. The elements that you need to configure are shown in **bold** with the values defined below.

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
<servlet>
        <servlet-name>myapp_group</servlet-name>
        <servlet-class>com.ibm.etools.
webservice.rt.dxx.servlet.DxxInvoker</servlet-class>
        <init-param>
          <param-name>faultListener</param-name>
          <param-value>org.apache.soap.server.DOMFaultListener</param-value>
        </init-param>
        <load-on-startup>-1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>myyapp_group</servlet-name>
        <url-pattern>/myapp/*</url-pattern>
    </servlet-mapping>
 <welcome-file-list>
       <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
```

Figure 12. web.xml

Only one <servlet> element can exist for each group, but a group can have multiple <servlet-mapping> elements. In this example, the <servlet-name> element defines a group named myapp_group.

When updating the this file, you provide the information for the following elements:

<servlet-name>
The name of the group. The servlet name must be a valid directory name under the groups directory (discussed below), which is used to store the DADX resources for this group of Web services. For example: myapp_group. This element exists under both the <servlet> and <servlet-mapping> elements.

<url-pattern>
The URL associated with the group. The <servlet-mapping> element associates the dxx_sales_db servlet with URLs of the form */url_pattern*/*. The URL pattern must be of this form for WORF to operate correctly. For example: /myapp/*

3. From the groups directory, create a subdirectory with the name of the group specified in the <servlet-name> element added in the previous step. The resources for this group will be stored in this subdirectory.
4. In the group directory, create a group.properties file, which defines the database connection information and other common attributes for each group of DADX Web services. The following is an example of what the group.properties might look like for the new group:

```
# myapp_group group properties
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver
dbURL=jdbc:db2:sample
userID=
password=
namespaceTable=myapp.nst
autoReload=true
reloadIntervalSeconds=5

for Informix, use the following database driver and URL:
dbDriver=com.informix.jdbc.IfxDriver
dbURL=jdbc\:informix-sqli://::informixserver=
```

Figure 13. group.properties example

Customizing the group.properties file

The `group.properties` file is a standard Java properties file. The properties have the following values:

groupNamespaceUri
Optional - defines the target namespace that is used in the generated WSDL and XML schema files (XSD) for web services in this group.

initialContextFactory
Optional, but used with `datasourceJNDI`. Required for WebSphere connection pooling. Specifies the Java class name of the JDNI initial context factory that is used to locate the DataSource for the database. This property enables connection pooling. See "Using connection pooling to improve performance" on page 10 to learn how to set up connection pooling.

datasourceJNDI
Optional, but used with `initialContextFactory`. Required for WebSphere connection pooling. Specifies the JNDI name of the DataSource for the database. Used with `initialContextFactory`, defines a DataSource for the database connection. This property enables connection pooling. See "Using connection pooling to improve performance" on page 10 to learn how to set up connection pooling. Either the DataSource or the JDBC connection must be defined.

dbDriver
Optional. Specifies the Java class name of the JDBC driver for connecting to the database. Either the DataSource or the JDBC connection must be defined. If both are defined, then the DataSource is tried first. If WORF cannot obtain the DataSource, then it tries the JDBC.

dbURL
Optional, but used with dbDriver. Specifies the JDBC URL of the database.

userID
Optional. The default is the user ID under which the WORF executes, which can be the same user ID used for connecting to the database. Specifies the user ID for the database.

password
Optional, but used with user ID. Specifies the password for the database. Algorithms are available that can help you encode and decode your

password. See Appendix D, “Encoding algorithm” on page 87 for an algorithm to handle password encoding.

namespaceTable
Optional. Specifies the resource name of the namespace table. References a Namespace Table (NST) resource that defines the mapping from DB2 XML Extender DTDIDs to XML Schema (XSD) namespaces and locations. See Figure 23 on page 42 for an example of the NST file.

enableXmlClob
Optional. Specifies whether retrieveXML operations will use the CLOB-based XML Extender stored procedures. The value of true is the default. This option is included only for backward compatibility.

The next two options control resource loading and caching. If autoReload is absent or false, then resource reloading is not done and reloadIntervalSeconds is ignored. If autoReload is true, when WORF accesses a resource (such as one of the following files: DAD, DADX, DTD, NST), it compares the current time with the time at which the resource was previously loaded. If more than the value of `reloadIntervalSeconds` has passed, then WORF checks the file system for a newer version and reloads the changed resource. Automatic reloading is useful at development time, in which case `reloadIntervalSeconds` can be set to 0. If the Web services are in production, autoReload should either be set to false, or `reloadIntervalSeconds` should be set to a large value to avoid impacting server performance.

autoReload
Optional, but used with `reloadIntervalSeconds`. Specifies whether to reload a resource. Values can be true or false. The default is false.

reloadIntervalSeconds
Optional, but used with `autoReload`. Controls resource loading and caching. Specifies the integer automatic reloading time interval in seconds. The default is 0, which means that WORF will check for a newer resource on every request

Defining the Web service with the DADX file

The DADX file specifies a Web service using a set of operations that are defined by SQL statements, a list of parameters, and, optionally, DAD file references. Operations are similar to methods that you can invoke. The operations in a DADX Web Service can be defined by the following operation types:

- XML collection operations (requires DB2 XML Extender)
  - <retrieveXML> - generates XML documents
  - <storeXML> - stores XML documents
- SQL Operations
  - <query> - queries the database
  - <update> - performs an update, insert or delete operation on the database
  - <call> - calls stored procedures

Syntax of the DADX file

The DADX file is an XML document. The elements of the DADX are described in“DADX syntax definitions” on page 27 and in Figure 14 on page 27. The DADX schema is provided in Appendix B, “XML schema for the DADX file” on page 63. The numbers next to the nodes and elements in “DADX syntax definitions” on page 27 identify the child groupings. An element numbered 1.3.1 means that it is

the first child (column) of the third child (result_set_metadata) of the root element (DADX).

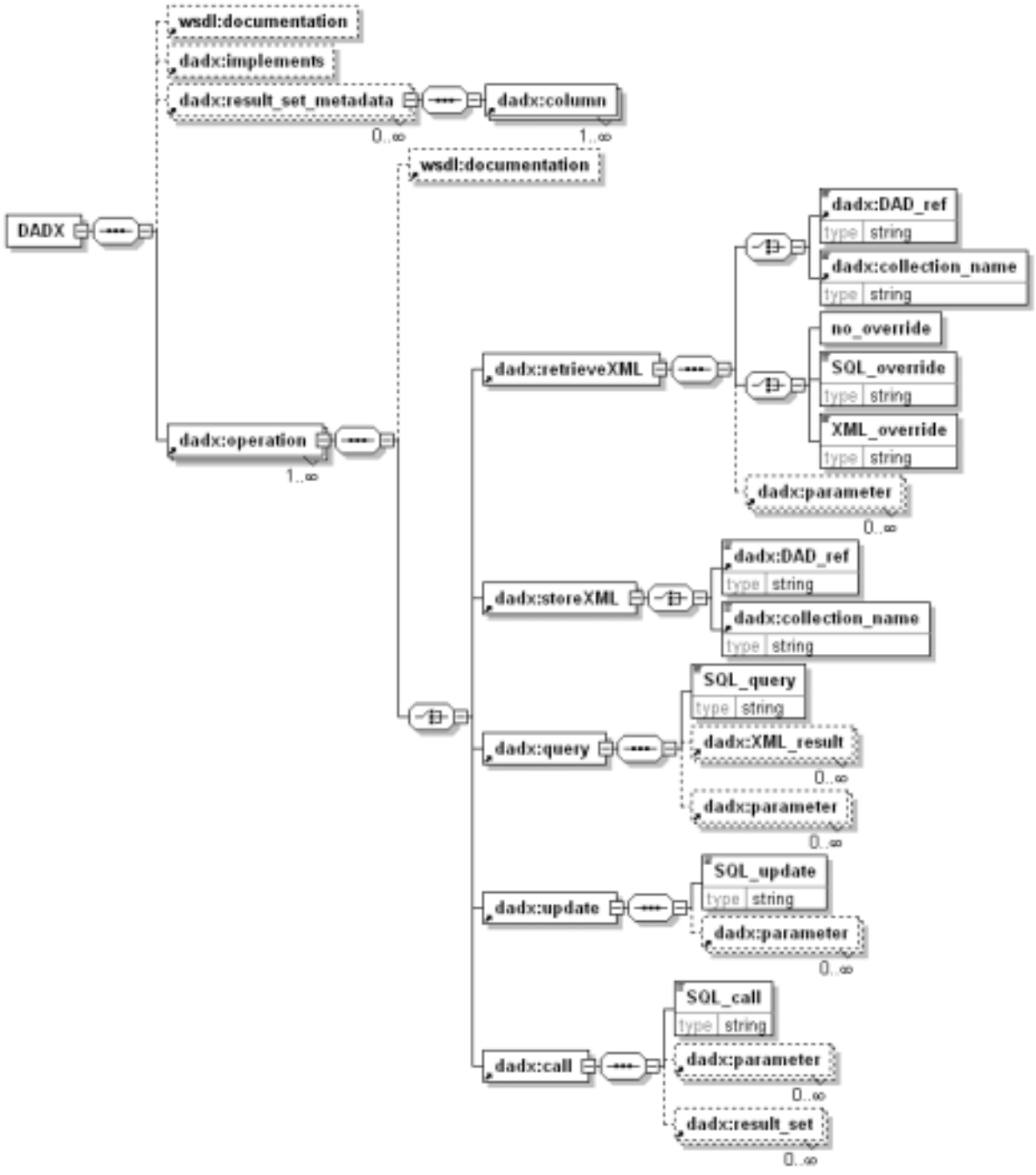

Figure 14. DADX syntax

1. Root element: <DADX>

Attributes:

xmlns:dadx

The namespace of the DADX.

xmlns:xsd
The namespace of the W3C XML Schema specification

xmlns:wsdl
The namespace of the W3C Web Services Definition Language specification

Children:

1.1 <wsdl:documentation>
Specifies a comment or statement about the purpose and content of the Web service. You can use XHTML tags.

1.2 <implements>
Specifies the namespace and location of the Web service description files. It allows the service implementor to declare that the DADX Web service implements a standard Web service described by a reusable WSDL document defined elsewhere; for example, in a UDDI registry.

1.3 <result_set_metadata>
Stored procedures can return one or more result sets which can be included in the output message. Metadata for a stored procedure result set must be defined explicitly in the DADX using the <result_set_metadata> element. At run-time, the metadata of the result set is obtained and it must match the definition contained in the DADX file.

Note: Therefore, only stored procedures that have result sets with fixed metadata can be invoked.
This restriction is necessary in order to have a well-defined WSDL file for the Web Service. A single result set metadata definition can be referenced by several <call> operations, using the <result_set> element. The result set metadata definitions are global to the DADX and must precede all of the operation definition elements.

Attributes:

name Identifies the root element for the result set.

rowname
Used as the element name for each row of the result set.

Children:

1.3.1 <column>
Defines the column. The order of the columns must match that of the result set returned by the stored procedure. Each column has a name, type, and nullability, which must match the result set.

Attributes:

name Required. Specifies the name of the column.

type Required if **element** is not specified. Specifies the type of column.

element
Required if **type** is not specified. Specifies the element of column.

as Optional. Provides a name for a column.

nullable
Optional. Nullable is either true or false. It indicates whether column values can be null.

1.4 <operation>

Specifies a Web service operation. The operation element and its children specify the name of an operation, and the type of operation the Web service will perform, such as compose an XML document, query the database, or call a stored procedure. A single DADX file can contain multiple operations on a single database or location. The following list describes these elements.

- Attribute:

  name A unique string that identifies the operation; the string must be unique within the DADX file. For example: `"findByColorAndMinPrice"`

- Children:

  Document the operation with the following element:

  1.4.1 <wsdl:documentation>
  Specifies a comment or statement about the purpose and content of the operation. You can use XHTML tags.

Specify the type of operation using one of the following child elements:

1.4.2 <retrieveXML>

Specifies to generate zero or one XML documents from a set of relational tables using the XML collection access method. Depending on whether a DAD file or an XML collection name is specified, the operation will call the appropriate XML Extender composition stored procedure

Children:

– Specify which of these stored procedures is used by passing either the name of a DAD file, or the name of the collection using one of the following elements:

1.4.2.1 <DAD_ref>
The content of this element is the name and path of a DAD file. If a relative path is specified for the DAD file, the current working directory is assumed to be the group directory.

1.4.2.2 <collection_name>
The content of this element is the name of the XML collection. Collections are defined using the XML Extender administration interfaces, as described in *DB2 XML Extender Administration and Programming*.

– Specify override values with one of the following elements:

1.4.2.3 <no_override/>
Specifies that the values in the DAD file are

not overridden. Required if neither <SQL_override> nor <XML_override> are specified.

1.4.2.4 <SQL_override>
Specifies to override the SQL statement in a DAD file that uses SQL mapping.

1.4.2.5 <XML_override>
Specifies to override the XML conditions in a DAD file that uses RDB mapping.

– Define parameters using the following element:

1.4.2.6 <parameter>
Required when referencing a parameter in an <SQL_override> or an <XML_override> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique within the operation.

Attributes:

name The unique name of the parameter. A parameter must have its contents defined by either an XML Schema element (a complex type) or a simple type.

element
Use the "element" attribute to specify an XML Schema element.

type Use the "type" attribute to specify a simple type.

kind Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:

- in

1.4.3 <storeXML>

Specifies to store (decompose) an XML document in a set of relational tables using the XML collection access method. Depending on whether a DAD file or an XML collection name is specified, the operation will call the appropriate XML Extender decomposition stored procedure. Children:

– Specify which of these stored procedures is used by passing either the name of a DAD file, or the name of the collection using one of the following elements:

1.4.3.1 <DAD_ref>
The content of this element is the name and path of a DAD file. If a relative path is specified for the DAD file, the current working directory is assumed to be the group directory.

1.4.3.2 <collection_name>

The content of this element is the name of an XML collection. Collections are defined using the XML Extender administration interfaces, as described in *DB2 XML Extender Administration and Programming*.

1.4.4 <query>

Specifies a query operation. The operation is defined by an SQL SELECT statement in the <SQL_select> element. The statement can have zero or more named input parameters. If the statement has input parameters then each parameter is described by a <parameter> element.

This operation maps each database column from the result set to a corresponding XML element. You can specify XML Extender user-defined types (UDTs) in the <query> operation, however this requires an <XML_result> element and a supporting DTD that defines the type of the XML column queried.

Children:

1.4.4.1 <SQL_query>

Specifies an SQL SELECT statement.

1.4.4.2 <XML_result>

Optional. Defines a named column that contains XML documents. The document type must be defined by the XML Schema element of its root.

Attributes:

name Specifies the root element of the XML document stored in the column.

element
Specifies the particular element within the column

1.4.4.3 <parameter>

Required when referencing a parameter in the <SQL_query> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique within the operation.

Attributes:

name The unique name of the parameter.
A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

element
Use the "element" attribute to specify an XML Schema element.

type Use the "type" attribute to specify a simple type.

kind Specifies whether a parameter passes

input data, returns output data, or does both. The valid values for this attribute are:

– in

1.4.5 <update>

The operation is defined by an SQL INSERT, DELETE, or UPDATE statement in the <SQL_update> element. The statement can have zero or more named input parameters. If the statement has input parameters then each parameter is described by a <parameter> element.

Children:

1.4.5.1 <SQL_update>

Specifies an SQL INSERT, UPDATE, or DELETE statement.

1.4.5.2 <parameter>

Required when referencing a parameter in the <SQL_update> element. Specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique with the operation.

Attributes:

name The unique name of the parameter.
A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

element

Use the "element" attribute to specify an XML Schema element.

type Use the "type" attribute to specify a simple type.

kind Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:

– in

1.4.6 <call>

Specifies a call to a stored procedure. The processing is similar to the update operation, but the parameters for the call operation can be defined as 'in', 'out', or 'in/out'. The default parameter kind is 'in'. The 'out' and 'in/out' parameters appear in the output message.

1.4.6.1 <SQL_call>

Specifies a stored procedure call.

1.4.6.2 <parameter>

Required when referencing a parameter in an <SQL_call> element. Specifies a parameter for an operation. Use a separate parameter element for

each parameter referenced in the operation. Each parameter name must be unique within the operation.

Attributes:

name The unique name of the parameter.
A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

element
Use the ″element″ attribute to specify an XML Schema element.

type Use the ″type″ attribute to specify a simple type.

kind Specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are:

- in
- out
- in/out

1.4.6.3 <result_set>

Defines a result set and must follow any <parameter> elements. The result set element has a name which must be unique among all the parameters and result sets of the operation, and must refer to a <result_set_metadata> element. One <result_set> element must be defined for each result set returned from the stored procedure.

Attributes:

name A unique identifier for the result sets in the SOAP response.

metadata
A result set meta data definition in the DADX file. The identifier must refer to the name of an element.

A simple DADX

This DADX contains one operation with an SQL query.

Figure 15 on page 34 shows a DADX file that defines a simple Web service:

