IBM DB2 Information Integrator

Application Developer’s Guide

Version 8.2

SC18-7359-01

IBM DB2 Information Integrator

Application Developer’s Guide

Version 8.2

SC18-7359-01

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 279.

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

You can order IBM publications online or through your local IBM representative:

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

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

© Copyright International Business Machines Corporation 2003, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2003, 2004

Preface

This book shows you why IBM® DB2 Information Integrator is the solution to help you integrate data through unified views and data placement. It also shows you how to use information to your best advantage.

Who should read this guide?

Data administrators, information analysts, system integrators, Web integrators, data librarians, data architects, and application developers can use the IBM DB2 Information Integrator solutions to create a strategic and open information integration platform.

Terminology

IBM DB2 Information Integrator uses standard terminology for database, connectivity, Structured Query Language (SQL), and local area network (LAN) concepts. All the DB2 Information Integrator concepts that are used in this book are defined in the glossary. Unless otherwise specified, assume the following meanings:

Data A raw fact. It can be structured, unstructured, or semi-structured. Data is usually organized for analysis. Data also helps you to make decisions.

Information
Data in a usable form, usually processed or interpreted in some way.

DB2 Information Integrator offerings

IBM DB2 Information Integrator is available in several offerings. Read the license agreement carefully for the terms and conditions of use for the edition that you install. For information about the installation and configuration of Information Integrator, see *DB2 Information Integrator Installation Guide*. For more information about the evolution of information integration, see the information integration support site http://www.ibm.com/software/data/integration/db2ii/support.html.

© Copyright IBM Corp. 2003, 2004

Chapter 1. Introduction to information integration development

This section describes the concepts and procedures for developing applications that integrate information in your enterprise.

Overview of information integration solutions

Businesses face many information integration challenges. The rapidly changing economic climate is driving the need for improved access to information, flexible analytical capabilities, and formal information inventories.

Introduction to information integration

IBM® has delivered world-class data management technology for over thirty years. IBM continues to enhance its enterprise offerings by developing information integration for small, medium, and large businesses. Information integration builds on the solid foundation of existing data management solutions. Information integration provides an end-to-end solution for transparently managing both the volume and diversity of data that is in the marketplace today.

The cost of doing business involves the need to integrate diverse and unconnected infrastructures. Businesses need the following goals:

- To integrate seamlessly with new businesses and link business applications with legacy systems
- To control the accelerating costs of managing disparate systems and integrating across heterogeneous pockets of automation
- To mitigate the shortages of people and skills while quickly reaching new markets

The need for a solution to efficiently access and manage information crosses product and industry boundaries.

Related concepts:

- "DB2 Information Integrator—the solution to integration" on page 5
- "What is information integration?" on page 1
- "Why is information integration important to your enterprise?" on page 2

What is information integration?

Information integration is a collection of technologies that combines database management systems, Web services, replication, federated systems, and warehousing functions into a common platform. It also includes a variety of programming interfaces and data models. Using the information integration technology, you can access diverse types of data (structured, unstructured, and semi-structured). You can transform that data into a format that provides easy access to information across the enterprise.

Information integration enables the integration of data and content sources with the following functions:

© Copyright IBM Corp. 2003, 2004

- Provides real-time read and write access
- Transforms data for business analysis and data interchange
- Manages data placement for performance, currency, and availability

Related concepts:
- "DB2 Information Integrator—the solution to integration" on page 5

Why is information integration important to your enterprise?

| The IBM® information integration strategy:
- | Provides users with the ability to manipulate legacy data.
- | Provides users with the ability to take advantage of familiar software to use | known assets and resources.
- | Provides users with the ability to acquire and easily maintain new data.
- | Provides users with the ability to use existing data management tools to access | data wherever it is located.

IBM has identified five types of integration that are based on an open services infrastructure. You can use these types of integration together or separately to solve business issues. The five types of integration that are listed here represent the various integration challenges that face businesses today. Information integration is at the core of these integration types.

User interaction
A user can work with a single, tailored user interface, which is available through virtually any device, with full transactional support. The user interaction results are integrated into multiple business systems.

Process integration
A business can change how it operates through modeling, automation, and monitoring of processes across people and heterogeneous systems, both inside and outside of the enterprise.

Application connectivity
Applications can connect to one another so that they share and use information for better use at the enterprise level.

Build to integrate
Users can build and deploy integration-ready applications by using Web services and existing assets. You can integrate new solutions with existing business assets.

Information integration
Diverse forms of business information can be integrated across the enterprise. Integration enables coherent search, access, replication, transformation, and analysis over a unified view of information assets to meet business needs.

IBM DB2® Information Integrator is a technology that provides a comprehensive solution to address customer requirements for information integration. The DB2 Information Integrator functions include information access, information integration, and information analysis.

Most modern integration needs are a mix of both application and information integration. Solution providers have multiple ways to integrate, share, and distribute information. It is more important to match the benefits of the different approaches to the different business integration requirements than to clearly define the boundary of the different types of integration.

Related concepts:
- "What is information integration?" on page 1
- "What problems does DB2 Information Integrator solve?" on page 3

Why information integration makes application development easier

The enterprise systems that exist today include customers, suppliers, partners, and electronic marketplaces. These systems interact with databases, application servers, content management systems, data warehouses, workflow systems, search engines, message queues, Web crawlers, mining and analysis packages, and other enterprise applications. The enterprise systems require a variety of programming interfaces, such as open database connectivity, Java database connectivity, Web services, Java objects, and Java 2 Platform, Enterprise Edition. Enterprise systems need to work with a variety of data models and languages, such as Structured Query Language, Extensible Markup Language, Web Services Description Language, and Simple Object Access Protocol.

Information integration is a technology approach that combines core elements from data management systems, data warehouses, Web services, and other enterprise applications into a common platform.

What problems does DB2 Information Integrator solve?

IBM® DB2® Information Integrator provides solutions to many of your enterprise problems. DB2 Information Integrator helps you to do the following things:
- Manage all forms of information. You can do this by using the storage, consolidation, archiving, transformation, loading, monitoring access, replication security, and the cleansing techniques that are provided with DB2 Information Integrator
- Unify your query models of local and federated data that are using XML, SQL, Web services, and messaging applications
- Analyze information efficiently using mining, categorization, summarization, and real-time decision making

The following table outlines some typical customer problems. In all of these examples, DB2 Information Integrator and the related technologies provide the software and framework for the solution.

Table 1. Typical customer problems

Typical customer problem	Solution	Results, or value added	Technical requirements
Enhance effectiveness of telephone sales.	Record the sales conversation and store in a text format. Combine data and text mining techniques over sales and transaction data and sales conversation data.	Improved sales offerings and strategies by examining patterns that would not be visible had the structured and unstructured data remained isolated.	Integration of structured data and unstructured data; mining techniques needed over the integrated data.

Table 1. Typical customer problems (continued)

Typical customer problem	**Solution**	**Results, or value added**	**Technical requirements**
Integrate disparate groups of information to improve business effectiveness. This might include home-grown applications, packaged applications, or applications acquired through mergers or acquisitions.	Use Extensible Markup Language (XML) to integrate and exchange information across the enterprise. Build enterprise models around XML to be used by newer applications.	Extract competitive value from existing information. Increase profitability through operational efficiencies. Shorten business cycle and decision making process.	Integration of structured and unstructured data including application data. Application and process integration (messaging, workflow). Mining of text data (categorization and search).
Improve profitability of current customer base.	Analyze customer accounts and behavior. Consolidate business data in a warehouse through replication. Use Intelligent Miner™ for Data to mine information and MQ messaging to flow information to business users.	Targeted product offerings can be based on customer profiles; higher profit per customer.	Replication, mining of relevant information, reliable distribution mechanism.
Scientific users require integrated view of chemical and biological information stored in distributed Oracle sources, and external nonrelational sources.	DiscoveryLink specialized data source support across multiple heterogeneous data sources. Scientific mining algorithms.	Real time access to information.	Complex query optimization; mining of relevant information.
Improve ordering process to accommodate varying customer demands. Integrate with existing operational systems.	Create new parts ordering architecture. Enter orders in database. Use replication and federated systems to transfer orders to older systems.	Improved order response time (reduced billing cycle, improved cash flow).	Replication, Federated systems, WebSphere®.
Provide improved customer service-integrate information across several divisions of a company to provide seamless customer relationship information.	Integrate client information across diverse business units to provide an integrated view of customers either as individuals or as a group.	Personalized recommendations to customers in response to customer electronic mail (e-mail). Provide loyalty point tracking across the enterprise.	Warehousing, federated systems, replication, WebSphere.

Related concepts:

- "The DB2 Universal Database family—a foundation for information integration" on page 13

- “DB2 Information Integrator—the solution to integration” on page 5
- “Introduction to information integration” on page 1
- “Planning for the information integration architecture” on page 5

The foundation of information integration

The foundation of information integration is the ability to model diverse data sources, such as relational, Extensible Markup Language (XML), or flat files. Using the technologies of information integration, you can get transformation capabilities over these modeled data sources. You can subsequently represent the merged view in ways that your business finds useful, such as with SQL statements embedded in applications, or Web services applications. A well-planned database architecture is useful and necessary in the information integration technologies for these reasons:

- Database management systems (DBMSs) have proven that they can manage the information explosion that has occurred in traditional business applications. Database administrators understand the problems of storage, retrieval, transformation, scalability, reliability, and availability.
- The database industry is learning to adapt to the diversity of data and access patterns in e-business applications. The applications and functions that are within the database industry use built-in object-relational support, and Extensible Markup Language (XML) capabilities. These functions support federated access to external data sources.
- The investment in database technology continues to grow and includes databases, supporting tools, application development environments, and people who are skilled in database techniques.

Related concepts:

- “The DB2 Universal Database family—a foundation for information integration” on page 13
- “DB2 Information Integrator—the solution to integration” on page 5
- “What problems does DB2 Information Integrator solve?” on page 3

DB2 Information Integrator—the solution to integration

The information integration solution for the enterprise is IBM® DB2® Information Integrator. DB2 Information Integrator is a framework for the complete integration of data and the processes that support the data. There is a variety of challenges that the technologies of DB2 Information Integrator can address. The best solution for your enterprise might be a combination of the individual Information Integrator technologies and companion products. Having a multiple set of solutions available to you makes it even more important to be able to move easily from one technology to another. DB2 Information Integrator delivers complete data integration with technologies that work together seamlessly.

Related concepts:

- “Introduction to information integration” on page 1
- “What problems does DB2 Information Integrator solve?” on page 3

Planning for the information integration architecture

In planning the architecture of your integration environment, you can use Web service tools, database management tools, and warehouse tools. You can use the language of integration, which is XML. You can use the language of database technologies, which is SQL.

Extensible Markup Language (XML) is a technology that you can use for Web applications, that packages data along with its metadata. XML is a key element in the strategy to integrate information. XML provides a common language that enables business-to-business communication. It provides an easy way to send semi-structured data across the Web so that nothing gets lost in translation. It masks the differences in endpoint infrastructures. XML renders information appropriately for a variety of devices, because it separates the document content from the document presentation. XML acts intelligently based on the content, or decision support, because it includes a description of the data and how it relates to other pieces of data. XML can help your company produce smarter search results because it provides a context for search arguments.