```
<?xml version="1.0" encoding="UTF-8"?>
<DADX
  xmlns="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    Simple DADX example that accesses the SAMPLE database.
  </wsdl:documentation>
  <operation name="listDepartments">
    <wsdl:documentation>
      Lists the departments.
    </wsdl:documentation>
    <query>
      <SQL_query>SELECT * FROM DEPARTMENT</SQL_query>
    </query>
  </operation>
</DADX>
```

Figure 15. Simple DADX file

This simple DADX file defines a Web Service with a single operation named `listDepartments` which lists the contents of the DEPARTMENT table. The operation name identifies the Web service activity, and is similar to a method name in programming languages.

XML collection operations

You can generate or store XML documents with the <retrieveXML> or <storeXML> operations. These operations call XML Extender stored procedures and require a DAD file or an XML collection reference. These stored procedures generate or store XML documents using the mapping in a DAD file, or by referring to an enabled XML collection. See *DB2 XML Extender: Administration and Programming* to learn how to create a DAD file.

The following example shows a more complex DADX file that generates an XML document from a DAD file. It references a stored procedure using the <RetrieveXML> element. The <DAD_ref> element specifies the name of a DAD file.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                      xmlns="http://www.w3.org/1999/xhtml">
   Provides queries for part order information at myco.com.
   See
   <a href="../documentation/PartOrders.html" target="_top">PartOrders.html</a>
   for more information.
  </wsdl:documentation>
  <operation name="findAll">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders with their complete details.
    </wsdl:documentation>
    <retrieveXML>
       <DAD_ref>getstart_xcollection.dad</DAD_ref>
       <no_override/>
    </retrieveXML>
 </operation>
</DADX>
```

Figure 16. DADX files that generates an XML document

The Web service generated from this DADX file calls the dxxGenXML stored procedure and generates XML documents. The stored procedure refers to the `getstart_xcollection.dad` file to determine which tables to use when generating the XML documents, and the XML document structure.

Using overrides in the DADX

The DADX file can override XML values and SQL statements in DAD file using the <XML_override> and <SQL_override> elements. The type of override is determined by whether the DAD file uses SQL mapping or RDB mapping. If you do not need to override the DAD values, use the <no_override/> element, as shown in Figure 16.

The following example, which builds on the previous example, uses an SQL override statement.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                      xmlns="http://www.w3.org/1999/xhtml">
    Provides queries for part order information at myco.com.
    See <a href="../documentation/PartOrders.html" target="_top">
         PartOrders.html</a> for more information.
  </wsdl:documentation>
  <operation name="findAll">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders with their complete details.
    </wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad
</DAD_ref>
        <SQL_override>
           select o.order_key, customer_name, customer_email,
            p.part_key, color, quantity, price, tax, ship_id, date, mode
           from order_tab o, part_tab p,
             table(select substr(char(timestamp(generate_unique())),16)
               as ship_id, date, mode, part_key from ship_tab) s
           where p.order_key = o.order_key and s.part_key = p.part_key
               order by order_key, part_key, ship_id
       </SQL_override>
 </retrieveXML>
  </operation>
</DADX>
```

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                      xmlns="http://www.w3.org/1999/xhtml">
    Provides queries for part order information at myco.com.
    See <a href="../documentation/PartOrders.html" target="_top">
         PartOrders.html</a> for more information.
  </wsdl:documentation>
  <operation name="findAll">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders with their complete details.
    </wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad</DAD_ref>
        <SQL_override>
           select o.order_key, customer_name, customer_email,
            p.part_key, color, quantity, price, tax, ship_id, date, mode
           from order_tab o, part_tab p,
             table(select substr(char(timestamp(generate_unique())),16)
               as ship_id, date, mode, part_key from ship_tab) s
           where p.order_key = o.order_key and s.part_key = p.part_key
               order by order_key, part_key, ship_id
       </SQL_override>
   </retrieveXML>
  </operation>
</DADX>
```

Figure 17. DADX file that generates an XML document with SQL override

Although the SQL statement can be overridden, the new SQL statement must produce a result set that is compatible with the SQL mapping defined in the DAD file. For example, the column names that appear in the DAD file must also appear in the SQL override.

If the DAD file uses RDB node mapping, the RDB nodes have to be overridden using the <XML_override> element. RDB node elements define DB2 tables, columns, and conditions that are to contain XML data. The following example shows a DADX file that references an RDB node DAD file. The <XML_override> element content overrides the conditions specified in the DAD file. The override string can contain input parameters using the host variable syntax. The name and type of all parameters must be defined in a list of parameter elements that are uniquely named within this operation. In this example, the override parameter overrides the query by limiting the price to be greater than $50.00 and restricting the date to be greater than 1998-12-01.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                      xmlns="http://www.w3.org/1999/xhtml">
    Provides queries for part order information at myco.com.
    See <a href="../documentation/PartOrders.html" target="_top">
         PartOrders.html</a> for more information.
  </wsdl:documentation>
  <operation name="findByExtendedPriceAndShipDate">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders with an extended price greater than $50.00
      and a ship date later than 1998-12-01.
    </wsdl:documentation>
<retrieveXML>
      <DAD_ref>order_rdb.dad</DAD_ref>
        <XML_override>
           /Order/Part/ExtendedPrice > 50.00 AND
           Order/Part/Shipment/ShipDate > '1998-12-01'
       </XML_override>
    </retrieveXML>
  </operation>

</DADX>
```

Figure 18. DADX file that generates an XML document with XML override

Declaring and referencing parameters in the DADX

You can use parameters in each of the operations. The <SQL_query>, <SQL_update>, and <SQL_call> statements for the SQL operations can reference parameters, as can the XML and SQL overrides used in the <retrieveXML> and <storeXML> operations. The parameters are declared using the <parameter> element and have simple XML Schema types that correspond to the built-in SQL data types. The supported types are described in Table 4.

Table 4. Supported XML Schema and SQL types

XML Schema Simple Type	SQL Type
string	CHAR, VARCHAR
decimal	DECIMAL, NUMERIC
int	INTEGER
short	SMALLINT
float	FLOAT
double	REAL, DOUBLE PRECISION,
date	DATE

Table 4. Supported XML Schema and SQL types (continued)

XML Schema Simple Type	SQL Type
time	TIME
timestamp	TIMESTAMP

To reference a parameter, use a colon prefix. For example:

```
<SQL_query>
 select * from order_tab where customer_name =:customer_name
</SQL_query>
```

To define the parameter, use the <parameter> element: For example:

```
<parameter name="customer_name" type="xsd:string"/>
```

Each parameter that you reference must be defined with a <parameter> element. The name attribute for this element identifies the parameter and must be unique within the operation.

The following example shows a query operation that retrieves a set of relational data using an SQL SELECT statement. The statement contains one input parameter using the parameter syntax.

```
<operation name="findCustomerOrders">
    <wsdl:documentation>Returns all the orders for a given customer.
    </wsdl:documentation>
    <query>
      <SQL_query>select * from order_tab where customer_name =
        :customer_name</SQL_query>
      <parameter name="customer_name" type="xsd:string"/>
    </query>
  </operation>
```

Figure 19. Query operation with a parameter

The next example shows parameters in an SQL override used by a retrieveXML operation:

```
<operation name="findByColorAndMinPrice">
    <wsdl:documentation>Returns all the orders that have the specified color and
       at least the specified minimum price.
    </wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad
</DAD_ref>
      <SQL_override>
       select o.order_key, customer_name, customer_email,
         p.part_key, color, quantity, price, tax, ship_id, date, mode
       from order_tab o, part_tab p,
         table(select substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
         and color = :color and price >= :minprice
       order by order_key, part_key, ship_id
      </SQL_override>
      <parameter name="color" type="xsd:string">
      <parameter name="minprice" type="xsd:decimal">
    </retrieveXML>
  </operation>
```

Figure 20. SQL override used by a retrieveXML operation

The WHERE clause of the SQL statement is modified to include search conditions. The SQL override can include one or more parameters, identified by using a colon. In this example, `findByColorAndMinPrice` references `:color` and `:minprice`. The parameters are declared with a <parameter> element and have simple XML schema file (XSD) types that correspond to the built-in SQL data types.

DADX operation examples

The following samples show DADX files with the various operations.

Example1: Query operation

This example shows a Query operation, using the default tagging, and can be used without XML Extender. This operation selects all of the orders for a given customer. To run this sample, you need the sales_db XML Extender sample database.

```
<?xml version="1.0"?>
 <DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                         xmlns="http://www.w3.org/1999/xhtml">
       mycompany part orders service.
     </wsdl:documentation>
  <implements namespace="http://www.poia.org/part_orders.wsdl"
       location="http://www.poia.org/part_orders.wsdl"/>
<operation name="findCustomerOrders">
    <wsdl:documentation>Returns all the orders for a given customer.
    </wsdl:documentation>
    <query>
      <SQL_query>select * from order_tab
                    where customer_name = :customer_name
      </SQL_query>
      <parameter name="customer_name" type="xsd:string"/>
    </query>
  </operation>
```

Figure 21. DADX with Query operation

The input parameters must be defined by a list of parameter elements that are uniquely named within this operation. If more control over the mapping is required, then you can use a DAD file.

You can use Query operation to use the XML Extender user-defined types (UDT) and user-defined functions (UDF), which allow you to query, extract, and update data from an XML column that contains XML documents. These XML documents require that you create a DTD that defines the type of the <XML_result> element. This element specifies the name of the column and the root element of the XML document contained in it.

The following example shows a Query operation that uses the VARCHAR UDT declared by the <XML_result> element. The `retrieveOrders` operation retrieves all the XML order documents from the SALES_TAB table using the UDF db2xml.varchar. The documents are stored using the XML Extender UDT XMLVARCHAR:

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:dtd1="http://schemas.myco.com/sales/order.dtd"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>Queries part orders at myco.com.</wsdl:documentation>
<operation name="retrieveOrders">
    <wsdl:documentation>Retrieves all the Order documents.
    </wsdl:documentation>
    <query>
      <SQL_query>select db2xml.varchar(order) from sales_tab
      </SQL_query>
      <XML_result name="ORDER" element="dtd1:Order"/>
    </query>
  </operation>
</DADX>
```

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema"
      xmlns:dtd1="http://schemas.myco.com/sales/order.dtd"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>Queries part orders at myco.com.</wsdl:documentation>

<operation name="retrieveOrders">
    <wsdl:documentation>Retrieves all the Order documents.
    </wsdl:documentation>
    <query>
      <SQL_query>select db2xml.varchar(order) from sales_tab
      </SQL_query>
      <XML_result name="ORDER" element="dtd1:Order"/>
    </query>
  </operation>
</DADX>
```

Figure 22. Query operation with UDF and UDT

When you have XML documents in a column and you want the WSDL to refer to the type of this document, you can use the XML_result tag. In this example we specify that the ORDER column contains fragments of element *dtd1:Order*. The line
<XML_result name="ORDER" element="dtd1:Order"/> refers to the namespace declaration. XML Extender stores XML documents that have no namespaces and that are defined by DTDs. Web services use XML Schemas (XSD) instead of DTDs, and make use of namespaces. You associate a namespace with a DTD by making an entry in the namespace table. WORF adds the namespace when it retrieves an XML document and removes the namespace when it stores a document. WORF also automatically translates DTDs to XSD. The above line defines column information in file *order.dtd*. The specific declaration that it refers to is in the following example:

```
<?xml encoding="US-ASCII"?>
<!ELEMENT Order (Customer, Part+)>
<!ATTLIST Order key CDATA #REQUIRED>
...
```

To point to the DTD, use a namespace table file, (NST) file. For example:

```
<?xml version="1.0"?>
  <namespaceTable xmlns="http://schemas.ibm.com/db2/dxx/nst">
    <mapping dtdid="c:\dxx\samples\dtd\getstart.dtd"
      namespace="http://schemas.ibm.com/db2/dxx/samples/dtd/getstart.dtd"
      location="/dxx/samples/dtd/getstart.dtd/XSD"/>
    <mapping dtdid="getstart.dtd"
      namespace="http://schemas.myco.com/sales/getstart.dtd"
      location="/getstart.dtd/XSD"/>
    <mapping dtdid="order.dtd"
      namespace="http://schemas.myco.com/sales/order.dtd"
      location="/order.dtd/XSD"/>
  </namespaceTable>
```

Figure 23. NST file

You must reference this file in the `group.properties` file. See the example in to learn more about this file.

Example 2: Update operation

The following example shows an operation that updates the e-mail address of a customer for a given order:

```
  <operation name="updateOrderEmail">
    <wsdl:documentation>Updates the email address for an order.
    </wsdl:documentation>
    <update>
      <SQL_update>update order_tab set customer_email = :email
                  where order_key = :key</SQL_update>
      <parameter name="key" type="xsd:int"/>
      <parameter name="email" type="xsd:string"/>
    </update>
  </operation>
 </DADX>
```

Figure 24. Update operation

The Update operation can contain SQL INSERT, DELETE, or UPDATE statements in the <SQL_update> element.

Example 3: Call operation

These examples show Call operations that call stored procedures.

If your stored procedure returns result sets, you must define these result sets in the result_set_metadata tag in the DADX file so that WORF can generate the WSDL and XML schema files (XSD) for this web service operation. Figure 25 on page 43 shows the definition of a result set meta data that is referenced two times.

```
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<result_set_metadata name="employeeSalaryReport" rowName="employee">
  <column name="NAME" type="VARCHAR" nullable="true" />
  <column name="JOB" type="CHAR" nullable="true" />
  <column name="3" as="SALARY" type="DOUBLE" nullable="true" />
</result_set_metadata>
<operation name="twoResultSets">
<call>
<SQL_call>CALL TWO_RESULT_SETS (:salary, :sqlCode)
</SQL_call>
  <parameter name="salary" type="xsd:double" kind="in" />
  <parameter name="sqlCode" type="xsd:int" kind="out" />
  <result_set name="employees1" metadata="employeeSalaryReport" />
  <result_set name="employees2" metadata="employeeSalaryReport" />
  </call>
 </operation>
</DADX>
```

Figure 25. Definition of a resultset meta data referenced two times

You can also call a stored procedure using the following format:

```
<operation name="callProc1">
    <wsdl:documentation>Call the Proc1 stored procedure.
    </wsdl:documentation>
    <call>
      <SQL_call>
        CALL Proc1 (:x, :y, :z)
      </SQL_call>
      <parameter name="x" type="xsd:string" kind="in"/>
      <parameter name="y" type="xsd:int" kind="in/out"/>
      <parameter name="z" element="dtd1:Order" kind="out"/>
    </call>
</operation>
```

Figure 26. DADX with alternate Call operation

Example 4: RetrieveXML operation

The DADX file in Figure 27 on page 44 implements one retrieveXML operation using the stored procedure dxxGenXML.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                         xmlns="http://www.w3.org/1999/xhtml">
      mycompany part orders service.
    </wsdl:documentation>
  <operation name="findByColorAndMinPrice">
    <wsdl:documentation>Returns all the orders that have the specified color
        and at least the specified minimum price.</wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad</DAD_ref>
      <SQL_override>
       select o.order_key, customer_name, customer_email,
         p.part_key, color, quantity, price, tax, ship_id, date, mode
       from order_tab o, part_tab p,
         table(select substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
         and color = :color and price >= :minprice
       order by order_key, part_key, ship_id
      </SQL_override>
      <parameter name="color" type="xsd:string"/>
      <parameter name="minprice" type="xsd:decimal"/>
    </retrieveXML>
  </operation>
</DADX>
```

Figure 27. DADX with retrieveXML operation

The operation in this example generates XML documents based on the mapping in the `getstart_xcollection.dad` file. The operation specifies an SQL override which replaces the SQL statement defined in the DAD file and references two parameters in the override statement: `:color` and `:minprice`.

The DAD file for this example is found in the appendix of *DB2 XML Extender: Administration and Programming*.

Example 5: StoreXML operation

This example shows a DADX file that references a DAD using RDB_node mapping, `getstart_xcollection_rdb.dad`:

```
<?xml version="1.0"?>
 <DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                          xmlns="http://www.w3.org/1999/xhtml">
       mycompany part orders service.
     </wsdl:documentation>

  <implements namespace="http://www.poia.org/part_orders.wsdl"
       location="http://www.poia.org/part_orders.wsdl"/>

   <operation name="storeOrder">
     <wsdl:documentation>Stores an automotive part order.
     </wsdl:documentation>
     <storeXML>
       <DAD_ref>getstart_xcollection_rdb.dad</DAD_ref>
     </storeXML>
   </operation>
 </DADX>
```

Figure 28. DADX with StoreXML operation

The storeXML operation is implemented by the dxxInsertXML stored procedure if a <collection_name> element is used instead of a <DAD_ref> element. It performs the same operations as the dxxShredXML procedure, but uses the name of an XML collection instead of a DAD file.

Chapter 4. Packaging your DB2 Web service

Web archives files (WAR) are used to package, distribute, and install Web applications. A Web application is a collection or a library of servlets, Java Server Pages (JSPs), HTML files, XML files, graphics, and other Web resources that logically make up a Java-based Web application.

You generally package the files that comprise your Web application in a single WAR file for deployment. A WAR file with a DB2 Web service contains the web.xml server configuration files, the group configuration files and the DAD and DADX files. If the Web service creator wants to disable automatic deployment by using an IBM ConfigManager, the Web service creator would also provide a deployment descriptor file for the Web services.

Generating deployment descriptors

WebSphere 4.0 uses a custom ConfigManager (com.ibm.soap.server.XMLDrivenConfigManager) that disables deployment and undeployment at run-time. Instead, WebSphere reads a file that lists the deployed services when the application is started. To create this file, you must create a deployment descriptor file for each Web Service, or DADX file, which identifies configuration and deployment information. You insert the file into the XML file named dds.xml, which is read by the WebSphere ConfigManager when the Web application starts. When using the Apache SOAP configuration managers, the Web services get deployed on demand at run-time so you do not have to add them to the deployment descriptor file.

1. Verify that the `worf.jar` and `soap.jar` files are in your CLASSPATH.
2. Generate the deployment descriptor from a DADX file, using the class `com.ibm.etools.webservice.rt.dadx.Dadx2Dd` with the following parameters:
   - **-r** The resource name of the Web service relative to the group. This argument is required.
   - **-p** The group path. This argument is required.
   - **-n** The group name. This argument is required.
   - **-i** The name of the DADX name. This argument is optional. If present, the file must exist and be readable. If absent then the DADX file is read from standard input. To indicate standard input, use a dash (–).
   - **-o** The deployment descriptor file name. This argument is optional. If present the file must be writable and will be overwritten if it already exists. If absent then the deployment descriptor file is written to standard output. To indicate standard output, use a dash (–).

   For example, if the current directory is the *WEB-INF* directory, the following command reads the `ZipCity.dadx` file from the dxx_travel group and writes the deployment descriptor to the `dds` subdirectory:

   ```
   java com.ibm.etools.webservice.rt.dadx.Dadx2Dd -r ZipCity.dadx -p \travel
      -n \dxx_travel -i classes\groups\dxx_travel\ZipCity.dadx
       -o classes\dds\dxx_travel\ZipCity.isd
   ```
3. Copy the content of the newly created ISD file and add it to the `dds.xml` file in your Web application directory.
4. Restart the Web application.

Creating the WAR file

To create a WAR file, do the following:

1. Create the basic directory structure for the WAR file as in the example in Figure 29.

```
WEB-INF\lib\worf-servlets.jar
WEB-INF\web.xml
worf\class8.gif
worf\doTest.jsp
worf\doTestError.jsp
worf\doTestInputs.jsp
worf\doTestInputsError.jsp
worf\doTestInputsPrompt.html
worf\doTestMethods.jsp
worf\doTestMethodsError.jsp
worf\doTestResultPrompt.html
worf\error_tsk.gif
worf\field8.gif
worf\info_tsk.gif
worf\method8.gif
worf\outline8.gif
worf\properties8.gif
worf\style.css
```

Figure 29. Directory structure

You can find this worf directory hierarchy in the services.war file. These files are used when you run the TEST page. The worf-servlets.jar file is in the lib directory in the dxxworf.zip. The web.xml is the standard J2EE web.xml. An empty web.xml would look like the example in Figure 30.