SQL is an ANSI (American National Standards Institute) standard for accessing relational database management systems. SQL provides access to relational, federated and nonrelational data. By using SQL-based queries in Web services, you can send SQL statements and stored procedure calls to DB2® Universal Database. Then, the Web services returns the results with some default tagging. SQL-based queries are useful on the Web when the returned data uses only a simple mapping of SQL data types, with column names as elements. Use DB2 XML Extender and SQL-based queries when you need user-defined mappings of SQL data to XML elements and attributes. If you are using DB2 XML Extender to store XML documents within a single column of a table, you can use SQL-based queries to retrieve those documents intact as a character large object (CLOB). Also, you can use DB2 XML Extender with SQL-based queries to invoke the user-defined functions that extract parts of the document.

You can use SQL-based queries to invoke DB2 UDB stored procedures. Stored procedures are natural for conversion to Web services since they are themselves an encapsulation of programming logic and database access. A Web service invocation of a stored procedure makes it possible to dynamically provide input parameters and to retrieve results.

The architecture of information integration includes three layers.

- The data layer.

  The data layer is the foundation of integration. This layer provides storage, retrieval, and transformation of data from base sources in different formats. This layer is based on a federated DBMS architecture. Data is stored as structured relational tables, semi-structured XML documents, or in unstructured formats. A hybrid XML and relational storage and retrieval infrastructure ensure high performance and data durability. This data layer uses the federated database technology with a flexible wrapper architecture to integrate external data sources, which can be traditional data servers, enterprise applications, or workflows.
- The services layer.

  Information integration is about accessing the data and determining how to use the data. This layer provides an infrastructure to transparently embed data access services into enterprise applications and business processes. This includes query processing, text search and mining, transformation, and replication.
- The application layer.

  Information integration is about programming interfaces that can use the data. The application layer provides standards-based programming model and query language to the other layers. You can base the interfaces on Web services, or traditional application programming interfaces. The application layer enhances standard query languages.

Each layer is a separate entity, but must also depend on and work with the other layers to provide a more complete integration strategy. Figure 1 on page 7 shows the data layer, the services layer, and the application layer.

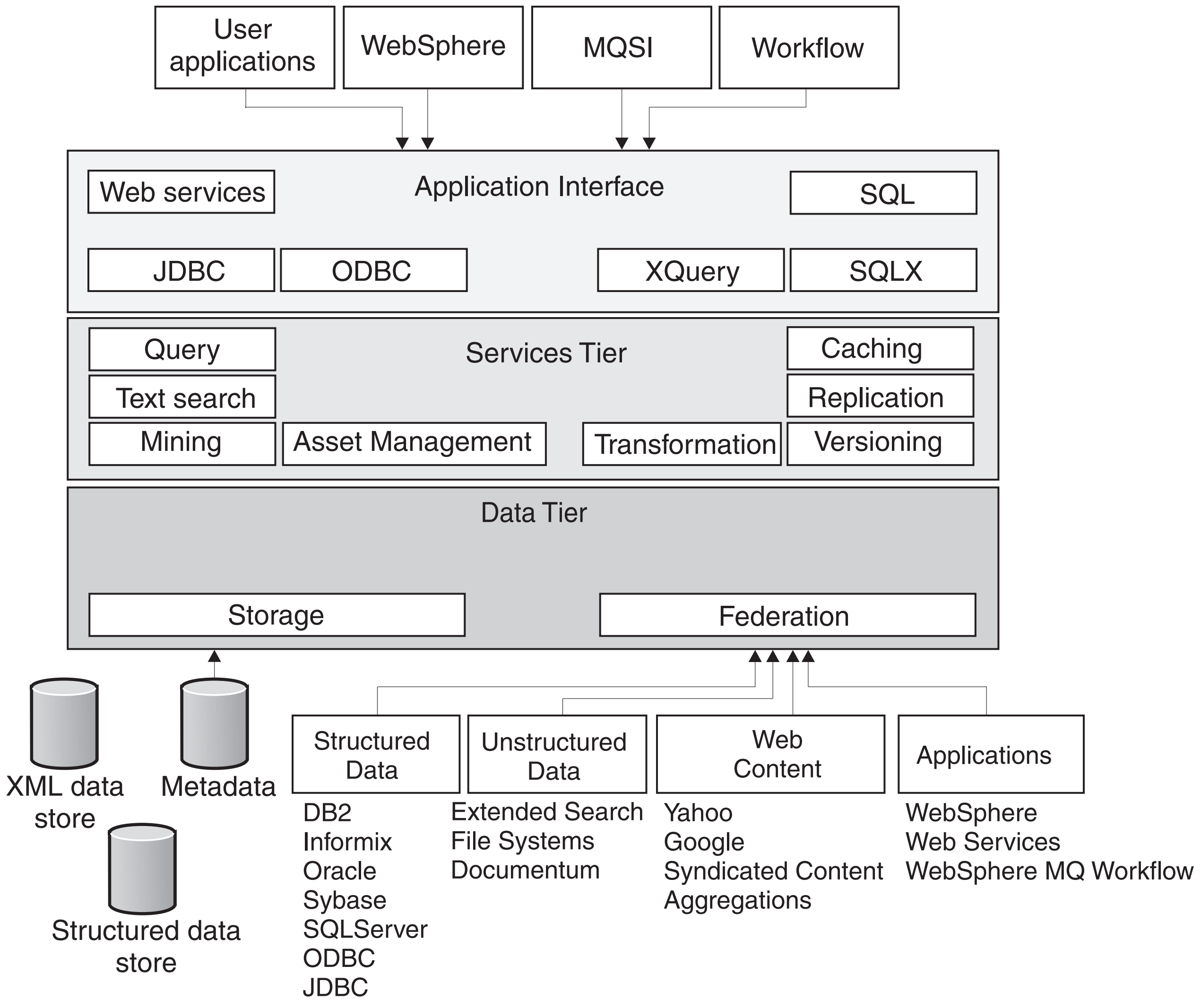


Figure 1. Integration layers

Related concepts:

- "DB2 Information Integrator—the solution to integration" on page 5
- "What problems does DB2 Information Integrator solve?" on page 3
- "Why is information integration important to your enterprise?" on page 2

DB2 Information Integrator—relational federated technologies

Federated database management systems offer help in accessing and manipulating disparate data. Federated systems provide a single-site image of distributed data that is stored or generated in a variety of formats. Federated systems offer a common interface for accessing this data. The federated database management system of IBM® DB2® Information Integrator features an extended catalog. This catalog is capable of maintaining statistics about remote data and using these statistics to globally optimize data access. By using the federated systems, you can gather statistics about remote data sources and use this information in the query optimizer technology that is provided by DB2 Universal Database™. This allows you to access data fast. Federated systems use the query rewrite facility of DB2 Universal Database to rewrite slower performing queries into their faster equivalent form.

In a federated database engine, you access data sources through software components that are called wrappers. Each wrapper contains information about the data source, such as the default mappings between the data source data types and the DB2 Universal Database data types. To implement a wrapper, the server uses routines that are stored in a library called a wrapper module. These routines allow the server to perform operations such as connecting to a data source and iteratively retrieving data from it. To use the wrappers to query, retrieve, and manipulate data, you must install the wrappers. Then, you must register each wrapper to add them to your federated system. For more information on the federated systems technology, including the definitions of the federated systems objects, see *Federated Systems Guide*.

DB2 Information Integrator federated technology provides a virtual database for multiple data sources. The data sources can do the following things:

- Run on different hardware and different operating system platforms
- Be provided by different vendors
- Use different application programming interfaces and different SQL dialects

The federated systems databases enables programmers to customize their database management system to access a data source of their choosing, whether relational or nonrelational.

Relational federated wrappers can query data in other relational database management systems (RDBMSs), such as the ones listed on page 8. Relational and nonrelational wrappers provide transparent access to these other database systems by mapping these sources to DB2 UDB. You can access these data sources with a single query, and make use of the performance techniques and query rewrite functionality that is provided by federated systems and DB2 UDB.

Among the relational wrappers that are offered by the federated relational technology are wrappers for the non-IBM relational databases and Informix® databases. You need the relational wrappers if you want to access data that is stored in some of the following data sources:

- Oracle
- Sybase
- Microsoft® SQL Server
- IBM DB2 family of products
- Teradata
- Open Database Connectivity (ODBC) sources

Related concepts:

- "Tuning query processing" in the *Federated Systems Guide*

Related tasks:

- "Global optimization" in the *Federated Systems Guide*

DB2 Information Integrator—nonrelational federated technologies

Nonrelational federated technologies consist of wrappers that provide access to nonrelational data. With these wrappers, you can access table-structured files, Excel files, Extensible Markup Language (XML) documents, BLAST search algorithms, Documentum data, Entrez sources, HMMER sources, BioRS, Extended Search sources, business applications and Web service providers that are described by Web service description language files. Through the extended search sources you can get access to a wide variety of unstructured data sources. Some of these sources include Domino™, Microsoft® Exchange, Microsoft Index Server, and Lightweight Directory Access Protocol (LDAP) directories, among others. For more information on federated systems see *Federated Systems Guide*. For more information on wrappers and how to create them, see *DB2 Information Integrator Wrapper Developer's Guide*.

The nonrelational set of wrappers contains several components that you can install:

Scientific data sources
These might include some unstructured data. Some of that data might contain genomic, proteomic, bioinformatic, and cheminformatic information that are developed for the life sciences industry. These wrappers enable a federated system to integrate genetic, chemical, biological, and other research data from distributed sources. For example, you can use one SQL statement to integrate protein sequence data from a database in Switzerland, chemical structure data from a database in Japan, and spectroscopic data that is stored in table-structured flat files on your local area network. The data appears as if it is in one virtual database.

Structured file data sources
These contain data that is stored in files with a defined, repeatable structure. For example, this might be an Excel spreadsheet or a flat file where each record contains the same number of fields, separated by a delimiter.

Application data sources
Application wrappers use an application to access the underlying data. The raw data can be in a number of standard and nonstandard formats.

Web services provider sources
The purpose of the Web services wrapper is to let DB2® UDB and SQL users access Web service providers described by Web services description language (WSDL) files. The Web service is invoked by exchanging SOAP messages between the Web services provider and the consumer. Web services wrappers use Web services just as the SOAP user-defined functions use Web services. You can issue SQL statements with federated nicknames and views that access Web services, or you can issue a set of user-defined functions to access Web services. Web services can be discovered by specifying the WSDL and then the necessary nicknames can be created based on information in the WSDL. You query the nicknames to access the Web services to modify or access information.

Examples of the data sources that need the nonrelational wrappers include some of the following data sources. This list is only an example list. For complete wrapper information, see the *DB2 Information Integrator Data Source Configuration Guide*.

- BLAST
- Excel
- Table-structured files

- Documentum
- HMMER
- Entrez
- Extended search
- BioRS
- Web services
- Business applications that are accessible through WebSphere® Business Integration, including SAP, PeopleSoft, and Siebel

Related concepts:

- "Data type mappings for nonrelational data sources" in the *Federated Systems Guide*
- "The Web service consumer functions" on page 145
- "The Web services wrapper and the Web services description language document" in the *IBM DB2 Information Integrator Data Source Configuration Guide*

WebSphere Portal examples and DB2 Information Integrator

You can use DB2® Information Integrator in a WebSphere® portal environment, along with WebSphere Application Server as the runtime environment to help achieve your customer goals without requiring significant investments in Web-enabled application development skills. DB2 Information Integrator works with WebSphere Portal and WebSphere Application Server to enable Web applications or components to access disparate data sources through a single API (SQL) so that the amount of complex application logic required to access and merge the disparate data is greatly reduced. You can also use DB2 Information Integrator to display full data content from the source, and with some configurations, write to a data source. DB2 optimization capabilities help ensure that data is retrieved efficiently from remote data sources.