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

Figure 30. Empty web.xml file

2. For each group,
   a. Create your group subdirectory (for example WEB-INF\classes\groups\myGroup) and include the group.properties and your DADX file in the subdirectory.
   b. Edit the WEB-INF\web.xml file to add the servlet and the servlet-mapping (for example, add a servlet name called *myGroup* and a URL-mapping called *myURLPath*). This is described in “Defining the web.xml and group.properties files” on page 23.
   c. See “Generating deployment descriptors” on page 47 for information on running the **Dadx2Dd** application. In the following example, the first instruction changes to the WEB-INF directory on a Windows platform. Then run the application. When the application completes, change back to the root directory.

```
cd WEB-INF
java com.ibm.etools.webservice.rt.dadx.Dadx2Dd
  -r ivt.dadx
  -p \myURLPath
```

```
 -n \myGroup
 -i classes\groups\myGroup\ivt.dadx
 -o classes\dds\myGroup\ivt.isd
cd ..
```

d. See “Generating deployment descriptors” on page 47 for information on the dds.xml file. Create or modify a dds.xml file to add the content of the generated file from Running the dadx2dd application on page 48 (for example WEB-INF\classes\dds\myGroup\ivt.isd). An empty dds.xml file looks like the following:

```
<?xml version='1.0'?>
<dds>
</dds>
```

3. Create the WAR file
   - from a command line by issuing the following command:

     ```
     jar -cvf minWORFwar.war WEB-INF worf
     ```

   - from WebSphere Studio by selecting **File** from the menu; then select **Export**, and then **WAR file**. Select the project name your Web application is in and specify a file name.
4. Deploy the WAR file as described in the sample installs for WebSphere Application Server or Apache Jakarta Tomcat (for example, with myContext as the Web application context).
5. Verify that you have created the WAR file correctly by running the TEST page. For example, issue the following command from your browser:

   ```
   http://<your WebAppServer>/myContext/myURLPath/ivt.dadx/TEST
   ```

   See “Testing the Web service”.

Testing the Web service

You can test Web services from the Web Service Sample Page. This page displays sample Web services that are provided with WORF. You can create your own welcome page for your Web services similar to the index.html page in the WORF samples. If you do this, you will need to include the URLs to all of the TEST pages of the DADX files.

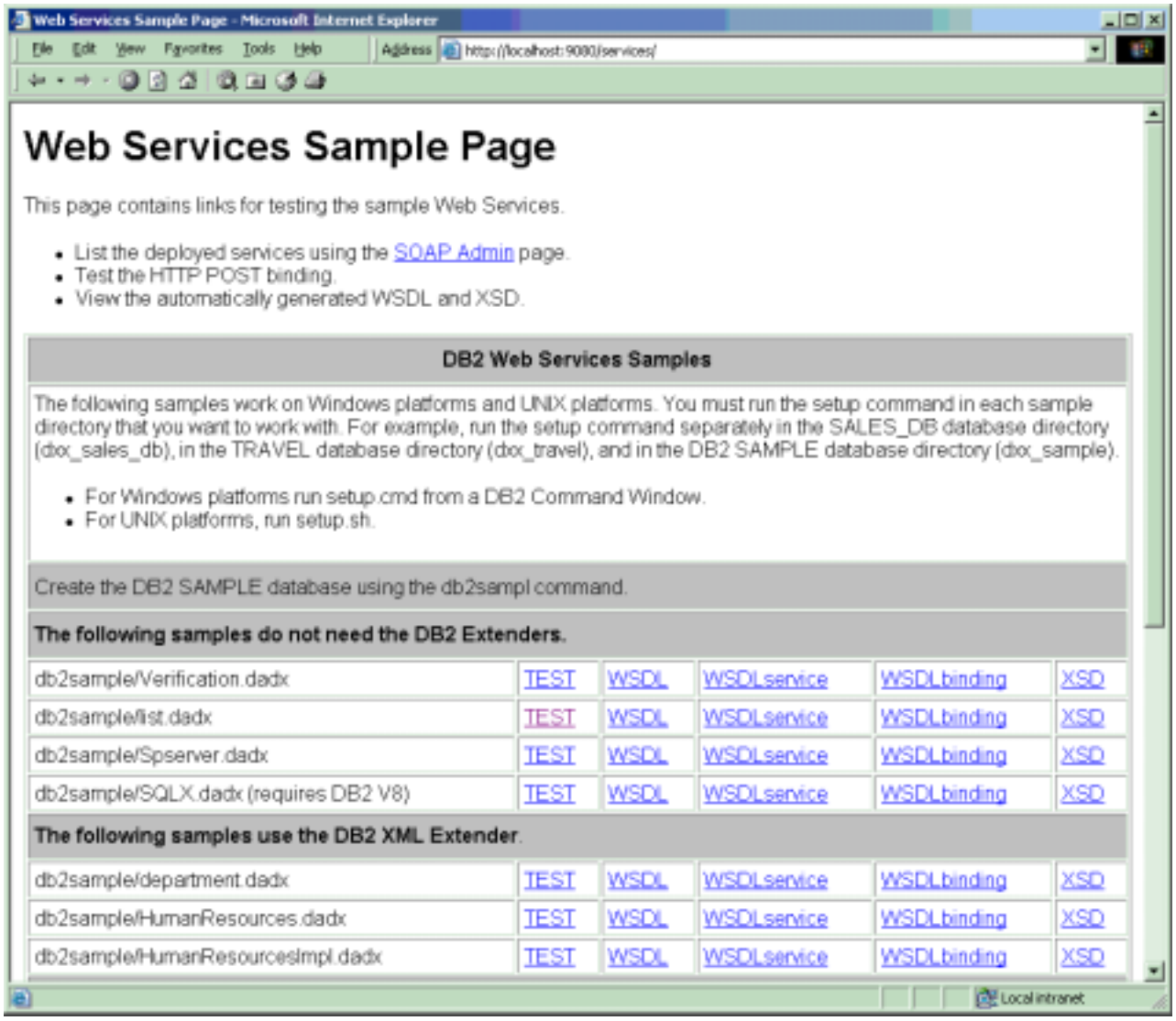

Figure 31. Web services sample page

Each Web service has its own line, with links to functions such as displaying the generated WSDL file, the generated XML schema (XSD) files, and the generated test page for the Web service. The TEST column has a link to the Web Service Test Page, described in “Scenario” on page 15.

To add and test your own Web service:

- Open the HTML file for the Web services Sample Page:

  `http://<your WebAppServer>/myContext/index.html`

  The Web services are in a table format, one row for each Web service.
- Create a new row for your Web service, modeling it on the HTML tagging provided for the product samples.

Chapter 5. Generating XSD and WSDL files from the DADX

The DADX file provides information that is used to generate the following types of files:

- The XML schema (XSD) file
  - The Web service has an XSD.
  - Any XML documents stored in columns have XSDs, but these are converted from the DTD.
- The WSDL document that describes the Web service
- The deployment and reusable WSDL parts for Universal Description, Discovery, Integration (UDDI) registration.

The following sections describe how to generate these files.

Converting a DTD to an XSD file

XML Extender currently uses DTDs to define document structure, while WSDL uses XML schemas (XSD files). WORF automatically creates an XML Schema (XSD) file. You must add an entry to the namespace table (NST file) to define the namespace associated with a DTD, and to enable conversion of the DTD to XSD.

You can request an XSD file by using the following URL syntax:

```
http://host/path/dtd_file.dtd/XSD
```

For example:

```
http://localhost:8080/services/sample/order.dtd/XSD
```

. In this case, the order.dtd file must be in WEB-INF\groups\dxx_sample.

WORF and the XML Extender locate DTDs through their DTDID. The DTDID is either a file name or the key value in the DTD_REF table of your database. XML Extender creates the DTD_REF table when you enable your database. The best practice is to store DTDs in the DTD_REF table since file locations may change when you move your Web application to another machine. The following extract from the Windows 2000 setup-xcollection.cmd file in the SALES_DB example shows how to insert DTDs into the DTD_REF table:

```
db2 "connect to SALES_DB"
rem Insert DTDs
db2 "insert into db2xml.dtd_ref values('getstart.dtd',
 db2xml.XMLClobFromFile('%CD%\getstart.dtd'),
 0, 'user1', 'user1', 'user1')"
db2 "insert into db2xml.dtd_ref values('order.dtd', db2xml.XMLClobFromFile('%CD%\order.dtd'),
 0, 'user1', 'user1', 'user1')"
```

Generating WSDL from the DADX

The DADX document contains the information required to implement the Web service. It also contains the information required to generate the WSDL document that describes the Web service.

WSDL is an XML vocabulary that is used to describe the interface of business services. It is used in publishing services to a UDDI registry. WSDL allows

development tools to programmatically create requester and provider code for use in binding to a Web service, as well as enabling preconditioned applications to dynamically bind to a Web service. It can be used to specify the data required for requests and responses, and uses XML Schema for precise data definition.

To generate the WSDL, submit the following URL:

`http://host_name:port/webapp_name/group_name/dadx_file.dadx/WSDL`

WORF dynamically generates the WSDL document. You can publish this in UDDI or some other Web service directory.

Using the running example, you can submit the following URL:

```
http://localhost:8080/services/sales/PartOrders.dadx/WSDL
```

Generating WSDL for UDDI registration

In the UDDI best practices document for using WSDL with UDDI Registries, it is recommended that the WSDL document be split into two parts: the deployment and reusable parts.

The deployment part includes the <service> element which contains the URLs where the service is deployed. The deployment part imports the reusable part which contains the other top level WSDL elements.

The reusable part corresponds to a UDDI <tModel> element and the deployment part corresponds to a UDDI <businessService>. Within the <businessService> element, each WSDL <port> element corresponds to a UDDI <bindingTemplate> element.

To learn more about UDDI and Web service registration, see the Universal Description, Discovery and Integration of Business for the Web site

To generate the WSDL parts, submit a URL with the WSDL path information:

- To generate the deployment part, submit a URL with `WSDLservice` key words:
  `http://host_name:port/webapp_name/group_name/DADX_file.dadx/WSDLservice`
- To generate the reusable part, submit a URL with `WSDLbinding` key words:
  `http://host_name:port/webapp_name/group_name/DADX_file.dadx/WSDLbinding`

The following example demonstrates how to generate the deployment and the reusable parts of the WSDL document. To generate the deployment part, submit a URL with `WSDLservice` command, as shown in the example:

```
http://localhost:8080/sales_db/part_orders.dadx/WSDLservice
```

To generate the reusable part, submit a URL with `WSDLbinding` command, as shown in the example:

```
http://localhost:8080/sales_db/part_orders.dadx/WSDLbinding
```

The example in “Generating WSDL for UDDI registration” deals with the case in which the service implementor creates a Web service that is unique to a company. However, one of the usage scenarios that UDDI is designed to handle is the case in which a standards body or vendor defines a Web service interface tModel, and service implementors use it. For example, the airline industry might define a Web service that provided flight schedules, which airlines could then implement. UDDI

provides a way for users to search for all registered services that implement a given tModel, so a travel planning application could locate all the airline flight schedule services.