One example of a business situation where DB2 Information Integrator can be used with a WebSphere portal to improve development efficiency involves the use of portals to deliver different information and services to employees in a single view. The fictitious insurance company, Cotton-wood Insurance Corporation, needed a way to deliver a variety of claim related information to its customer support representatives in a single view. All the data that customer support representatives needed to work with was spread across disparate relational and non relational data stores including:

- Customer account information in a DB2 UDB for z/OS™ database
- Adjuster appraisal forms and daily operational notes in a Lotus® Domino™ server
- Police reports stored as XML files
- New insurance claims written to WebSphere MQ queues to initiate new claims processing

The developers at Cotton-wood created a simple portal to aid service representatives in serving their insurance customers by providing all related claims information within a single web based view. Within the portal, several portlets are used to deliver information or services to the portal. DB2 Information Integrator helped the developers at Cotton-wood to access insurance company data with a unified view across all of the data sources. The Cotton-wood developers worked with standard WebSphere portlets and DB2 Information Integrator to minimize the code required to access the different data sources. Instead of writing code to work with each native API for the different data source formats, the developers used SQL to access DB2 Information Integrator to query the data, allowing the complex access and join logic to be handled by the DB2 Information Integrator.

One portlet the Cotton-wood developers created, provided customer service representatives with a single view of all open claims. This required the accessing and merging of customer account information stored in DB2 UDB, police reports stored as XML files, and adjuster appraisal forms stored in a Lotus Domino server. Without DB2 Information Integrator, the developers would have had to access each of the data sources and then write the portlet code required to merge all the data together and display it back to the user. Instead, the development team only needed to develop one portlet and were able to utilize a single SQL statement to run against the DB2 UDB, XML, and Lotus Domino data.

When the portlet runs, the SQL query travels from the portal application to the DB2 Information Integrator federated server. The federated server accesses all the necessary data, making disparate data sources appear to be a single resource. This eliminates the need for developers to write code to manage multiple connections, queries, and join logic that would otherwise be required. After collecting the data from the various sources, the federated server returns a single result set to the client, providing the customer service representative all the relevant claims information that is integrated from the DB2 UDB, XML, and Lotus Domino data sources.

This example shows how DB2 Information Integrator can significantly reduce the amount of customized code, skill requirements, and time needed to develop application logic necessary for accessing and merging data from disparate relational and nonrelational sources. By providing a single API for your development team to work with, and utilizing data merging and optimization features from DB2 UDB, DB2 Information Integrator allows developers to focus more on the usability of your application rather than how to access and merge the data together.

For additional information on WebSphere Portal and DB2 Information Integrator see Sample code for the WebSphere Portal and DB2 Information Integrator.

Related concepts:

- "Advantages of a federated system" on page 163
- "DB2 Information Integrator—nonrelational federated technologies" on page 9

Related tasks:

- "Deploying a federated application" on page 181

Introducing scenarios used throughout this guide

This section introduces the scenarios that are discussed throughout this guide. They are based on actual customer experiences, but represent a composite of several companies. The company names are fictitious.

The Cottonwood Distributors, Inc.—a warehouse example

Cottonwood Distributors, Incorporated (CDI) is an existing, well-established distribution company. CDI acts as a broker for commodity parts. The company has been in business for many years and has been a loyal DB2® Universal Database customer. CDI uses a relational database (DB2) to store their information. This information consists of hundreds of thousands of parts, thousands of suppliers for those parts, and the customers that want to order those parts. They have sales representatives that enter orders through their client applications. Sales representatives take orders over the phone, and supply bids to customers that are looking for different parts. Customer sales follow-up is outsourced to an external vendor (who also managed their human resources processes). CDI uses Extensible Markup Language (XML) for their correspondence and reporting. The system has been under the care of very skilled database administrators that have fine-tuned it and kept the system stable for many years.

CDI has challenges pertaining to the size and location of the data. The challenges intensify when they acquire two of their competitors in the industry, MyDogwood, Incorporated, and MyOak, Incorporated. The two new companies have similar operations, and just as much data as the parent company, but the data exists in different databases. MyDogwood, Incorporated has data in an Oracle database, and MyOak, Incorporated has data in an Informix® database.

Following the merger, CDI has heterogeneous data sources in a variety of formats. These data sources are on systems that are already loaded and configured to run successfully, with lots of untrusted users. The processes must handle millions of parts and thousands of suppliers. So that CDI can stay competitive, they must replace the sales representatives on the phone with thousands of Web users that request bids and place their orders online. In addition, suppliers want to be able to submit quotes for parts on the Internet. So, CDI introduces a Web-based brokerage exchange. Customers will access the parts information and order parts on the internet. Suppliers will access the internet and provide quotes on parts. The sales follow-up and human resources processes continue as outsourced functions. CDI must now handle the added challenges of managing issues of accessibility of data, currency of data, and real-time updates across multiple databases.

Related reference:

- Appendix A, "Script examples for Cottonwood Distributors, Inc. and YBar, Inc. scenarios," on page 217

Discovering the data—the employee skills scenario

A company called YBar, Incorporated, specializes in human resources, identifying skills and company needs, and looking at resumes and assigning new hires. The company has an employee database that contains information about people and their work skills. They have to work with a flat file that is managed by an unrelated application existing outside of the database. This application manages job profiles. In addition, they have to deal with an application owned by a solutions provider outside of the company.

They have two major challenges.

1. They must identify the best candidates to fill job openings.

   These are mostly internal job openings, and they base their search on skills that are found on personnel resumes. They have a Web application that collects resumes from candidates in an Extensible Markup Language (XML) format. The Human Resources system (HR) keeps track of current job profiles for all employees.
2. They must send the list of employees (including their current job description) to the external education provider so they can offer customized classes to the right people.

They will use WebSphere® MQ for sending messages in their applications. Then, they can use federated systems to join the employee database and the job database, and publish the list over WebSphere MQ.

YBar, Incorporated has a table called *Employee* that contains information about the employees in the company. It also has a table called *Job* that describes the jobs.

Table 2. YBar, Inc. Employee and Job tables and columns

EMPLOYEE	JOB
Emp_ID	Job_ID
Lastname	Job_Description
Firstname	Title
Dept_ID	Responsibilities
Current_Job_ID	

Related concepts:

- "Employee database scenario - solution design" on page 170

Related reference:

- Appendix A, "Script examples for Cottonwood Distributors, Inc. and YBar, Inc. scenarios," on page 217

Information integration components

Information integration uses proven data management technologies (the DB2 Universal Database family) as the foundation of IBM DB2 Information Integrator. DB2 Universal Database uses parallelism within a single system and over clustered systems and can support thousands of concurrent users and terabytes of data. Its object-relational underpinnings provide an extensible framework for adding new data types with tailored functions. The information integration infrastructure provides client application programming interfaces (APIs) that use messaging and workflow facilities, such as those that are provided by IBM WebSphere. A database event, such as the arrival of a new piece of information, can transparently create a notification, such as putting a new message on a queue. WebSphere Studio provides an open, integrated development environment for Java-based database and federated database applications. It provides graphical capabilities to perform various functions such as the following tasks:

- Create SQL requests
- Understand join paths
- Develop stored procedures
- Extend the database with user-defined functions
- Turn database requests into Web services
- Develop Extensible Markup Language (XML) schemas and document type definitions (DTDs)

The DB2 Universal Database family—a foundation for information integration

| DB2® Universal Database is the foundation technology of IBM® DB2 Information
| Integrator. DB2 Universal Database™ can manage various kinds of information
| whether the information is stored in DB2 Universal Database, Oracle, Sybase, or
| other databases.

DB2 Extenders™ can manage images, video, audio, or voice recordings, Extensible Markup Language (XML) documents, complex text documents, spatial objects, and more. DB2 Universal Database Data Links Manager can manage data in an external file system. DB2 Universal Database can handle referential integrity, access control, consistency, and recovery. DB2 XML Extenders, Net Search Extender, and Spatial Extenders provide data type-specific extensions to query, access, update, and manage various data objects. For example, with DB2 extenders, you can access XML documents, query by image shape or color, or query by a given location.

DB2 Universal Database and its associated components provide you with the ability to store and retrieve the following types of data:

- Structured relational tables
- Semi-structured XML documents
- Unstructured content such as byte streams and scanned images

For more information about DB2 Universal Database, see http://www.ibm.com/software/data/db2/.

DB2 XML Extender

DB2 XML Extender serves as a repository for XML documents and their document type definitions (DTDs). It also provides data management functions, such as data integrity, security, recoverability, and manageability. You can store an entire document as an XML user-defined column or decompose the document into multiple tables and columns. XML elements and attributes can use indexes to ensure a faster search. You can retrieve an entire document or extract XML elements and attributes dynamically in an SQL query. In addition, the XML Extender provides stored procedures to create XML documents from existing data.

DB2 XML Extender can use functions and stored procedures that allow access to DB2 Universal Database message queuing functions directly from DB2 XML applications. You can use WebSphere® MQ tools to develop the message query functions and stored procedures. DB2 XML Extender contains some message query functions. With these functions, your application can do the following things:

- Use SQL statements to send, read, or receive documents as XML messages in the service points (queues) as defined in the application messaging interface.
- Compose an XML message from tables and send it directly to the message queue, or decompose an XML message from the queue into relational tables.

Net Search Extender

You can use IBM Net Search Extender with the built-in federated support of DB2 Information Integrator to index and search text data that is stored in DB2 Universal Database and Informix® IDS databases. You can also use IBM Net Search Extender in federated sources such as Oracle, Sybase, and Microsoft® SQL Server. Integration with the intelligent strategies of the database manager optimizer ensures high performance and a full-text search that works seamlessly within an SQL full select.

Spatial Extender

With DB2 Spatial Extender, you can obtain facts and figures that pertain to objects that can be defined geographically (such as in terms of their location on earth, or within a region of the earth). These facts and figures are the spatial information, and the things are the geographic features. For example, you can use DB2 Spatial Extender to determine whether any populated areas overlap the proposed site for a landfill. The populated areas and the proposed site are features. A finding as to whether any overlap exists is an example of spatial information. If overlap exists, the extent of it is also be an example of spatial information.

To produce spatial information, DB2 Spatial Extender must process data that defines the locations of features. Such data, called spatial data, consists of coordinates that reference the locations on a map or similar projection. For example, to determine whether one feature overlaps another, DB2 Spatial Extender must determine where the coordinates of one of the features are situated with respect to the coordinates of the other.

You can use DB2 Spatial Extender to exchange spatial data between your database and the external data sources. More precisely, you can import spatial data from external sources by transferring it to your database in files, called data exchange files. You can also export spatial data from your database to data exchange files from which external sources can acquire it.

DB2 Warehouse Manager

A data warehouse is a collection of data that is obtained from a variety of data sources. End users obtain the data in a way that they can understand and use in a business context. You can group the data about the business or organizational structure by using warehouse tools to build, manage, and analyze data that is extracted from heterogeneous environments.

DB2 Warehouse Manager provides SQL-based extract, transform, and load capabilities to move and transform data. In addition, DB2 Warehouse Manager includes a metadata management solution, the Information Catalog Center. The Information Catalog Manager also provides an integration point for third-party independent software vendors to perform bidirectional metadata and job-scheduling exchange. DB2 Warehouse Manager includes a distributed extract, transform, and load job-scheduling system. DB2 Warehouse Manager agents support direct data movement between source systems and target systems without the added cost of a centralized server. DB2 Warehouse Manager supports full refresh, incremental updates, and data movement options including IBM's integrated data replication functions.