Use the DADX <implements> element to declare that the service implements a Web Service described by a reusable WSDL document defined elsewhere. An example of an <implements> element is shown in Figure 32.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://schemas.ibm.com/db2/dxx/dadx"
  ...
  elementFormDefault="qualified">
  <import namespace="http://schemas.xmlsoap.org/wsdl/"
   schemaLocation="wsdl.xsd"/>
        ....
   <element name="DADX">
    <annotation>
      <documentation>
        Defines a Web Service.
        The Web Service is described by an optional WSDL documentation element.
        ....
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref="wsdl:documentation" minOccurs="0"/>
        <element ref="dadx:implements" minOccurs="0"/>
        <element ref="dadx:result_set_metadata" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="dadx:operation" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
 ...
  <element name="implements">
    <annotation>
      <documentation>
        Defines the namespace and location of a set of WSDL bindings
        defined elsewhere. This information is imported into the
        WSDL document generated for this Web Service.
      </documentation>
    </annotation>
    <complexType>
      <attribute name="namespace" type="anyURI" use="required"/>
      <attribute name="location" type="anyURI" use="required"/>
    </complexType>
  </element>
...
</schema>
```

Figure 32. Element <implements>

Appendix A. DADX environment checker

Document Access Definition Extension (DADX) provides a way for Web Services Object Runtime Framework (WORF) to support database access. WORF is an extension to Apache SOAP 2.2 or later that makes Web services easier to develop and use. One of the key features of WORF is the resource-based deployment. Resource-based deployment means that resource files, such as DADX files, can be used to develop Web services with WORF. Resource files describe the Web services to WORF, so that WORF can generate the appropriate Web services from these files.

The resource files can be created by the user and must conform to specific syntax and semantic rules. See “Defining the Web service with the DADX file” on page 26 for information on the rules. The resource files can make references to each other (for example, a DADX file can contain references to DAD files). These references must be correct so that the Web services can be deployed properly.

Purpose and operation of the DADX environment checker

The DADX environment checker performs different syntax and semantic checks on the NST, DAD and DADX files used to create and run Web services with WORF. Use the DADX environment checker to help minimize the number of errors that occur when deploying Web services with WORF.

The DADX environment checker is a Java application that is called from the command line. When invoked, it produces an output file that contains errors, warnings, and success indicators. The name of the output text file is user-defined. If no name is specified, the standard output is used. See the syntax of the command in “Running the DADX environment checker” on page 56.

Installing the DADX environment checker

The DADX environment checker is included in dxxworf.zip, in the \tools\lib subdirectory. The jar files containing the code for this tool are CheckersCommon.jar and DADXEnvChecker.jar. Make sure that you have a JRE or JDK Version 1.3.1 or later, installed on your system. Update your classpath to include all of the following archives:

- CheckersCommon.jar, DADXEnvChecker.jar and worf.jar, included in the \tools\lib directory in dxxworf.zip
- xmlParserAPIs.jar and xercesImpl.jar, included in the binary distribution for Xerces-J 2.0.2 downloadable at http://xml.apache.org
- soap.jar, included in the binary distribution for SOAP 2.2 downloadable at http://xml.apache.org
- activation.jar, included in the binary distribution for JavaBeans Activation Framework 1.0.1, downloadable at http://java.sun.com
- mail.jar, included in the binary distribution for JavaMail 1.2 downloadable at http://java.sun.com
- servlet.jar, included in the distribution for Jakarta Tomcat Version 3.2.x through 4.0.3 downloadable at http://www.apache.org/
- db2java.zip, included in the /java directory located in the directory where you installed DB2

For example, if you are running in the Windows environment, you must set your classpath by issuing the following command appropriate to your own directory structure:

```
set classpath=
c:\dxxworf\tools\lib\CheckersCommon.jar;
c:\dxxworf\tools\lib\DADXEnvChecker.jar;
c:\dxxworf\tools\lib\worf.jar;
c:\Xerces\xmlParserAPIs.jar;
c:\Xerces\xercesImpl.jar;
c:\soap2-2\soap.jar;
c:\jaf-1.0.1\activation.jar;
c:\javamail-1.2\mail.jar;
c:\jakarta-tomcat-3.2.2\lib\servlet.jar;
c:\Program Files\sqllib\java\db2java.zip;
```

Running the DADX environment checker

The DADX environment checker is a Java program, that can run on JDK version 1.3.1 and later. To run the DADX environment checker, execute the following command (on a single line):

```
 java com.ibm.etools.webservice.util.Check_install
[-srv] [-schdir pathToSchemasDir] [-sch schemaLocations] [-out outputFile]
 fileToCheck
```

For example if you extracted dxxworf.zip in directory c:\dxxworf, you would type the following to run the DADX checker on the resource files contained by the c:\tomcat\webapps\services directory:

```
 java com.ibm.etools.webservice.util.Check_install
 -srv -schdir c:\dxxworf\schemas
-out myOutputFile.txt c:\tomcat\webapps\services
```

Parameters

Here are the parameters that can be used to run the DADX environment checker:

-schdir *pathToSchemasDir*
: Specifies the absolute path to the directory where the schemas used for validating NST and DADX files are stored

-sch *schemaLocations*
: Specifies a list of schemas to be used by the parser to validate the files. The DADX checker allows the user to specify the value of a property of the Xerces parser. This property can be used to specify the location of XML schemas needed to perform the validation of the files being parsed. You specify the location of a schema by providing the name of the target namespace of the schema (for example: *http://myschema*) followed by the actual location of the schema. It could be a path in the file system (for example, *c:\dir\schema1.xsd*) or a valid URL. But the XML documents themselves can contain declarations of schema locations. The schemaLocation attribute is used in an XML document to provide this information. Here is an example of the beginning of an XML document:

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report
http://www.example.com/Report.xsd">
```

For a particular namespace, the parser will use the schema location defined using the property of the parser, even if the schemaLocation attribute

defines another schema location for the same namespace. The syntax for schemaLocations is the same as for schemaLocation attributes in instance documents: for example, *http://www.example.com file_name.xsd*. The user can specify more than one XML Schema: for example, -sch *http://www.example_1.com file_name_1.xsd http://www.example_2.com file_name_2.xsd*

-out *outputFile*
Specifies the output text file name; if omitted, the standard output is used.

-srv Indicates that the checks must be performed on all of the NST, DAD and DADX files found under the Web services module directory (for example c:\tomcat\webapps\services) passed as the *fileToCheck*. If this option is not used, then the checks are performed only on the DADX file that is passed as the file to check and on the related data contained in other resource files. For example, the DAD files referred to in this DADX file will be checked and then the DTDIDs referred to in these DAD files will be checked in the NST file. And only the data related to the DADX file will be checked in the NST file and in the web.xml file.

fileToCheck
If parameter *-srv* is not used, then the value of *fileToCheck* is the DADX file that is checked. If parameter *-srv* is used then the *fileToCheck* value is the root directory of the Web services module; for example, the root directory of an unzipped .war file as *services* for the services.war module.

-help Displays command line option information

-version
Displays version information

Sample files

A sample file can be found in the \tools\samples directory from dxxworf.zip. *DADXEnvChecker_sample.txt* is an output text file showing the results of the checks performed on a Web services module. This file is generated by the DADX environment checker and uses the file name you gave in the -out parameter.

Indicating errors and warnings in the output text file

When the **-srv** parameter is used, errors, warnings, and success indicators are grouped together in paragraphs. Each paragraph is associated with a checked file. The results of checking each file are displayed in the output file if you indicated a file name, or in the standard output device if no filename is indicated.

The paragraphs are grouped together according to the path, or subdirectories, in directory **groups**. Here is an excerpt of an output text file showing the error messages corresponding to the checks performed in files sales_db.nst and getstart_xcollection.dad belonging to group /groups/dxx_sales_db:

```
## Checking group: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sales_db
## Checking NST file: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sales_db\sales_db.nst
INFO. Line 5: file "c:\dxx\samples\dtd\getstart.dtd" is accessible.
ERROR. Line 12: file "wrongDtd.dtd" CANNOT be found
either in the file system or in the database.
INFO. Line 8: file "getstart.dtd" is accessible.
## Checking DAD file: c:\tomcat\webapps\services\WEB-
```

```
INF\classes\groups\dxx_sales_db\getstart_xcollection.dad
WARNING. Line 4: DTDID "dtd_.dtd" CANNOT be found in the DTD_REF table.
INFO. Line 9: the DTDID "c:\dxx\samples\dtd\getstart.dtd"
has been declared in the NST file.
```

Errors, warnings and success messages can begin with a line number if the error or warning or success event is related to a specific line. The line numbers in the output text indicate the line numbers where the checked elements associated with the messages were found in the files. There is no order related to the output within a paragraph.

Checks performed by the DADX environment checker

On invocation, when the **-srv** parameter is used, the DADX environment checker first performs checks on the web.xml file within directory web-inf. Then, the DADX environment checker performs checks on the following types of files found in each group directory in the web-inf\classes\groups directory:

- NST files
- DAD files
- DADX files

On invocation without the **-srv** parameter, the DADX environment checker first performs checks on the DADX file that is passed as the file to check. Then, the DADX environment checker checks the DAD files that are referenced in this DADX file. It also performs checks on the NST file of the group to which the DADX file belongs. The DADX environment checker eventually checks the web.xml file within the web-inf directory containing the DADX file.

Connection to the database with which the group is associated

For some checks on NST and DADX files, the DADX environment checker tries to query the database with which the group is associated. Therefore, before starting checking the files of a group, the checker will try to establish a connection to the database using data contained in file **group.properties**.

The following example shows an error message provided by the checker because the connection to the database failed:

```
Checking group: c:\test\jakarta-tomcat-3.2.2
##Checking group: c:\tomcat\webapps\services
\WEB-INF\classes\groups\dxx_travel
WARNING. Connection error [IBM][CLI Driver]
SQL1013N The database alias name or database name
"TRAVELLL" could not be found.
SQLSTATE=42705
```

Checks performed on the web.xml

The DADX environment checker checks the **web-inf\web.xml** file under the root directory of the Web Service module, which is services in our example.

Here is an excerpt of the web.xml file:

```
<servlet>
<servlet-name>dxx_sales_db</servlet-name>
<servlet-class>com.ibm.etools.webservice.rt.dxx.servlet.DxxInvoker
</servlet-class>
<init-param>
<param-name>faultListener</param-name>
```

```
<param-value>org.apache.soap.server.DOMFaultListener
</param-value>
</init-param>
<load-on-startup>-1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dxx_sales_db</servlet-name>
<url-pattern>/sales/*</url-pattern>
</servlet-mapping>
```

The <servlet-class> tags, which are direct children of the <servlet> tags must have as a value either com.ibm.etools.webservice.rt.isd.servlet.IsdInvoker or com.ibm.etools.webservice.rt.dxx.servlet.DxxInvoker. When their values are different, the checker provides an error message. The following example shows the results of the checks performed on <servlet-class> tags in a web.xml document:

```
INFO. Line 21: servlet class for
servlet "dxx_sales_db" is a correct servlet class.
ERROR. Line 31: servlet class
"com.ibm.etools.webservice.rt.dxx.servlet.OtherInvoker"
for servlet "dxx_sample"
is NOT a correct servlet class.
INFO. Line 41: servlet class
for servlet "dxx_travel"
is a correct servlet class.
```

Each <servlet-mapping> tag contains a <servlet-name> tag with a value that must be the same as the value of the <servlet-name> tag of a <servlet> tag. If this is not the case the checker provides an error message as shown in the following example:

```
ERROR. There is no <servlet> tag declaring servlet
"isd_demos" mapped at line 50.
```

On the other hand, each <servlet> tag must have a corresponding <servlet-mapping> tag with the same servlet name as theirs. If a <servlet> tag has no corresponding <servlet-mapping> tag, the checker provides the following kind of message:

```
ERROR. There is no <servlet-mapping> tag
for servlet "dxx_travel" declared
at line 40.
```

Each <servlet-mapping> tag also contains a <url-pattern> tag with a value that must be unique. If two <url-pattern> tag have the same value, the checker provides an error message as shown in the following example:

```
ERROR. Line 56: "/sales/*" is already
declared as the URL pattern for servlet "isd_demos"
(see line 50).
```

Checks performed on NST files

In each group directory there might be an NST file. NST files declare the namespace table of the group. They contain mappings between DTD identifiers and the namespace and location of the XML schema that is automatically generated from the DTD.

Here is an excerpt of an NST file:

```
<namespaceTable
xmlns="http://schemas.ibm.com/db2/dxx/nst">
<mapping dtdid="c:\dxx\samples\dtd\getstart.dtd"
    namespace="http://schemas.ibm.com/db2/dxx/samples/dtd/getstart.dtd"
```

```
    location="/dxx/samples/dtd/getstart.dtd/XSD"/>
<mapping dtdid="getstart.dtd"
    namespace="http://schemas.myco.com/sales/getstart.dtd"
    location="/getstart.dtd/XSD"/>
```

The DADX environment checker first validates NST files against their schema **nst.xsd**. Here is an example of a validation error reported by the checker:

```
ERROR. Validation error, in
"file:///c:/tomcat/webapps/services/WEB-
INF/classes/groups/dxx_sales_db/sales_db.nst",
line 8, column 35. cvc-complex-type.2.4.a:
Invalid content starting with element 'mappin'.
The content must match
'("http://schemas.ibm.com/db2/dxx/nst":mapping){0-UNBOUNDED}'.
ERROR. Validation error, in
"file:///c:/tomcat/webapps/services/WEB-
INF/classes/groups/dxx_sales_db/sales_db.nst",
line 17, column 32. cvc-complex-type.4: Attribute 'dtdid'
must appear on element 'mapping'.
ERROR. Validation error, in
"file:///c:/tomcat/webapps/services/WEB-
INF/classes/groups/dxx_sales_db/sales_db.nst",
line 17, column 32. Duplicate unique value
[ID Value: /order.dtd/XSD] declared for identity constraint
of element "namespaceTable".
```

Eventually, the checker checks that the dtdid attributes of the <mapping> elements are either:

- a correct path in the file system, or
- a value stored in column DTDID in the db2xml.DTD_REF table

The following example shows the results of the checks on the <mapping> elements of an NST file:

```
INFO. Line 5: file
"c:\dxx\samples\dtd\getstart.dtd" is accessible.
ERROR. Line 14: file
"wrongDtd.dtd" CANNOT be found either in
the file system or in the database.
```

Checks performed on DAD files

The Document Access Definition (DAD) file is an XML file that is supported in DB2 XML Extender. The DAD associates XML documents to DB2 database tables through two alternative access and storage methods: XML columns and XML collections.

The following example shows the beginning of a DAD file:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<dtdid>c:\dxx\samples\dtd\getstart.dtd</dtdid>
<validation>NO</validation>
<Xcollection>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Order SYSTEM
 "c:\dxx\samples\dtd\getstart.dtd"
</doctype>
<root_node>
<element_node name="Order">
...
```

The DADX environment checker first checks that the DAD file is valid against its DTD **dad.dtd**. You must ensure that the path to dad.dtd specified in the DOCTYPE declaration of the DAD is correct.

Then the checker gets the value of the <dtdid> tag if it is present. If the value of this tag does not match a value stored in column DTDID in the db2xml.DTD_REF table, then the checker issues a warning. If the <validation> tag in the DAD contains a value of *YES*, then the checker issues an error message:

```
## Checking DAD file:
c:\tomcat\webapps\services\WEB-INF
\classes\groups\dxx_sales_db\order.dad
ERROR. Line 4: DTDID "wrongDtd.dtd"
CANNOT be found in the DTD_REF table.
```

Then the checker determines whether the DAD file declares an Xcollection or an Xcolumn. If it declares an Xcollection, the DTD specified in the <doctype> element is extracted. The DADX environment checker checks that this DTD is declared in the NST file.

The following example shows the results of the checks of an Xcolumn and an Xcollection DAD belonging to the same group:

```
## Checking DAD file: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sales_db\getstart_xcolumn.dad
INFO. Line 4: DTDID "getstart.dtd" was found in the DTD_REF table.

## Checking DAD file: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sales_db\order-public.dad
INFO. Line 4: DTDID "order.dtd" was found in the DTD_REF table.
ERROR. Line 8: the DTDID "order.dtd" has NOT been
declared in the NST file.
```

You can also perform other checks on the DAD files by using the DAD checker. The DAD checker is a separate tool that is also contained in the /tools/lib directory in dxxworf.zip. For more information, see the documentation on this tool at http://www-4.ibm.com/software/data/db2/extenders/xmlext/download/beta/dadchecker.html.

Checks on DADX files

Document Access Definition Extension (DADX) is a technology for rapidly creating Web services that access databases. DADX lets you define Web service operations using the standard SQL statements SELECT, INSERT, UPDATE, DELETE, and CALL, and the DB2 XML Extender stored procedures.

Here is an excerpt of a DADX file:

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<operation name="find">
<documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
Returns the parts from order #1 with price > 20000.
    </documentation>
<retrieveXML>
<DAD_ref>getstart_xcollection.dad</DAD_ref>
<no_override/>
</retrieveXML>
</operation>
<operation name="findByMinPrice">
<retrieveXML>
<collection_name>
getstart_xcollection.dad
```

```
</collection_name>
<no_override/>
<parameter name="minprice"
type="xsd:decrimal"/>
</retrieveXML>
</operation>
```

The DADX environment checker first validates the DADX file against its schema, dadx.xsd. Then the checker gets the values of the <DAD_REF> or <collection_name> tags and it checks that the values of these tags are:

- for <DAD_REF> tags, a correct path to a DAD file in the file system
- for <collection_name>, the name of an enabled collection, which is a value stored in column COL_NAME from table db2xml.xml_usage.

The following example shows the results of the checks performed on a DADX file:

```
## Checking DADX file: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sales_db\PartOrders.dadx
ERROR. Validation error, in "file:///c:/tomcat/webapps/services/WEB-
INF/classes/groups/dxx_sales_db/PartOrders.dadx",
line 8, column 67. cvc-complex-type.2.4.c:
The matching wildcard is strict, but no declaration
can be found for element 'as'.
INFO. Line 16: for operation "find",
DAD "getstart_xcollection.dad" was found.
ERROR. Line 26: for operation "findAll",
DAD "non_existing_dad.dad" was NOT found.
INFO. Line 44: for operation "findByColor",
DAD "getstart_xcollection.dad" was found.
INFO. Line 65: for operation "findByMinPrice",
DAD "getstart_xcollection.dad" was found.
```

If an <operation> tag has no <DAD_REF> or <collection_name> tag as a child, the checker issues a message indicating that no check was performed for this particular operation, as shown in the following example:

```
## Checking DADX file: c:\tomcat\webapps\services\WEB-
INF\classes\groups\dxx_sample\HelloSample1.dadx
INFO. Line 10: no <DAD_ref> or <collection_name>
elements to check for operation "listDepartments".
```

The DADX environment checker also checks if WORF will be able to find a deserializer for the parameters declared in the DADX file. A deserializer reconstructs XML messages received across a network connection into the specified variable or object. For every <parameter> tag, the value of its type attribute must be a type that can be deserialized. If no deserializer can be found for a particular type, the checker provides an error message as shown in the following example:

```
ERROR. Line 13: no deserializer was found to deserialize a
 "http://www.w3.org/2001/XMLSchema:ssstring", using encoding
"http://schemas.xmlsoap.org/soap/encoding/".
```

Appendix B. XML schema for the DADX file

The following schema, `dadx.xsd`, describes the DADX.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
 targetNamespace="http://schemas.ibm.com/db2/dxx/dadx"
 xml:lang="en"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 elementFormDefault="qualified">
 <import namespace="http://schemas.xmlsoap.org/wsdl/"
schemaLocation="wsdl.xsd"/>
 <annotation>
    <documentation>
      A Document Accession Definition Extension (DADX) document
defines a Web Service that is implemented by operations that
 access a relational database and that optionally use stored procedures,
 types and functions provided by the DB2 XML
 Extender.
    </documentation>
 </annotation>
```

Figure 33. DADX schema (Part 1 of 10)

```
<element name="DADX">
    <annotation>
      <documentation>
        Defines a Web Service.
        The Web Service is described by an optional WSDL documentation element.
        The Web Service may implement a set of WSDL bindings defined elsewhere.
        The Web Service consists of one or more uniquely named operations.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref="wsdl:documentation" minOccurs="0"/>
        <element ref="dadx:implements" minOccurs="0"/>
        <element ref="dadx:result_set_metadata" minOccurs="0"
        maxOccurs="unbounded"/>
        <element ref="dadx:operation" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <key name="result_set_metadataNames">
      <selector xpath="dadx:result_set_metadata"/>
      <field xpath="@name"/>
    </key>
    <keyref name="resultSetMetatdata" refer="dadx:result_set_metadataNames">
      <selector xpath="dadx:operation/dadx:call/dadx:result_set"/>
      <field xpath="@metadata"/>
    </keyref>
    <unique name="operationNames">
      <selector xpath="dadx:operation"/>
      <field xpath="@name"/>
    </unique>
 </element>
```

Figure 33. DADX schema (Part 2 of 10)

```
<element name="implements">
    <annotation>
      <documentation>
        Defines the namespace and location of a set of WSDL bindings
        defined elsewhere. This information is imported into the
        WSDL document generated for this Web Service.
      </documentation>
    </annotation>
    <complexType>
      <attribute name="namespace" type="anyURI" use="required"/>
      <attribute name="location" type="anyURI" use="required"/>
    </complexType>
 </element>
 <element name="result_set_metadata">
    <annotation>
      <documentation>
        Defines the metadata for a result set returned by a stored procedure.
        Each metadata element defines a global element
        in the WSDL for the Web Service.
        The metatdata name defines the name of its global element.
        The metadata rowName defines the element name of each row.
        The metadata contains one or more column definitions.
      </documentation>
    </annotation>
      <complexType>
      <sequence>
        <element ref="dadx:column" minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="rowName" type="NCName" use="required"/>
    </complexType>
     </element>
```

Figure 33. DADX schema (Part 3 of 10)

```
<element name="column">
      <annotation>
       <documentation>
         Defines the metadata for a column of a result set
returned by a stored procedure.The column name, type, and nullability
must match the values returned by the JDBC result set metadata at run-time.
         A column is considered to be nullable unless it is
explicitly defined to not accept nulls. If the "nullable"
attribute is absent then the column is considered to not be nullable.
         The element name associated with the column
is defined by the value of the "as" attribute if present,
or the column name otherwise.
         The element may contain an XML document,
in which case it must have an "element" attribute that
defines the XML Schema name of its root element.
       </documentation>
    </annotation>
         <complexType>
      <attribute name="name" type="string" use="required"/>
      <attribute name="type" type="dadx:columnType" use="required"/>
      <attribute name="nullable" type="boolean"/>
      <attribute name="as" type="string"/>
      <attribute name="element" type="QName"/>
    </complexType>
     </element>
```

Figure 33. DADX schema (Part 4 of 10)

```
<simpleType name="columnType">
    <restriction base="string">
      <enumeration value="BIGINT" />
      <enumeration value="CHAR" />
      <enumeration value="CLOB" />
      <enumeration value="DATE" />
      <enumeration value="DECIMAL" />
      <enumeration value="DOUBLE" />
      <enumeration value="FLOAT" />
      <enumeration value="INTEGER" />
      <enumeration value="NUMERIC" />
      <enumeration value="REAL" />
      <enumeration value="SMALLINT" />
      <enumeration value="TIME" />
      <enumeration value="TIMESTAMP" />
      <enumeration value="TINYINT" />
      <enumeration value="VARCHAR" />
    </restriction>
 </simpleType>
 <element name="operation">
    <annotation>
      <documentation>
        Defines an operation of the Web Service.
        Each operation has a unique name and is optionally described
        by WSDL documentation.
        An operation is defined using one of the supported operators.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref="wsdl:documentation" minOccurs="0" />
        <choice>
          <element ref="dadx:retrieveXML" />
          <element ref="dadx:storeXML" />
          <element ref="dadx:query" />
          <element ref="dadx:update" />
          <element ref="dadx:call" />
        </choice>
      </sequence>
      <attribute name="name" type="NCName" use="required" />
    </complexType>
 </element>
```

Figure 33. DADX schema (Part 5 of 10)

```
<element name="retrieveXML">
   <annotation>
     <documentation>
       Retrieves a set of XML documents by composing them from relational data.
       This operator requires the DB2 XML Extender.
       The mapping from relational data to XML is defined by a
       Document Access Definition (DAD) which can be specified
       by referring to either a resource file or the name of an XML Collection
       that has been previously enabled in the database.
       The DAD must define an XML Collection and can use either SQL
       or RDB mapping. The DAD behavior may be modified by an override.
       If no override is desired, the no_override element must be used.
       Otherwise, the SQL_override element must be used for SQL mapping and the
       XML_override element must be used for RDB mapping. In either case, the
       override string may contain input parameters using the host variable syntax.
       The name and type of all parameters must be defined in a list of
       parameter elements that are uniquely named within this operation.
       </documentation>
   </annotation>
   <complexType>
     <sequence>
       <choice>
         <element ref="dadx:DAD_ref"/>
         <element ref="dadx:collection_name"/>
       </choice>
       <choice>
         <element name="no_override">
           <complexType/>
         </element>
         <element name="SQL_override" type="string"/>
         <element name="XML_override" type="string"/>
       </choice>
       <element ref="dadx:parameter" minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
   </complexType>
   <unique name="retrieveXmlParameterNames">
     <selector xpath="dadx:parameter"/>
     <field xpath="@name"/>
   </unique>
</element>
```

Figure 33. DADX schema (Part 6 of 10)