Replication

With replication technology, you can replicate data between your central database and regional transactional databases. Replication makes business data available to the regional databases for prompt transaction processing. In a replication environment, your business can capture data changes from a source database and propagate the changes to any target database, without requiring application changes. Replication provides data transformation that uses standard SQL, including multitable joins and stored procedures. Replication supports point-in-time and near real-time replication with embedded transformation for populating data warehouses and data marts.

Replication also supports the ability to reconstruct DB2 Universal Database transactions and publish those transactions as messages in XML. Replication can also send the transactions as messages to WebSphere MQ message queues that can then be consumed by a message subscribing application or the Q Apply program. Event publishing captures data by means of the Q Capture program, another component of Q Replication. Event publishing allows you to publish committed transactional or row-level data from DB2 Universal Database tables as messages in

| WebSphere MQ message queues. The messages can be read and interpreted
| directly by user applications, or they can first be interpreted by a message broker
| such as WebSphere Business Integration Message Broker or a DB2 MQ listener
| daemon.

Related concepts:

- "What solutions does data warehousing provide?" in the *Data Warehouse Center Administration Guide*
- "Introduction to XML Extender" in the *DB2 XML Extender Administration and Programming*
- "How XML data is handled in DB2" in the *DB2 XML Extender Administration and Programming*
- "Replication in the Data Warehouse Center" in the *Data Warehouse Center Administration Guide*
- "Introduction to Q replication—Overview" in the *IBM DB2 Information Integrator Replication and Event Publishing Guide and Reference*
- "Introduction to event publishing—Overview" in the *IBM DB2 Information Integrator Replication and Event Publishing Guide and Reference*
- "The purpose of DB2 Spatial Extender" in the *IBM DB2 Spatial Extender and Geodetic Extender User's Guide and Reference*
- "How to use DB2 Spatial Extender" in the *IBM DB2 Spatial Extender and Geodetic Extender User's Guide and Reference*
- "How features, spatial information, spatial data, and geometries fit together" in the *IBM DB2 Spatial Extender and Geodetic Extender User's Guide and Reference*

Related tasks:

- "Method for retrieving an XML document" in the *DB2 XML Extender Administration and Programming*
- "Planning for SQL replication" in the *IBM DB2 Information Integrator SQL Replication Guide and Reference*

Related reference:

- "Fullselect" in the *SQL Reference, Volume 1*

Web services capabilities in information integration

Web services are key innovations for integration:

- Web services promote interoperability: Web services design the interaction between a service provider and a service requester to be completely platform independent and language independent.
- Web services enable just-in-time integration: As service requesters use service brokers to find service providers, the discovery takes place dynamically.
- Web services reduce complexity through encapsulation: Service requesters and providers concern themselves with the interfaces necessary to interact with each other. As a result, a service requester has no idea how a service provider implements its service, and a service provider has no idea how a service requester uses its service. Web services encapsulates those details inside the requesters and providers.
- Web services technologies allow you to cast older applications as a Web service. This means that you can use the applications and packages that are already in place in your enterprise in interesting new ways. In addition, the infrastructure

associated with the older applications (such as security, directory services, and transactions) can be *wrapped* as a set of services as well.

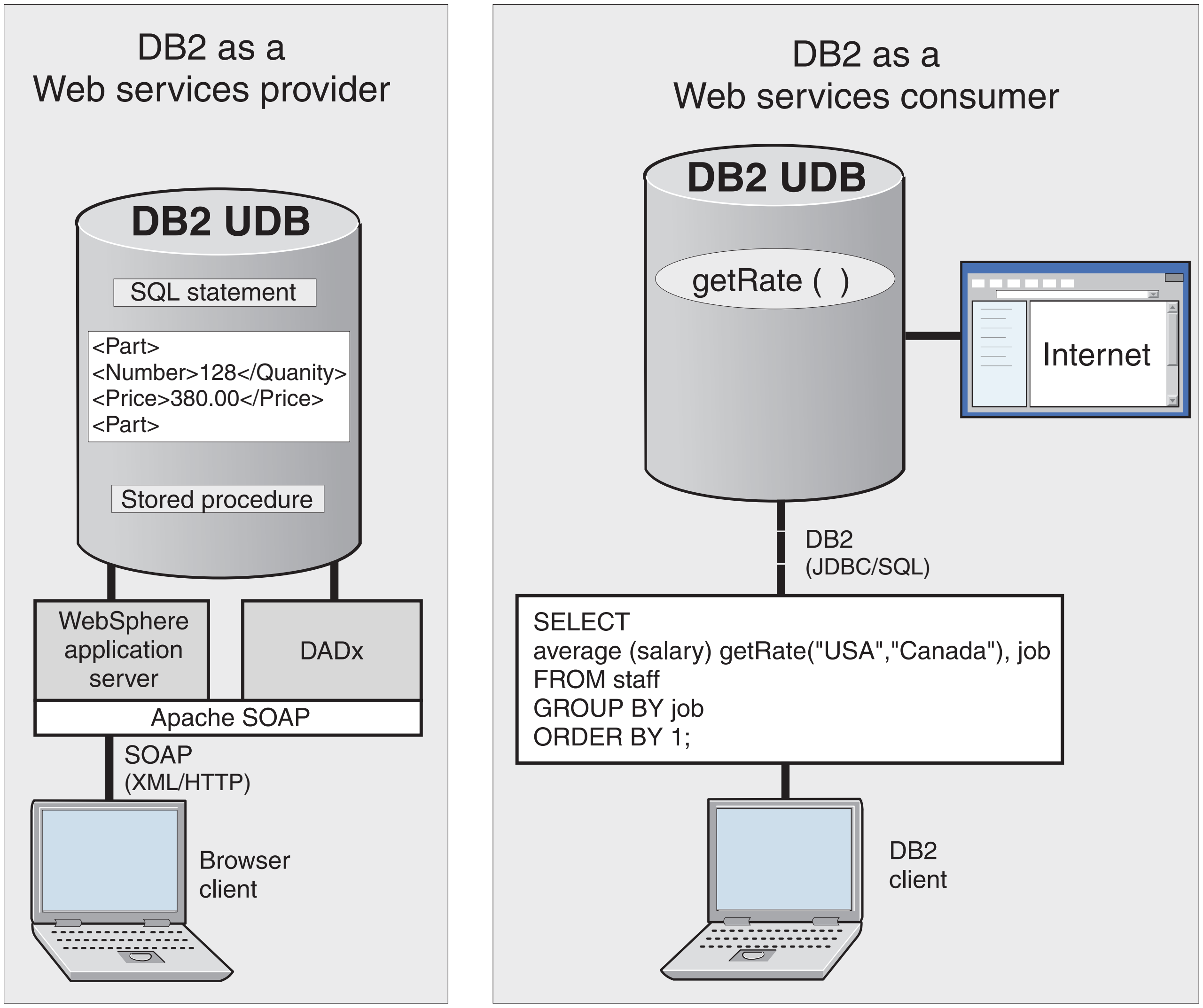
Web services provide a simple interface between the provider and consumer of application resources using a Web Service Description Language (WSDL).

Web services provider
: A Web services client application can obtain access to a DB2® Universal Database with a Web services description language (WSDL) interface. You can create a WSDL interface to DB2 UDB data by using the Web services Object Runtime Framework (WORF), also known as Document Access Definition Extension (DADX) files. After you define the operations to access DB2 UDB data with the DADX file, then you deploy the DADX file and its runtime environment (Apache SOAP version 2.3 or Apache Axis version 1.2) to a supported Java™ Web application server environment (Apache Jakarta Tomcat or IBM® WebSphere® Application Server). After you have the DB2 Web service tested and deployed, any Web services client can start using the DB2 Web service.

Web services consumer - the user-defined functions
: When DB2 Universal Database™ becomes the consumer, Web services can take advantage of the optimization that is built within the database. By using SQL statements, you can consume and integrate Web services data. By using SQL to access Web services data, you can reduce some application programming efforts because the data can be manipulated within the context of an SQL statement before that data is returned to the client application. You can convert an existing WSDL interface into a DB2 UDB table or scalar function by using tools that are provided in WebSphere Studio version 5 and later. During the execution of an SQL statement, you establish a connection with the Web service provider, and then you receive a response document as a relation table or a scalar value.

Web services consumer - the Web services wrapper
: Within the federated systems, a Web services wrapper is available to allow users to access Web services with SQL statements on nicknames and views that invoke Web services. You can create a Web services wrapper and nicknames that specify input to the Web service and access the output from the Web service with SELECT statements.

Figure 2 on page 18 shows the participation by DB2 Universal Database in the Web services environment:



```
<Part>
<Number>128</Quanity>
<Price>380.00</Price>
<Part>
```

```
SELECT
average (salary) getRate("USA","Canada"), job
FROM staff
GROUP BY job
ORDER BY 1;
```

Figure 2. Web services provider and the SOAP user-defined functions

IBM DB2 Information integrator uses many Web services capabilities, such as the following capabilities:

- The ability to expose stored procedure functionality as a Web service.

  By using the Web Services Object Runtime framework (WORF) and Document Access Definition Extension (DADX), application servers can serve these Web services to clients. The applications servers might be WebSphere Application Server or Apache Tomcat.
- The ability for all SQL statements that DB2 UDB executes, including stored procedures to become SOAP clients and request Web services from SOAP servers. The Web service then presents the data either as a SQL value or as a table that you can combine with other SQL data.

The WebSphere Application Server is infrastructure software for dynamic e-business. The WebSphere Studio application development environment provides the tools that you need to build, deploy, and integrate your e-business.

WebSphere Application Server is a J2EE-compliant application server that provides an environment for open distributed computing. WebSphere Application Server provides a middle ground between a client and the resource management systems (such as databases). It allows clients (such as applets or C++ clients) to interact with data resources (such as relational databases or WebSphere MQ) and with existing applications.

Related concepts:

- "Introduction to using DB2 as a Web services provider – WORF" on page 25
- "Web services provider features" on page 30
- "The Web service consumer functions" on page 145
- "Overview of the Web services process" on page 32
- "The Web services wrapper and the Web services description language document" in the *IBM DB2 Information Integrator Data Source Configuration Guide*

Related tasks:

- "Registering nicknames for Web services data sources" in the *IBM DB2 Information Integrator Data Source Configuration Guide*

WebSphere MQ

IBM® WebSphere® MQ (formerly known as IBM MQSeries®) is used for dynamic integration. It connects applications through a simple consistent programming interface or noninvasive adapters on a variety of platforms across all of the major networking systems. WebSphere MQ allows systems to operate independently, but assures delivery of information. It includes message encryption through a Secure Sockets Layer (SSL) for extra security and enhanced performance. Because of its reliability and robustness, you can use WebSphere MQ in mission-critical, high-value solutions across all industries today.

WebSphere MQ provides support for applications with a number of application programming interfaces:

Message Queuing Interface
Message queuing is one method of program-to-program communication. Message Queuing allows programs to send and receive application-specific data without direct connections. Programs communicate by sending or retrieving messages to or from named queues. Programs do not need to know the location of the named queues. You can replicate programs for availability or performance. You can relocate programs or queues.

Application Messaging Interface
The MQSeries Application Messaging Interface is a simple application programming interface that provides support for point-to-point messaging and publish and subscribe messaging. The Application Messaging Interface simplifies application development by moving function from the application program into a data repository. The three essential parts of the Application Messaging Interface syntax are the service, the policy, and the message. The service defines where to send the message. The policy defines how to send the message. The message is what is sent. The service encapsulates local or remote queues. The policy encapsulates options for the message such as *priority* or *retry*. The message part might contain application message data and attributes such as format or correlation identifiers.