```
<element name="storeXML">
   <annotation>
     <documentation>
       Stores an XML document by decomposing it into relational data.
       This operator requires the DB2 XML Extender.
       The mapping from relational data to XML is defined by a
       Document Access Definition (DAD) which can be specified
       by referring to either a resource file or the name of an XML Collection
       that has been previously enabled in the database.
       The DAD must define an XML Collection and must use RDB mapping.
       </documentation>
   </annotation>
   <complexType>
      <choice>
       <element ref="dadx:DAD_ref"/>
       <element ref="dadx:collection_name"/>
     </choice>
    </complexType>
</element>
<element name="query">
   <annotation>
     <documentation>
       Retrieves a set of relational data using an SQL SELECT statement.
       The result set must consist of uniquely named columns.
       If any result set column contains XML documents, the XML document
type must be defined using an XML_result element.
       The statement may contain input parameters using the host variable syntax.
       The input parameters must be defined by a list of parameter
elements that are uniquely named within this operation.
     </documentation>
   </annotation>
   <complexType>
     <sequence>
       <element name="SQL_query" type="string"/>
       <element ref="dadx:XML_result" minOccurs="0" maxOccurs="unbounded"/>
       <element ref="dadx:parameter" minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
   </complexType>
   <unique name="XML_resultNames">
     <selector xpath="dadx:XML_result"/>
     <field xpath="@name"/>
   </unique>
   <unique name="queryParameterNames">
     <selector xpath="dadx:parameter"/>
     <field xpath="@name"/>
   </unique>
</element>
```

Figure 33. DADX schema (Part 7 of 10)

```
<element name="update">
    <annotation>
      <documentation>
        Updates a relational table using an SQL INSERT,UPDATE,or DELETE
 statement and reports the number of rows affected.
        The statement may contain input parameters using the host variable syntax.
        The input parameters must be defined by a list of parameter
elements that are uniquely named within this operation.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element name="SQL_update" type="string"/>
        <element ref="dadx:parameter" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <unique name="updateParameterNames">
      <selector xpath="dadx:parameter"/>
      <field xpath="@name"/>
    </unique>
 </element>
```

Figure 33. DADX schema (Part 8 of 10)

```
<element name="call">
    <annotation>
      <documentation>
        Calls a stored procedure.The call statement contains
in, out, and in/out parameters using host variable syntax.
        The parameters are defined by a list of parameter elements
that are uniquely named within the operation.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element name="SQL_call" type="string"/>
        <element ref="dadx:parameter" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="dadx:result_set" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <unique name="callParameterNames">
      <selector xpath="dadx:parameter"/>
      <field xpath="@name"/>
    </unique>
    <unique name="callResultSetNames">
      <selector xpath="dadx:result_set"/>
      <field xpath="@name"/>
    </unique>
  </element>
<element name="result_set">
      <annotation>
      <documentation>
        Defines a result set. The name defines the element name of
the result set and becomes part of the output message. The metatdata name
 refers to a result set metadata element defined in the same document.
      </documentation>
    </annotation>
        <complexType>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="metadata" type="NCName" use="required"/>
    </complexType>
    </element>
<element name="DAD_ref" type="string"/>
<element name="collection_name" type="string"/>
<element name="parameter">
    <annotation>
      <documentation>
        Defines a named parameter. A parameter must have it contents defined either
 by an XML Schema element or type, but not both.The parameter kind in one of in,
        out, or in/out, with in being the default.
      </documentation>
    </annotation>
```

Figure 33. DADX schema (Part 9 of 10)

```
<complexType>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="element" type="QName"/>
      <attribute name="type" type="QName"/>
      <attribute name="kind" type="dadx:parameterKindType" default="in"/>
    </complexType>
 </element>
 <simpleType name="parameterKindType">
    <restriction base="string">
      <enumeration value="in"/>
      <enumeration value="out"/>
      <enumeration value="in/out"/>
    </restriction>
 </simpleType>
 <element name="XML_result">
    <annotation>
      <documentation>
        Defines a named column that contains XML documents. The document type
        must be defined by the XML Schema element of its root.
      </documentation>
    </annotation>
    <complexType>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="element" type="QName" use="required"/>
    </complexType>
 </element>
</schema>
```

Figure 33. DADX schema (Part 10 of 10)

Appendix C. Sample files

The following sections provide the longer sample files used throughout this document. In the example, the database is called SALES_DB, which is the sample database used in the documentation and samples shipped with DB2 XML Extender Version 7.2 FixPak 7. The SALES_DB database stores information about part orders.

Suppose that you must provide a Web Service that retrieves orders based on the following conditions:

- find all the orders
- find all the orders for parts of a specified color
- find all the orders whose price is greater than or equal to a minimum price

You create a DADX file named `PartOrders.dadx` that contains the following operations:

- findAll
- findByColor
- findByMinPrice

You create a Web Service by deploying the `PartOrders.dadx` file to the `services` Web application which has been configured with the dxx_sales_db instance of WORF. The deployment location of this file is `WEB-INF/classes/groups/dxx_sales_db/PartOrders.dadx`.

The Web Service supports access by the following protocols:

- HTTP GET
- HTTP POST
- HTTP SOAP

HTTP GET and POST are useful for simple access from Web browsers. In this case, the request uses the content type of `application/x-www-form-urlencoded`.

For example, suppose the Web services are deployed on the host www.mycompany.com. The following URLs would invoke the Web services using HTTP GET:

- `http://www.mycompany.com /services/sales/PartOrders.dadx /findAll`
- `http://www.mycompany.com /services/sales/PartOrders.dadx /findByColor?color=red`
- `http://www.mycompany.com /services/sales/PartOrders.dadx /findByMinPrice?minprice=20000`

This syntax encodes the method in the URL as the extra path information and the parameters as the query string. The responses to these requests have a content type of `text/xml`. For HTTP POST the query string is sent in the body of the request instead of the URL, but its content type is still application/x-www-form-urlencoded. Here is an example of an HTTP POST request when captured with a TCP trace utility. The example shows both the HTTP header and body:

```
POST /services/sales/PartOrders.dadx/findByColor HTTP/1.1
User-Agent: Java1.3.0
Host: localhost:9081
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-length: 12
color=red+++
```

A Web Service defined by a DADX file is self-describing. It dynamically generates a documentation and test page, WSDL documents, and XML Schema. The following HTTP GET URL requests the documentation and test page:

```
http://www.mycompany.com/services
/sales/PartOrders.dadx/TEST
```

The following HTTP GET URL requests the WSDL description of the service:

```
http://www.mycompany.com/services
/sales/PartOrders.dadx/WSDL
```

For HTTP SOAP, the services are invoked by sending SOAP envelopes using POST to the URL:

```
http://www.mycompany.com/services
/sales/PartOrders.dadx/SOAP
```

But with a request content type of `text/xml` instead of `application/x-www-form-urlencoded`. Here is an example of a SOAP request when traced with a TCP monitor like the one built into WebSphere Studio, or the one that is part of Apache SOAP. This example includes the HTTP header information and the HTTP body:

```
POST /services/sales/PartOrders.dadx/SOAP HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 547
SOAPAction: "http://tempuri.org/sales/PartOrders.dadx"

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:findByColor xmlns:ns1="http://tempuri.org/sales/PartOrders.dadx" SOAP-
ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
<color xsi:type="xsd:string" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">red </color>
</ns1:findByColor>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

.

PartOrder DADX file

`PartOrders.dadx` implements all three of its operations using the <retrieveXML> operator which uses the XML collection access method. In general, each operation can use a different operator and access method.

```
<?xml version="1.0"?>
  <DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                        xmlns="http://www.w3.org/1999/xhtml">
      Provides queries for part order information at myco.com.
      See <a href="../documentation/PartOrders.html" target="_top">
       PartOrders.html</a> for more information.
    </wsdl:documentation>

    <operation name="findAll">
      <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
        Returns all the orders with their complete details.
      </wsdl:documentation>
      <retrieveXML>
          <DAD_ref>getstart_xcollection.dad</DAD_ref>
          <SQL_override>
          select o.order_key, customer_name, customer_email,
            p.part_key, color, quantity, price, tax, ship_id, date, mode
          from order_tab o, part_tab p,
            table(select substr(char(timestamp(generate_unique())),16)
              as ship_id, date, mode, part_key from ship_tab) s
          where p.order_key = o.order_key and s.part_key = p.part_key
          order by order_key, part_key, ship_id
          </SQL_override>
      </retrieveXML>
    </operation>
```

Figure 34. The PortOrder.DADX file (Part 1 of 3)

```
<operation name="findByColor">
      <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Returns all
      the orders that include one or more parts that have the specified
      color, and only shows the details for those parts.</wsdl:documentation>
      <retrieveXML>
        <DAD_ref>getstart_xcollection.dad</DAD_ref>
        <SQL_override>
         select o.order_key, customer_name, customer_email,
           p.part_key, color, quantity, price, tax, ship_id, date, mode
         from order_tab o, part_tab p,
           table(select substr(char(timestamp(generate_unique())),16)
             as ship_id, date, mode, part_key from ship_tab) s
         where p.order_key = o.order_key and s.part_key = p.part_key
           and color = :color
         order by order_key, part_key, ship_id
        </SQL_override>
        <parameter name="color" type="xsd:string"/>
      </retrieveXML>
    </operation>
```

Figure 34. The PortOrder.DADX file (Part 2 of 3)

```
<operation name="findByMinPrice">
     <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Returns all
 the orders that include one or more parts that have a price greater than
 or equal to the specified minimum price, and only shows the details for
 those parts.</wsdl:documentation
     <retrieveXML>
       <DAD_ref>getstart_xcollection.dad</DAD_ref>
       <SQL_override>
        select o.order_key, customer_name, customer_email,
          p.part_key, color, quantity, price, tax, ship_id, date, mode
        from order_tab o, part_tab p,
          table(select substr(char(timestamp(generate_unique())),16)
            as ship_id, date, mode, part_key from ship_tab) s
        where p.order_key = o.order_key and s.part_key = p.part_key
          and p.price >= :minprice
        order by order_key, part_key, ship_id
       </SQL_override>
       <parameter name="minprice" type="xsd:decimal"/>
     </retrieveXML>
   </operation>

 </DADX>
```

Figure 34. The PortOrder.DADX file (Part 3 of 3)

getstart_xcollection.dad file

The DADX file references the following XML Extender DAD file. This file specifies the structure of XML documents that are to be created using the <retrieveXML> operation.

```
<?xml version="1.0"?>
  <!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
  <DAD>
  <validation>NO</validation>
  <Xcollection>
  <SQL_stmt>select o.order_key, customer_name, customer_email, p.part_key,
   color, quantity, price, tax, ship_id, date, mode
   from order_tab o, part_tab p,
   table(select substr(char(timestamp(generate_unique())),16) as ship_id, date,
   mode, part_key from ship_tab) s
     where o.order_key = 1 and p.price > 20000 and
           p.order_key = o.order_key and
           s.part_key = p.part_key
     ORDER BY order_key, part_key, ship_id</SQL_stmt>
```

Figure 35. The getstart_xcollection.dad file (Part 1 of 3)

```
<prolog>?xml version="1.0"?</prolog>
  <doctype>!DOCTYPE Order SYSTEM "c:\dxx\samples\dtd\getstart.dtd"</doctype>
  <root_node>
  <element_node name="Order">
    <attribute_node name="key">
      <column name="order_key"/>
    </attribute_node>
    <element_node name="Customer">
      <element_node name="Name">
        <text_node><column name="customer_name"/></text_node>
      </element_node>
      <element_node name="Email">
        <text_node><column name="customer_email"/></text_node>
      </element_node>
    </element_node>
```

Figure 35. The getstart_xcollection.dad file (Part 2 of 3)

```
 <element_node name="Part">
      <attribute_node name="color">
        <column name="color"/>
      </attribute_node>
      <element_node name="key">
        <text_node><column name="part_key"/></text_node>
      </element_node>
      <element_node name="Quantity">
        <text_node><column name="quantity"/></text_node>
      </element_node>
      <element_node name="ExtendedPrice">
        <text_node><column name="price"/></text_node>
      </element_node>
      <element_node name="Tax">
        <text_node><column name="tax"/></text_node>
      </element_node>
      <element_node name="Shipment" multi_occurrence="YES">
        <element_node name="ShipDate">
          <text_node><column name="date"/></text_node>
        </element_node>
        <element_node name="ShipMode">
          <text_node><column name="mode"/></text_node>
        </element_node>
      </element_node>
    </element_node>
  </element_node>
  </root_node>
  </Xcollection>
  </DAD>
```

Figure 35. The getstart_xcollection.dad file (Part 3 of 3)

POIAPartOrders.dadx file

The `POIAPartOrders.dadx` file shows how an individual application implements a standard Web service.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns="http://www.w3.org/1999/xhtml">
    Provides queries for part order information at myco.com.
    This Web Service is compliant with the Part Ordering Industry
    Association standard.
    See <a href="../documentation/PoiaPartOrders.html"
        target="_top">PoiaPartOrders.html</a> for more information.
  </wsdl:documentation>
```

Figure 36. The PoiaPartOrders.dadx (Part 1 of 4)

```
<implements namespace="http://www.poia.org/PartOrders.wsdl"
    location="http://www.poia.org/PartOrders.wsdl"/>

  <operation name="findAll">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Returns
        all the orders with their complete details.</wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad</DAD_ref>
      <SQL_override>
      select o.order_key, customer_name, customer_email,
        p.part_key, color, quantity, price, tax, ship_id, date, mode
      from order_tab o, part_tab p,
        table(select substr(char(timestamp(generate_unique())),16)
          as ship_id, date, mode, part_key from ship_tab) s
      where p.order_key = o.order_key and s.part_key = p.part_key
      order by order_key, part_key, ship_id
      </SQL_override>
    </retrieveXML>
  </operation>
```

Figure 36. The PoiaPartOrders.dadx (Part 2 of 4)

```
<operation name="findByColor">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders that include one or more parts that
      have the specified color, and only shows the details for
      those parts.
    </wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad</DAD_ref>
      <SQL_override>
       select o.order_key, customer_name, customer_email,
         p.part_key, color, quantity, price, tax, ship_id, date, mode
       from order_tab o, part_tab p,
         table(select substr(char(timestamp(generate_unique())),16)
           as ship_id, date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
         and color = :color
       order by order_key, part_key, ship_id
      </SQL_override>
      <parameter name="color" type="xsd:string"/>
    </retrieveXML>
  </operation>
```

Figure 36. The PoiaPartOrders.dadx (Part 3 of 4)

```
<operation name="findByMinPrice">
    <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders that include one or more parts that
      have a price greater than or equal to the specified minimum
      price, and only shows the details for those parts.
    </wsdl:documentation>
    <retrieveXML>
      <DAD_ref>getstart_xcollection.dad</DAD_ref>
      <SQL_override>
       select o.order_key, customer_name, customer_email,
         p.part_key, color, quantity, price, tax, ship_id, date, mode
       from order_tab o, part_tab p,
         table(select substr(char(timestamp(generate_unique())),16) x
           as ship_id, date, mode, part_key from ship_tab) s
       where p.order_key = o.order_key and s.part_key = p.part_key
         and p.price >= :minprice
       order by order_key, part_key, ship_id
      </SQL_override>
      <parameter name="minprice" type="xsd:decimal"/>
    </retrieveXML>
  </operation>

</DADX>
```

Figure 36. The PoiaPartOrders.dadx (Part 4 of 4)

WSDL files

The WSDL files generated from the Getting Started examples.

The deployable part:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
 targetNamespace=
     "http://localhost:8080/services/sales/PartOrders.dadx/WSDLservice"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:binding=
       "http://localhost:8080/services/sales/PartOrders.dadx/WSDLbinding"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://localhost:8080/services/sales/PartOrders.dadx/WSDLservice">
 <import
 location="http://localhost:8080/services/sales/PartOrders.dadx/WSDLbinding"
 namespace="http://localhost:8080/services/sales/PartOrders.dadx/WSDLbinding"/>
 <service name="theService">
 <port binding="binding:theSoapBinding" name="theSoapPort">
   <soap:address
     location="http://localhost:8080/services/sales/PartOrders.dadx/SOAP"/>
 </port>
 <port binding="binding:theGetBinding" name="theGetPort">
   <http:address location=
       "http://localhost:8080/services/sales/PartOrders.dadx/"/>
 </port>
 <port binding="binding:thePostBinding" name="thePostPort">
   <http:address location=
       "http://localhost:8080/services/sales/PartOrders.dadx/"/>
 </port>
 </service>
</definitions>
```

Figure 37. PartOrders-service.wsdl: deployment WSDL part

The reusable part:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
 targetNamespace=
       "http://localhost:8080/services/sales/PartOrders.dadx/WSDLbinding"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://localhost:8080/services/sales/PartOrders.dadx/WSDLbinding"
 xmlns:xsd-1999="http://www.w3.org/1999/XMLSchema"
 xmlns:xsd1="http://localhost:8080/services/sales/PartOrders.dadx/XSD">
 <wsdl:documentation
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    Provides queries for part order information at myco.com.
    See
    <a href="./documentation/PartOrders.html" target="_top">PartOrders.html
    </a> for more information.

 </wsdl:documentation>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 1 of 7)

```
<types>
   <schema
    targetNamespace=
         "http://localhost:8080/services/sales/PartOrders.dadx/XSD"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:imp1="http://schemas.ibm.com/db2/dxx/samples/dtd/getstart.dtd"
    xmlns:tns="http://localhost:8080/services/sales/PartOrders.dadx/XSD">
    <import
     nameSpace="http://schemas.ibm.com/db2/dxx/samples/dtd/getstart.dtd"
     schemaLocation=
      "http://localhost:8080/services/sales/dxx/samples/dtd/getstart.dtd/XSD" />
    <element name="findAllResult">
      <complexType>
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" ref="imp1:Order" />
        </sequence>
      </complexType>
    </element>
    <element name="findByColorResult">
      <complexType>
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" ref="imp1:Order" />
        </sequence>
      </complexType>
    </element>
    <element name="findByMinPriceResult">
      <complexType>
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" ref="imp1:Order" />
        </sequence>
      </complexType>
    </element>
   </schema>
 </types>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 2 of 7)

```
<message name="findAllInput" />
 <message name="findAllOutput">
    <part element="xsd1:findAllResult" name="return" />
 </message>
 <message name="findByColorInput">
    <part name="color" type="xsd-1999:string" />
 </message>
 <message name="findByColorOutput">
    <part element="xsd1:findByColorResult" name="return" />
 </message>
 <message name="findByMinPriceInput">
    <part name="minprice" type="xsd-1999:decimal" />
 </message>
 <message name="findByMinPriceOutput">
    <part element="xsd1:findByMinPriceResult" name="return" />
 </message>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 3 of 7)

```
<portType name="thePortType">
    <operation name="findAll">
      <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
      Returns all the orders with their complete details.

      </wsdl:documentation>
      <input message="tns:findAllInput"/>
      <output message="tns:findAllOutput"/>
    </operation>
    <operation name="findByColor">
      <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
       Returns all the orders that include one or more parts that have the
       specified color, and only shows the details for those parts.
      </wsdl:documentation>
      <input message="tns:findByColorInput"/>
      <output message="tns:findByColorOutput"/>
    </operation>
    <operation name="findByMinPrice">
      <wsdl:documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
       Returns all the orders that include one or more parts that have a
       price greater than or equal to the specified minimum price, and only
       shows the details for those parts.
      </wsdl:documentation>
      <input message="tns:findByMinPriceInput"/>
      <output message="tns:findByMinPriceOutput"/>
    </operation>
 </portType>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 4 of 7)

```
<binding name="theSoapBinding" type="tns:thePortType">
    <soap:binding style="rpc" transport=
         "http://schemas.xmlsoap.org/soap/http">
      <operation name="findAll">
        <soap:operation soapAction="urn:/sales/PartOrders.dadx" />
        <input>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </input>
        <output>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </output>
      </operation>
      <operation name="findByColor">
        <soap:operation soapAction="urn:/sales/PartOrders.dadx" />
        <input>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </input>
        <output>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </output>
      </operation>
      <operation name="findByMinPrice">
        <soap:operation soapAction="urn:/sales/PartOrders.dadx" />
        <input>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </input>
        <output>
          <soap:body namespace="urn:/sales/PartOrders.dadx" use="literal" />
        </output>
      </operation>
    </soap:binding>
 </binding>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 5 of 7)

```
<binding name="theGetBinding" type="tns:thePortType">
    <http:binding verb="GET" />
    <operation name="findAll">
      <http:operation location="findAll" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml />
      </output>
    </operation>
    <operation name="findByColor">
      <http:operation location="findByColor" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml />
      </output>
    </operation>
<operation name="findByMinPrice">
      <http:operation location="findByMinPrice" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml />
      </output>
    </operation>
 </binding>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 6 of 7)

```
<binding name="thePostBinding" type="tns:thePortType">
    <http:binding verb="POST"/>
    <operation name="findAll">
      <http:operation location="findAll"/>
      <input>
        <mime:content type="application/x-www-form-urlencoded"/>
      </input>
      <output>
        <mime:mimeXml/>
      </output>
    </operation>
    <operation name="findByColor">
      <http:operation location="findByColor"/>
      <input>
        <mime:content type="application/x-www-form-urlencoded"/>
      </input>
      <output>
        <mime:mimeXml/>
      </output>
    </operation>
    <operation name="findByMinPrice">
      <http:operation location="findByMinPrice"/>
      <input>
        <mime:content type="application/x-www-form-urlencoded"/>
      </input>
      <output>
        <mime:mimeXml/>
      </output>
    </operation>
 </binding>
</definitions>
```

Figure 38. PartOrders-binding.wsdl: reusable WSDL part (Part 7 of 7)

Appendix D. Encoding algorithm

This is an algorithm that encodes and decodes the password within the group.properties file.

1. Convert the clear text information into a sequence of data bytes using UTF-8 character encoding. Let L be the length of the data byte sequence.
2. Convert the data bytes into a further sequence of data bytes, data8, that is 8 times longer. Byte k of data8 is computed as follows. Let k = j * L + i where 0 <= i < L and 0 <= j < 8. First mask bit j of data byte i. Second, exclusive or this with k. This step distributes the bits of each data byte throughout the length of the data8 sequence.
3. Apply the standard base64 encoding algorithm to data8. This step renders the bytes as printable characters and also increases the length by a factor of 4/3.
4. Prefix the encoded string with ″encoded:″ to denote that it has been encoded.

Appendix E. Command reference

This section represents the commands you can use to do specific functions within WORF.

Encoder

Encodes or decodes a password in the group.properties file. For more information on the encoding algorithm, see Appendix D, “Encoding algorithm” on page 87.

- Example of encoding (assumes that worf.jar is listed in the CLASSPATH): `java com.ibm.etools.webservice.rt.util.Encoder -in group.properties -out group.properties`
- Example of decoding (assumes that worf.jar is listed in the CLASSPATH): `java com.ibm.etools.webservice.rt.util.Encoder -action decode -in group.properties -out group.properties`

Dadx2Dd

Generates a deployment descriptor from a DADX file. For information on the parameters to use with this command, see step 2 on page 47.

- Example: `java com.ibm.etools.webservice.rt.dadx.Dadx2Dd`

Check_install

Validates a DADX file. For information on the parameters to use with this command, see “Parameters” on page 56.

- Example: `java com.ibm.etools.webservice.util.Check_install [-srv] [-schdir pathToSchemasDir] [-sch schemaLocations] [-out outputFile] fileToCheck`

dadchecker

Validates a DAD file. For more information on the parameters to use with this command, see http://www.ibm.com/software/data/db2/extenders/xmlext/download/beta/dadcheck_rn.html

- Example: `java dadchecker.Check_dad_xml [-dad | -xml] [-all] [-dup dupName] [-enc encoding][-dtd dtdPath] [-xstruct xmlDocument] [-out outputFile] fileToCheck`

Appendix F. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

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

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

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

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

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

Bibliography

- *DB2 Universal Database SQL Reference, Volume 1*, SC09-4844
- *DB2 Universal Database SQL Reference, Volume 2*, SC09-4845
- *DB2 Universal Database WebSphere Performance Tuning Guide*, SG24-6417-00
- *DB2 XML Extender: Administration and Programming*, SC27-1234-00
- *DB2 XML Extender: Administration and Programming, Version 7.2 Release Notes*
- *Apache Jakarta Tomcat Version 4 documentation, available at http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html*
- *Using WSDL in a UDDI Registry 1.07, available at http://www.uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf*
- *Web Services Description Language (WSDL) 1,1, available at http://www.w3.org/TR/2001/NOTE-wsdl-20010315*
- *Web Sphere Application Server documentation available at http://www-3.ibm.com/software/webservers/appserv/library.html*

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

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

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Printed in U.S.A.