Java™ Messaging Services
The Java Messaging Services application programming interfaces allow applications to create, send, receive, and read messages. They enable asynchronous and reliable communications.

Use the WebSphere messaging facilities to receive, process, and store information. Then use DB2® Warehouse Manager to coordinate the collection and process the message data. IBM DB2 Information Integrator acts as the source and destination for Extensible Markup Language (XML) information that is processed by MQSeries Integrator. The WebSphere messaging functions speed implementation of distributed applications by simplifying application development and testing, and by using a consistent application programming interface across all platforms.

Related concepts:

- "DB2 Information Integrator—the solution to integration" on page 5
- "How to use WebSphere MQ functions within DB2" on page 200

Related tasks:

- "Installing DB2 WebSphere MQ functions" on page 190

Planning and testing your applications

You need to plan for the kinds of data objects (for example databases, tables, nicknames) that you will be using, and how you can best use those objects. Then you need to determine how to deploy the objects. For example, you can define views for abstract models. Then support more complex models by nesting views. You can use a view to mask multiple tables that use a variety of data sources. Then, you can manipulate the view in a Web application, and make it available on the Internet.

Installation planning

Planning for the installation of the information integration infrastructures involves understanding your current environment, and knowing what components solve your business solutions the best. Most of the tools and environments that you need require at least 256 MB of random access memory (RAM) on a Windows® platform. You also need to examine the software and hardware configuration requirements for each component in terms of disk space, communication setups, prerequisites, and maintenance. See *DB2 Information Integrator Installation Guide* for more information on considerations for installing federated systems.

Related concepts:

- "DB2 Information Integrator—the solution to integration" on page 5
- "Planning for the information integration architecture" on page 5

Related reference:

- "DB2 Information Integrator installation worksheet" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Configuring your applications and environments

Configure your database client and server so that they can connect to each other. You can use Transmission Control Protocol/Internet Protocol (TCP/IP) as a communications network. On a Windows® environment, you can add entries into the *services* file on each system to specify a service name and port number.

In a typical configuration, the WebSphere® Message Queue server and the DB2® servers reside on the same machine. The DB2 clients can be local or remote, using WebSphere servlets, Enterprise Java™ beans, or Web applications.

The federated systems technologies do not require that you install any software on the machine that hosts the data source. The federated databases communicate with the data source through a client server architecture, using the source's normal client. In this way, the federated data source looks like just another application to the source.

You create a federated system by installing the DB2 Universal Database™ engine, and then enabling the federated functionality. You then configure the federated system to talk to the data sources. There are several steps to add a new data source to a federated system. First, you must install a wrapper for the source. Then you must tell the federated database where to find this wrapper. Do this by issuing a CREATE WRAPPER statement. You only need one wrapper for multiple sources if they have the same type. For example, even if the federated system includes five Oracle database instances, possibly on different machines, you only need one Oracle wrapper. You issue one CREATE WRAPPER statement. However, you must identify each separate source to the system with a CREATE SERVER statement. If there are five Oracle database instances, then issue five CREATE SERVER statements.

Be sure to set the properties on the database for the server and the federated systems (SVCENAME, FEDERATED). Connect to the database from the server. For each data source you want to access, create the necessary wrapper objects, server objects, and user mappings.

```
db2 update dbm cfg using svcename myID authentication server
db2 update dbm cfg using federated yes
db2 connect reset
db2stop
db2start
db2 connect to rdjdb user user1 using pass1word
db2 create wrapper net8 options (DB2_FENCED 'N')
db2 create server oracle8 type oracle version 8.1.5
    wrapper net8 authorization oracleuser1
    password oraclepwd
    options (node 'orafcle8.world', password 'Y', pushdown 'Y')
db2 create user mapping for user1
    server oracle8
    options (REMOTE_AUTHOID 'oracleuser1',
       REMOTE_PASSWORD 'oraclepwd')
...
```

Figure 3. Example of database connection and wrapper set up

Related concepts:

- "Federated systems" in the *Federated Systems Guide*
- "How you interact with a federated system" in the *Federated Systems Guide*

Related tasks:

- "Configuring and running MQListener" on page 207

Related reference:

- "Checklist for planning your federated system configuration" in the *IBM DB2 Information Integrator Data Source Configuration Guide*

Performance and tuning planning— materialized query tables in a federated system

A materialized query table is a table that is defined based on the result of a query. A materialized query table mechanism allows administrators to define materialized views of data in a set of underlying tables, or nicknames. See the *Federated Systems Guide* for more information on federated systems objects and their definitions. For certain classes of queries, the database can automatically determine whether the materialized query table can answer a query, without accessing the base tables.

By using materialized query tables, you can transparently route read-only queries to the data cache, while you transparently route updates to the database. Replication asynchronously propagates the changed data to the cache according to a user-specified policy. By using federated caching and replication, a user can ask complex queries more effectively.

You can use materialized views that use aggregate tables for better performance in an e-business environment. For example, in e-commerce, you can use materialized views to cache product catalog information on mid-tier servers to improve the performance of browsing the catalog, without involving the summary data. DB2® Universal Database supports caching by allowing you to define materialized views over nicknames that are used to define a remote table. You populate the materialized view by pulling data from the remote table and storing it locally, which results in significant performance benefits. You get the best performance when you include the REFRESH DEFERRED parameter with the materialized views. The materialized query tables, or materialized views, keep the frequently referenced data close to the application server. The materialized query tables or views also provide a means to buffer the finely tuned systems from the traffic generated by Web applications.

You can define materialized query tables on nicknames. A nickname is an identifier that is used to reference the object located at the data source that you want to access. The nicknames identify objects, known as data source objects. By using a nickname to reference a materialized query table, the local DB2 UDB instance can cache the remote data. The caching capability results in better performance for federated queries, because the queries access the remote data locally. If the remote table is not available, DB2 UDB can use the materialized query table that is defined on the remote table if it meets the routing criteria. The results of this technique are improved availability and performance. The REFRESH IMMEDIATE option is not applicable to a materialized query table that refers to nicknames.

By using materialized query tables, you can avoid repeating calculations, such as represented by the SUM table, for each query. Assume that you have a table called CUSTOMER_ORDER that stores customer orders for several years. The table has over one million records, with an average row width of 400 bytes. Now, assume that you need to run multiple queries on orders for the year 2001 and that you only need three columns from the table. Here is a typical SQL statement to get the information from the three columns:

```
select SUM(AMOUNT), trans_dt
  from db2inst2.CUSTOMER_ORDER
  where trans_dt between '1/1/2001' and '12/31/2001'
  group by trans_dt
```

If there are good indexes on the CUSTOMER_ORDER table, then the access path shows an index scan for this statement.

However, you can create a materialized query table that contains the columns and rows that you need, including the calculation for the grand total:

```
CREATE TABLE DB2INST2.SUMMARY_CUSTOMER_ORDER_2001
AS
(SELECT SUM(AMOUNT) AS
  TOTAL_SUM, TRANS_DT, STATUS
  FROM DB2INST2.CUSTOMER_ORDER
  WHERE TRANS_DT
  BETWEEN '1/1/2001' AND '12/31/2001'
  GROUP BY TRANS_DT, STATUS)
DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

If you use the clause DATA INITIALLY DEFERRED, your data is not inserted into the table as part of the CREATE TABLE statement. Issue a REFRESH TABLE statement to populate the table. The data in the table reflects the result of the query as a snapshot at the time that you issue the REFRESH TABLE statement. To populate the materialized query table that you created, issue the following statement:

```
REFRESH TABLE DB2INST2.SUMMARY_CUSTOMER_ORDER_2001;
```

The queries that run on the materialized query table are faster. The materialized query table is smaller in size and its rows are short (just 45 bytes, as opposed to 400 bytes in the base table).

By using materialized query tables, you can optimize queries that join multiple tables across several servers on the LAN. With materialized query tables, you can cache the answer set on a single server, and update that cache when data changes in any of the servers. Any type of federated relational data can be a candidate for these types of joins. This includes Oracle tables, SQL Server, Sybase, message queues, Web services, and other relational data sources.

Plan to use a data warehouse to remove direct access to your operational, or day-to-day data. Performance is improved because you are accessing the warehouse data for ad hoc queries. By placing your date in a data warehouse, you create stores of informational data that you can extract from the operational data and then transform for decision making. For example, create the following table, joining a customers table and an account table, to store the customer and account information for bad accounts:

```
CREATE TABLE bad_account AS
     (SELECT customer_name, customer_id, a.balance
       FROM account a, customers c
       WHERE status IN ('delinquent', 'problematic', 'hot')
       AND a.customer_id = c.customer_id)
       DATA INITIALLY DEFERRED REFRESH DEFERRED
```

If a user asks whether an account is delinquent, the DB2 Universal Database™ optimizer recognizes that the materialized query table has cached the requested information. Instead of accessing the base table *account*, DB2 Universal Database accesses table *bad_account*. This provides a better response time and returns the customer information.

IBM® federated systems includes a cost-based optimizer. The optimizer considers not only standard database statistics, such as cardinality and indices, but also network and server resources, and the query power available in the data source engine.

Related concepts:

- "Nickname characteristics affecting pushdown opportunities" in the *Federated Systems Guide*

Related tasks:

- "Creating a materialized query table" in the *Administration Guide: Implementation*

Security and authorization

In distributed computing systems, where users, application servers, and resource managers, are often spread out across the world, securing computing system resources is a complicated task. A good security service provides two main functions: authentication and authorization.

Authentication takes place when a principal (a user or a computer process) initially tries to gain access to a computing resource. At that point, the security service challenges the principal to prove that the principal is valid. Human users typically prove their identity by entering their user IDs and passwords. A process normally presents an encrypted key. If the password or key is valid, the security service gives the user a token or ticket. This token identifies the principal and authenticates the principal. After the authentication, the principal tries to use the resources within the boundaries of the computing system protected by the security service. However, a principal can use a particular computing resource only with the proper authorization.

Authorization takes place when an authenticated principal requests the use of a resource. The security service determines if the user can use that resource. Typically, you associate access control lists (ACLs) with resources to handle authorization. The resources define which users or processes (or groups of users or processes) are authorized to use the resource. If the security service authorizes the principal, the principal gains access to the resource. In a distributed computing environment, principals and resources must be mutually suspicious of the identity of each other until each has proven their identity to the other. This is necessary because a principal might fake its identity to get access to a resource. That resource might get valuable information from the principal. To solve this problem, the security service contains a security server that acts as a trusted third party. It authenticates principals and resources so that these entities can prove their identities to each other.

The authenticated user must have the appropriate privileges (such as SELECT, INSERT, UPDATE, or DELETE). The privileges must be on both the nickname in the federated database and on the underlying table or other object at the remote data source. This is how the federated database can accept the request. It is not sufficient to have the privileges on the local DB2® Universal Database. See the *Federated Systems Guide* for more information on the authorizations and privileges.

Related concepts:

- "Advantages of designing queries in IBM DB2 Information Integrator" on page 164
- "Security in DADX Web services" on page 26

Chapter 2. Developing Web services

This section explains the development and use of Web services. IBM DB2 Information Integrator contains a Web services provider (WORF), and two types of Web services consumers (the SOAP user-defined functions, and Web services wrappers).

Introduction to Web services provider

Web services are sets of business functions that applications or other Web services can invoke programmatically over the Internet by using a Web Service client interface. In IBM DB2 Information Integrator, 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.

Introduction to using DB2 as a Web services provider – WORF

You can use Web services to enable remote access to DB2® Universal Database information. Web services include a set of application functions that perform some useful service on the behalf of a consumer, or a requester, such as informational or transactional functions. Web services perform functions, which can be anything from simple requests to complicated business processes. The consumer generally only needs to know the Web services description language interface to the Web service. In addition, the Web service can change usually without affecting the consumer, unless a change is made in the interface.

Web services promote interoperability. The reality of interoperability assumes that the information technology industry uses a set of standards that provide guidance on the development and integration of Web services. The Web Services Interoperability Organization is an open industry effort that is chartered to promote and ensure Web services interoperability across platforms, applications and programming languages. The Web Services Interoperability Organization uses specifications that are developed by the World Wide Web Consortium (W3C) and the UDDI.org. Web services interoperability means that you can create your Web services on a variety of SOAP or Web Services platforms, including Apache SOAP, Apache Axis, or Microsoft® Visual Studio.Net.

The Web service application programmer designs the interaction between a service provider, and a consumer, or service requester to be completely independent of platforms and languages. You can use just-in-time integration, because service requesters can find service providers dynamically. Web services reduce complexity through encapsulation. Service requesters and providers are concerned only with the interfaces necessary to interact with each other, not their underlying implementation. Web services give new life to legacy applications because you can cast an existing application as a Web service. The basic elements of Web services include simple object access protocol (SOAP), Universal Description, Discovery, and Integration (UDDI), and Web services description language (WSDL).

Web services allow you to access data from a variety of databases and internet locations. After you have accessed the data, a Web service consumer can search, mine, and then transform the data to use it with a data warehouse for further analysis.

You can define a Web service to access data in the database by using a simple Document Access Definition Extension (DADX) file. You can create this DADX file, which is an XML file, by using a simple text editor, or by using the WebSphere® Studio Application Developer and the wizards available from WebSphere Studio.

The DADX file drives the Web services run-time environment, which includes various database management tools and the Web Object Runtime Framework (WORF). The WORF runtime environment provides a simple mapping of XML schema to SQL data types. The DADX file can contain standard SQL statements, such as SELECT, INSERT, UPDATE, DELETE, and CALL statements to query and update a database and call stored procedures. If you want to process SQL statements at runtime, then you must enable the dynamic query service (DQS) that is provided by WORF by using a DADX file that includes only the <DQS/> tag.

If you do not use the WORF runtime environment, you need to write your own program to handle the details of creating the Web service, such as developing your own WSDL. Some of the functions that WORF provides include the following:
- Analyzing the Web service request
- Connecting to the database
- Executing the SQL request
- Encoding the output message from the SQL results
- Returning the message back to the client

The DADX file can also contain DB2 XML Extender elements, such as Document Access Definition (DAD) file references, XML collection operations to generate and store XML documents, or user-defined types (UDT), and user-defined functions (UDF). The DAD file defines a mapping between XML and relational data. DB2 XML Extender allows XML documents to be stored intact, and optionally indexed in side tables. DB2 XML Extender does this by using the XML column access method, or as a collection of relational tables by using the XML collection access method.

WORF is available with IBM® DB2 Information Integrator, as well as DB2 Universal Database™ Version 8 and WebSphere Studio Version 5. WORF also works with Informix®.

Related concepts:
- "Web services provider features" on page 30
- "Definition of a DADX file" on page 29
- "Overview of the Web services process" on page 32

Security in DADX Web services

You can secure Web services by using the security mechanisms of your application server. The mechanisms discussed here are authentication, encryption, and securing the database user ID.

Authentication

You can secure the DADX Web service endpoints by enabling authentication. When you enable authentication, you ensure that only those people who are authenticated can call your Web service. Access control in Java™ 2 Enterprise Edition (J2EE) is specified by the URL. Each DADX Web service uses URLs for different parts of the Web service, such as the endpoint and during the generation of the actual WSDL, and in the test page. For each URL that DADX uses, you can specify which roles (and as part of that definition you can specify which users) are allowed to access that URL. The process of setting authorizations for authenticated users of Web services is similar to granting SELECT privileges on a DB2® Universal Database table.

The following list shows examples of URLs and URL patterns that you can use to protect Web services, specific operations, or services provided by WORF :

- Enable access control for all URLs of a DADX, the test pages, and the WSDL generation by using the following URL:
  ```
  http://hostname:port/myContext/myGroup/myDadx.dadx/*
  ```
- Enable access control for only the test page by using the following URL:
  ```
  http://hostname:port/myContext/myGroup/myDadx.dadx/TEST
  ```
- Enable access control for all of the DADX Web services in a group by using the following URL:
  ```
  http://hostname:port/myContext/myGroup/*
  ```

You can define security constraints in the Application Server Toolkit or Application Assembly tool of WebSphere® Application Server Version 5. You can also define security constraints in the Web perspective of WebSphere Studio Application Developer Version 5. The constraints that you define can include role names, so that anyone with that particular role name can access the Web area.

Encryption

You can secure the DADX Web services by encrypting messages through HTTPS. Encryption ensures that nobody can read the messages that are exchanged between the Web service client and the Web service provider. See the documentation that is part of your application server to determine how to enable HTTPS.

Database security

In WebSphere Application Server Version 5, you can specify the database user ID on the application server for a JNDI data source. In the group.properties file, you can refer to that JNDI data source so that the DB2 Web service provider uses the user ID that you specify in WebSphere. Your database user password is encrypted on the application server.

For more information on authentication, encryption and data source authentication, see the information related to your particular application server. Also see the following WebSphere documentation for specific information on security:

- *IBM WebSphere Application Server, Version 5: Security*
- *IBM WebSphere V5.0 Security WebSphere Handbook Series*

Related concepts:

- “WebSphere Studio” in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Preparing and creating the Web archive file" on page 136
- "Customizing the group.properties file" on page 64

Related reference:

- Appendix E, "Web services command reference," on page 263
- Appendix D, "Web services encoding algorithm," on page 261

Using Web services provider with iSeries

Web services are sets of business functions that applications or other Web services can invoke programmatically over the Internet. One use for Web services is the enabling of remote access to DB2® Universal Database information. Web services include a set of application functions that perform some useful service on the behalf of a requester such as informational or transactional functions. Web services perform functions, which can be anything from simple requests to complicated business processes. The requester generally only needs to know the application programming interfaces (APIs) to the Web service. In addition, the Web service can usually change without affecting the requester.

Web services promote interoperability. Web services design the interaction between a service provider, and a service requester to be completely independent of platforms and languages. You can use just-in-time integration, because service requesters can find service providers dynamically. Web services reduce complexity through encapsulation. Service requesters and providers are concerned only with the interfaces necessary to interact with each other, not their underlying implementation. Web services give new life to legacy applications because you can cast an existing application as a Web service. The basic elements of Web services include simple object access protocol (SOAP), Universal Description, Discovery, and Integration (UDDI), and Web services description language (WSDL).

Web services allow you to collect data from a variety of databases and Internet locations. After you have collected the data, the Web services consumer can search and mine the data and transform it with subsets of the data warehoused for further analysis.

You can define a Web service that implements standard SQL statements, such as SELECT, INSERT, UPDATE, DELETE, and CALL statements. You can also define these Web services by using the DB2 XML Extender stored procedures.

DB2 XML Extender uses an XML document format called Document Access Definition (DAD) to define the mapping between XML and relational data. The Document Access Definition Extension (DADX) file specifies a Web service. It does this by using a set of operations that are defined by SQL statements, by a list of parameters, and by DAD file references. Operations are similar to programming methods that you can invoke. You can use XML collection operations to generate and store XML documents. You can use SQL operations to query and update the database and call stored procedures.

You can access DB2 Universal Database™ stored procedures and data from a Web service by using available database management tools and the Web Object Runtime Framework (WORF). These tools allow you to invoke stored procedures and SQL statements as Web service operations, in addition to specifying storage and retrieval operations on XML data. WORF provides simple mapping of XML schema to SQL data types.

When using stored procedures, you must authorize the *PGM object that is created for each CREATE PROCEDURE statement. When you use Java™ stored procedures, you should authorize the user (or *PUBLIC) to the Java class file. When you use Java stored procedures, store all of the class files in the following directory:

```
/QIBM/UserData/OS400/SQLLib/Function
```

Make sure that *Spserver.class* is in this directory. The stored procedure SAMPLE.TESTRS is the only stored procedure that is an SQL stored procedure. It has no dependency on the Java class.

To define the sample stored procedures and catalog them in DB2 Universal Database, use the following steps:

1. >qsh
2. `>cd /QIBM/UserData/WebASAEs4/worf/`
   `installedApps/servicesApp.ear/services.war/WEB-INF/`
   `classes/groups/dxx_sample`

   where **WebASAEs4** is the version of WebSphere®, and **worf** is the name of the WebSphere instance.
3. >db2 -f Spcreate.db2

To remove the stored procedure definitions, execute the following command:

```
>db2 -f Spdrop.db2
```

WORF is available as part of IBM® DB2 Information Integrator. It is also available with DB2 Universal Database Version 8 and WebSphere Studio Version 4 and Version 5. When delivered with WebSphere Studio, you can use the tools to automate the building of DADX Web services. These tools include a wizard to create DADX files that are based on SQL statements or DAD files. It also includes tools to create DAD files. WORF also works with Informix®.

You can use DB2 XML Extender to implement Web services by using the WORF runtime environment with DB2. DB2 XML Extender consists of stored procedures, user-defined types (UDT), and user-defined functions (UDF). You can use these features to store and retrieve XML data by using DB2. DB2 XML Extender allows XML documents to be stored intact, and optionally indexed in side tables. It does this by using the XML column access method, or as a collection of relational tables by 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.

Related concepts:

- "Overview of the Web services process" on page 32

Related tasks:

- "Defining the web.xml and group.properties files in the iSeries platform" on page 62
- "Installing and deploying the WORF examples in iSeries" on page 54
- "Installing the Web services provider software requirements on iSeries" on page 37

Definition of a DADX file

| A document access definition extension (DADX) file specifies how to create a Web
| service. A Web service is a function that you invoke over the Web. You can create

the Web service by using a set of operations that are defined by SQL statements, stored procedure calls, or DAD files. Web services store Extensible Markup Language (XML) documents or retrieve XML documents, including some that are managed by DB2® XML Extender. Web services that are specified in a DADX file are called *DADX Web services*, or IBM® DB2 Information Integrator Web services.

WORF provides the run-time support for invoking DADX documents as Web services. These Web services use the Apache Simple Object Access Protocol (SOAP) (Version 2.3 or later) engine, or the Apache Axis engine (Version 1.2). Both of these SOAP engines are supported by WebSphere® Application Server and Apache Jakarta Tomcat.

The contents of the DADX file determine if the Web service will use a set of predetermined SQL operations or dynamic SQL operations. If the DADX file contains a dynamic query services tag (<DQS/>), then you can specify the SQL operations from a browser or embed the operations in an application if you installed the WORF test Web application.

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

Related concepts:
- "Defining the Web service with the document access definition extension file" on page 66
- "Dynamic database queries that use the Web services provider" on page 90

Related reference:
- "A simple DADX file" on page 75

Web services provider features

WORF provides the following features:
- "Resource-based deployment"
- Automatic service redeployment, at development time, when defining resource changes
- Hypertext Transfer Protocol (HTTP) GET and POST bindings, in addition to SOAP
- WSDL and XSD file generation, including support for UDDI best practices, which represents industry standards
- Documentation and test page generation
- Generation of the Web service inspection language (WSIL) page in HTML and XML format

A key feature of WORF is that it supports resource-based deployment of Web services. Resource files, such as DADX files, describe the Web services to WORF, so that WORF can generate the appropriate Web services from these files. When you request the resource file, WORF loads the file 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.

You can create your own resource files. The resource files must conform to specific syntax and semantic 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 you can deploy the Web services properly.

In addition to specifying storage and retrieval operations on Extensible Markup Language (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 procedure result sets have fixed metadata. Fixed metadata refers to data with a fixed number and a fixed shape, which implies a certain number of columns, with certain column names and data types. The operation signature includes the input and output parameters. You can execute stored procedures when you use dynamic query services (DQS) provided by WORF, with no fixed set of metadata or result sets that are required You can also specify SQL statements to select, insert, update and delete data. And, WORF provides simple mapping of Extensible Markup Language (XML) schema to SQL data types. These particular features do not require the XML Extender.

Related concepts:
- “WSDL from a DADX file” on page 87
- “Accessing the Web service with GET, POST, and SOAP bindings” on page 111
- “Web services automatic reloading” on page 127
- “Web services documentation” on page 126
- “Web services that exist from Web services provider” on page 120

Related reference:
- “Syntax of the DADX file” on page 67

Web service provider operations used with DADX files

DADX files support three kinds of Web service operations: non-dynamic SQL operations, dynamic SQL operations, and XML collection 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. Your application returns the data by using only a simple mapping of SQL data types, using column names as elements.

SQL operations: non-dynamic
The SQL operations can be non-dynamic. Non-dynamic operations are those that are predefined within the DADX file. There are three elements that make up the predefined SQL operations type:

<query>
Queries the database

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

<call> Calls stored procedures that can return 0 or more result sets

SQL operations: dynamic
The SQL operations can be dynamic operations, depending on the content of the DADX file. Dynamic operations are those that are generated in a SOAP message with no predefined SQL operations. The following elements are dynamic operations:

<getTables>
Retrieves a description of available tables.

<getColumns>
Retrieves a description of columns.

<executeQuery>
Issues a single SQL statement.

<executeUpdate>
Issues a single INSERT, UPDATE, DELETE.

<executeCall>
Calls a single stored procedure.

<execute>
Issues a single SQL statement.

Extensible Markup Language (XML) collection operations (requires DB2 XML Extender)
These storage and retrieval operations help you to map XML document structures to DB2 Universal Database™ tables. You can either compose XML documents from existing DB2 data, or decompose (storing untagged elements or attribute content) XML documents into DB2 data. This method is useful for data interchange applications, particularly when the application frequently updates the contents of XML documents.

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.

Related concepts:
- "Introduction to using DB2 as a Web services provider – WORF" on page 25
- "Testing Web services applications – a scenario" on page 109
- "Dynamic database queries that use the Web services provider" on page 90

Related reference:
- "DADX operation examples" on page 80
- "Dynamic query service operations in the Web services provider" on page 99

Overview of the Web services process

The following is an overview of the steps that are needed to make DB2® Universal Database a Web services provider. The Web application developer creates the following general hierarchy with the steps:

```
Web application -> group ->
  DADX file (the Web service) -> the SQL operations
```

If you use enterprise archive files, then the general hierarchy is:

```
Enterprise application -> Web application ->
  group -> DADX file (the Web service) -> the SQL operations
```

1. The database administrator sets up the databases.
2. The database administrator optionally enables the databases for DB2 XML Extender. The retrieveXML and storeXML operations require DB2 XML

Extender. Using the XML column also requires the DB2 XML Extender. 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 an enterprise archive or Web archive (EAR or WAR) file. When the EAR or WAR file is deployed, it becomes a folder containing the Web application where he can do further modifications. The EAR file is an enterprise application that is described in the Java™ 2 Enterprise Edition specification (J2EE). WAR files are also described in the J2EE specification. The Web application folder is on a server that is 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 5 documentation

     You can create WAR or EAR files on WebSphere Application Server.
   - Apache Jakarta Tomcat documentation

     You can create only WAR files on Tomcat.
4. The Web application developer then creates a group and a group.properties file for that group. The group.properties file contains information about the database connection and other related information used by WORF. A group is a number of Web services operations that access a database. You can have one group for each database or even multiple groups for the same database, and within that group, you can define one or more DADX files. The DADX file, which specifically defines the Web service contains the operations that execute the Web service. See Defining the web.xml and group.properties files for more information on group properties.
5. The database developer optionally creates the DAD to map XML and relational data conversion (required when you use XML Extender stored procedures).
6. The Web services developer creates the DADX document. The content of the DADX file determines if you can use dynamic queries in your Web application. A DADX file with a dynamic query services tag (</DQS>) contains only that tag which acts as a switch to enable dynamic queries. A non-dynamic DADX file defines a set of operations and contains information that is used to create the Web service. See Syntax of the DADX file for more information on the rules for creating DADX files.
7. Optional: When you use the WebSphere Application Server configuration manager for Windows® or UNIX®, the Web service developer creates and deploys a deployment descriptor for the Web service. A deployment descriptor is either an **isd** file (used with the Apache SOAP engine) or a **deploy.wsdd** file (used with the Apache Axis engine), that identifies configuration and deployment information. On the Apache Axis engine, deployment descriptors are created automatically by WORF. Each Web service that uses Apache SOAP can contain one **.isd* file. A Web application can contain multiple Web services. Copy all the isd or wsdd files into the dds.xml file. (This step is automatic for users of Apache Jakarta Tomcat). See Generating deployment descriptors for more information on deployment descriptors.
8. You can verify the Web service by using the DADX test page that is available if you deploy the WORF examples that are shipped with IBM® DB2 Information Integrator. You can copy the Java Server Pages from the WORF directory in the apache-services.war file or the axis-services.war file to test some of the WORF functionality in your application.

Keep in mind that in some environments, these tasks might be performed by a single individual.

Related concepts:

- "Dynamic database queries that use the Web services provider" on page 90
- "Defining a group of Web services" on page 58

Related tasks:

- "Deploying a federated application" on page 181
- "Generating deployment descriptors" on page 133
- "Defining the web.xml and group.properties files" on page 59
- "Deploying WORF examples on WebSphere Application Server Version 4.0.4 for z/OS or OS/390" on page 44
- "Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX" on page 41

Related reference:

- "Syntax of the DADX file" on page 67

Installing and configuring the Web services provider

Determine the capacity of your system and plan for the software that needs to be installed for your application programming interfaces and web service applications.

Web services provider software requirements for UNIX and Windows

You set up Web services object runtime framework (WORF), or the Web services provider, on any of the following operating systems:

- Windows NT®
- Windows 2000®
- Linux
- AIX®
- Solaris Operating Environment

In addition to being packaged with IBM DB2 Information Integrator, WORF is also part of DB2 Universal Database Extended Server Edition, Version 8, and WebSphere Studio Version 5. WORF is in the following path in DB2 Universal Database Version 8: *<DB2 UDB installed location>\samples\java\Websphere\dxxworf.zip*.

You can use the following database environments:

- IBM DB2 Universal Database™ Version 8 or later

  http://www.ibm.com/software/data/db2. This includes DB2 XML Extender.
- Informix Dynamic Server (IDS) Version 9.3

Additionally, use the following software (depending on the server that you use, most of this software might already be part of your environment):

- Java™ Java Development Kit (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 5, (http://www.ibm.com/software/webservers/appserv/)
  - Apache Web server:
    - Apache Jakarta Tomcat Version 3.3.1 through 4.0.3 or later (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 the Xerces parser to your CLASSPATH to use it as the XML parser
  - Apache SOAP 2.3 or later binary, or Apache Axis 1.2 (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/). Apache SOAP requires JavaBeans Activation Framework Version.
  - j2ee.jar, version 1.3 or later (http://java.sun.com/)
  - qname.jar (http://java.sun.com/)
  - *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j

Related concepts:
- "Definition of a DADX file" on page 29

Related tasks:
- "Installing and deploying the WORF examples on Apache Jakarta Tomcat" on page 52
- "Installing the Web services provider software requirements" on page 36

Web services provider software requirements for OS/390 and z/OS

You can set up the Web services provider (Web object runtime framework) in any of the following operating systems:
- OS/390 Version 2.8 or later
- z/OS Version 1.1 or later

You can use the following database environments:
- IBM DB2 Universal Database for OS/390 Version 7 or DB2 Universal Database for z/OS (http://www.ibm.com/software/data/db2/os390)
- IBM DB2 XML Extender for OS/390 Version 7 or later (http://www.ibm.com/software/data/db2/extenders/xmlext/index.html). Required for store and retrieve operations

Additionally, use the following software:
- WebSphere Application Server Version 4.01 Service Level W401505 or later
- JavaMail Version 1.2 (http://java.sun.com/)
- JavaBeans Activation Framework Version 1.0.1(http://java.sun.com/)
- j2ee.jar, version 1.3 or later (http://java.sun.com/)
- qname.jar (http://java.sun.com/)

- IBM XML Toolkit for z/OS and OS/390 Version 1.4 with program temporary fix (PTF) UW95866 (http://www.ibm.com/servers/eserver/zseries/software/xml/)
- *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.

Related tasks:

- "Testing the Web service" on page 109
- "Deploying WORF examples on WebSphere Application Server Version 4.0.4 for z/OS or OS/390" on page 44

Configuring the Web services provider for WebSphere Application Server on UNIX, Windows, z/OS, and OS/390

You can run Web services on WebSphere® Application Server Advanced Edition. Web services object runtime framework (WORF) provides the run-time support for invoking document access definition extension (DADX) documents as Web services over Hypertext Transfer Protocol (HTTP) with Apache SOAP 2.3 (or later) or Apache Axis 1.2 (or later). WebSphere Application Server 5 or higher, and other servlet engines support this. WebSphere lets you secure your SOAP Web services. See the WebSphere documentation on securing SOAP services for more information. The following sections describe the Web services.

Installing the Web services provider software requirements

Prerequisites:

Ensure that you have the required software installed. See Web services provider software requirements for UNIX and Windows to verify your installation in Windows and UNIX.

You need the DB2® XML Extender for advanced mapping control between XML and relational data. Verify the DB2 UDB installation by creating the DB2 UDB SAMPLE database, if it is not already created. The Web services requires Java Database Connectivity (JDBC) 2.0, which is the default in DB2 Universal Database Version 8.

See Web services provider software requirements for OS/390 and z/OS to verify your z/OS installation.

Procedure:

The procedures to prepare the Web services environment in UNIX and Windows are as follows:

1. Stop any services that use DB2 Universal Database (such as WebSphere Application Server)
2. Stop DB2.
3. For DB2 Universal Database versions earlier than Version 8, select Java Database Connectivity (JDBC) 2.0. Run the *C:\SQLLIB\java12\usejdbc2.bat* file, assuming that you installed DB2 in C:\SQLLIB\ when using a Windows environment.
4. Restart DB2.
5. Start WebSphere Application Server Advanced Edition 5.1 from its install directory. These instructions assume that you installed WebSphere Application Server in a Windows environment in C:\WebSphere\Appserver.

The procedures to prepare the Web services environment on OS/390 or z/OS are as follows:

1. Create a new directory to store application extensions, if one does not already exist
2. Set the APP_EXT_DIR environment variable in your designated J2EE server instance to this application extensions directory
3. Add the following JAR files to the application extensions directory:

   xerces.jar
   This file is in the IBM XML Toolkit for z/OS which you can download from http://www.ibm.com/servers/eserver/zseries/software/xml/.

   mail.jar
   This file is in JavaMail

   activation.jar
   This file is in the Java Beans Activation Framework

   j2ee.jar
   You can download this file from http://java.sun.com/products

   qname.jar
   You can download this file from http://java.sun.com/products

   wsdl4j.jar
   You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.
4. Verify the configuration of the J2EE server instance with the following steps:
   - Ensure that the soap.jar included with WebSphere Application Server is part of your CLASSPATH
   - Add to the jvm.properties file of the J2EE server instance the following settings:

   ```
   com.ibm.ws390.server.classloadermode=2
   com.ibm.ws.classloader.ejbDelegationMode=false
   ```
5. Restart the J2EE server.

Related tasks:

- "Web services provider software requirements for OS/390 and z/OS" on page 35
- "Installing or migrating WORF on Apache Jakarta Tomcat" on page 51
- "Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX" on page 39

Related reference:

- "Web services provider software requirements for UNIX and Windows" on page 34

Installing the Web services provider software requirements on iSeries

Prerequisites:

Ensure that you have the required software installed. Create the SAMPLE database from interactive SQL with the following command:

```
CALL QSYS/CREATE_SQL_SAMPLE('SAMPLE')
```

You need the DB2 XML Extender for advanced mapping control between XML and relational data. To use DB2 Universal Database XML Extenders, make sure that you install the product. You can verify that you have DB2 Universal Database XML Extenders on your system by issuing the CL command, **GO LICPGM**. For DB2 for iSeries, V5R2, if you have DB2 Universal Database XML Extenders, the following entries display as a result of the GO LICPGM command:

- 5722DE1 *COMPATIBLE DB2 UDB Extenders
- 5722DE1 *COMPATIBLE DB2 UDB Text Extender
- 5722DE1 *COMPATIBLE DB2 UDB XML Extender
- 5722DE1 *COMPATIBLE Text Search Engine

Enable DB2 Universal Database XML extenders with the following CL command: **CALL PGM(QDBXM/QZXMADM) PARM(enable_db LOCALRDB)**. LOCALRDB is the *LOCAL database name in the relational database directory. To work with the relational database entries, issue the following CL command: **WRKRDBDIRE**. If you use the document type definitions (DTDs) that are in the sample files, execute the script *setup-dxx.cmd*. WORF requires Java Database Connectivity (JDBC) 2.0, which is the default in DB2 Universal Database Version 8.

Procedure:

The procedures to prepare the WORF environment are as follows:

1. Stop any services that use DB2 (such as WebSphere Application Server)
2. For DB2 Universal Database versions earlier than Version 8, select Java Database Connectivity (JDBC) 2.0. Run the *C:\SQLLIB\java12\usejdbc2.bat* file, assuming that you installed DB2 in C:\SQLLIB\ when using a Windows environment.
3. Start WebSphere Application Server Advanced Edition 4.01 or 5.0, assuming that you installed WebSphere Application Server in C:\WebSphere\Appserver.

Related concepts:

- "Using Web services provider with iSeries" on page 28

Related tasks:

- "Installing the Web services provider software requirements for Apache Jakarta Tomcat on iSeries" on page 53
- "Defining the web.xml and group.properties files in the iSeries platform" on page 62
- "Installing and deploying the WORF examples in iSeries" on page 54

DTD definitions for XML Extender

Ensure that the database administrator has set up any databases or subsystems required for the application, and enables them for use by DB2 XML Extender (if you use XML Extender). The following table lists the default locations that the XML Extender samples reference.

Table 3. XML Extender samples reference the following document type definitions (DTDs)

Platform	Default location of DTDs
DB2 UDB Version 7.2 FixPak 7 or later Windows	`c:\dxx\samples\dtd\getstart.dtd` `c:\dxx\dtd\dad.dtd`

Table 3. XML Extender samples reference the following document type definitions (DTDs) (continued)

Platform	Default location of DTDs

DB2 UDB Version 8 Windows	`c:\<DB2 UDB installed location>\samples\db2xml\dtd\getstart.dtd`
	`c:\<DB2 UDB installed location>\samples\db2xml\dtd\dad.dtd`

DB2 UDB Version 8 on Solaris Operating Environment	`/opt/IBMdb2/V8.1/samples/db2xml/dtd/dad.dtd`
DB2 UDB Version 8 on AIX	`/usr/opt/db2_08_01/samples/db2xml/dtd/dad.dtd`
DB2 UDB Version 8 on Linux	`/usr/IBMdb2/V8.1/samples/db2xml/dtd/dad.dtd`
DB2 UDB Version 7 on OS/390 and z/OS or DB2 UDB Version 8 on OS/390 and z/OS	`/u/USER/dxx/dtd/dad.dtd`

The following is a list of some of the files that reference dad.dtd:
- department.dad
- department2.dad
- departmentStd.dad
- order.dad
- order-public.dad
- getstart.xml
- order-10.xml
- sales_db.nst

Related tasks:
- "Converting a document type definition to an XML schema" on page 86

Related reference:
- "DADX operation examples" on page 80

Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX

Prerequisites:

Install WebSphere Application Server on your work station in a path such as C:\WebSphere\Appserver (in a Windows environment).

Procedure:

To migrate to WORF Version 8.2 from an earlier version, refer to the Migration summary section. To install WORF Version 8.2, complete the following steps:

1. Unzip *dxxworf.zip* to a directory, such as *C:\worf* so that the directory has the following contents:
   - *readme.html*

- *lib\apache-services.war* and *lib\axis-services.war* - sample Web applications that contain Web services which use WORF.
- *lib\worf.jar* - WORF library. You install this on the class path of the servlet engine
- *lib\worf-servlets.jar*
- *schemas* - Extensible Markup Language (XML) schemas for the DADX and namespace tables (NST) XML files, including `wsdl.xsd`, `db2WebRowSet.xsd`, and `dadx.xsd`.
- *tools* - The tools directory contains the DAD and DADX checker tools.

2. Verify that the directory of the server you are using (such as WebSphere Application Server) contains the appropriate Web services engine jar files. If the files are not in the directory, copy the jar files from the directory that contains the Apache Axis or Apache SOAP files so that you can enable the appropriate Web services engine.
   - If you are using the Apache Axis framework, copy axis.jar to c:\WebSphere\AppServer\lib. Then, copy the contents of axis/lib to c:\WebSphere\AppServer\lib to access the other Apache Axis JAR files.
   - If you are using the Apache SOAP framework copy soap.jar to WebSphere\AppServer\lib.
3. Copy *worf.jar* to *C:\WebSphere\AppServer\lib*.
4. If your WebSphere server is a release earlier than WebSphere 5.0.2, you must download a file from http://java.sun.com/xml/downloads/saaj.html, named `saaj.jar`. Copy `saaj.jar` to *C:\WebSphere\AppServer\lib*.
5. Start the WebSphere Application Server.
6. Open the Administrator's console by selecting Start—> Programs —>IBM WebSphere —>Administrator's Console.
7. Configure WebSphere to run with your DB2 UDB environment:
   a. From the left navigation pane, click Servers —> Application Servers.
   b. Find the name of your server in the right content pane and click on the server name.
   c. Click Process Definition —> Java Virtual Machine.
   d. On the Configuration page, specify the class path as the path to the Java database information. If you installed DB2 Universal Database in directory sqllib\, and you use the group.properties file that comes with the WORF samples, the following example is a valid path:
   ```
   C:\SQLLIB\java\db2java.zip
   ```
   e. Click **Apply** or **OK**.
   f. Save the configuration.
8. Stop the WebSphere Application Server.

Migration summary

To migrate to WORF Version 8.2 from an earlier version of WORF:
1. Replace the worf.jar file by copying the *lib\worf.jar* from the Version 8.2 dxxworf.zip to *C:\WebSphere\AppServer\lib*. The dxxworf.zip is located in the following path: *<DB2 UDB installed location>\samples\java\Websphere\dxxworf.zip*.
2. For each application that you deployed, replace the JSP files in the worf directory of that application, with the files in the worf directory of the apache-services.war, or the axis-services.war. Then re-deploy the application.

Related concepts:

- "Defining a group of Web services" on page 58

Related tasks:

- "Installing the Web services provider software requirements" on page 36

Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX

Prerequisites:

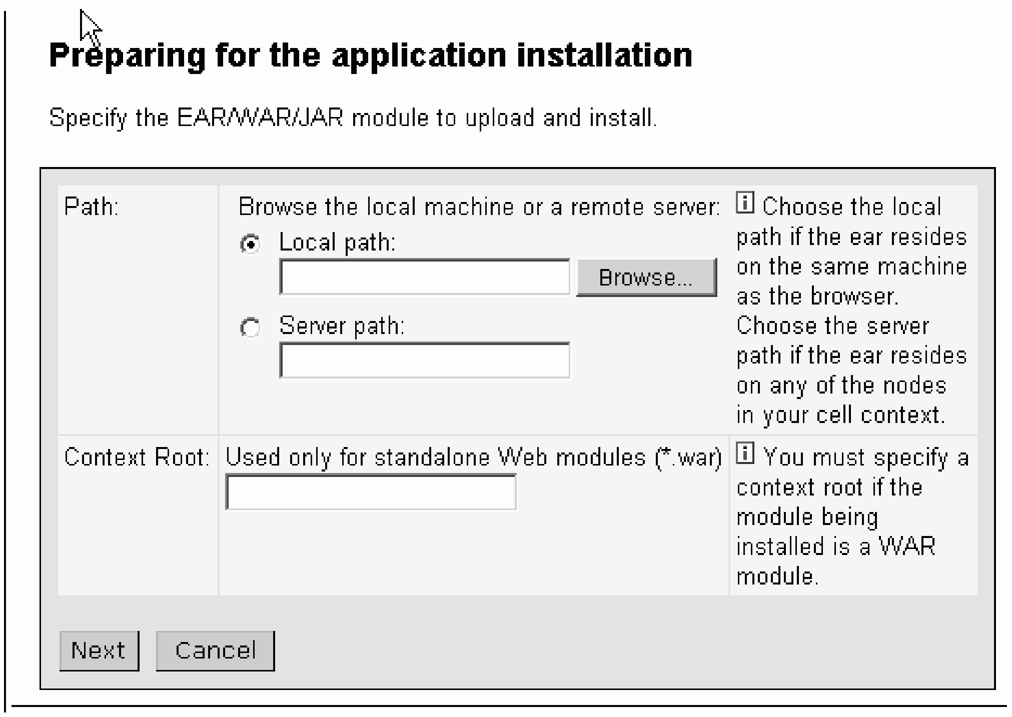
- Install WebSphere Application Server on your work station in a path such as C:\WebSphere\Appserver (on your Windows environment).
- Install WORF.

Procedure:

To install and deploy the WORF examples, complete the following steps:

1. Start the WebSphere Administration Server.
2. Open the Administrator's console by selecting Start —> Programs —> IBM WebSphere —> Administrator's Console.
3. Select Applications —> Enterprise Applications. The content window displays all of the enterprise applications that you installed on the current server.
4. Install *apache-services.war* or *axis-services.war* as an enterprise application by clicking the **Install** push button.
   a. From the **Local path** field, click the **Browse** push button to locate the path to the correct services file that is included in the c:\WORF\lib directory. WORF ships two services files for you to choose from when running the samples. These files are apache-services.war and axis-services.war. Select the correct services file for the SOAP engine that you are running.
   b. Specify a context name for the Web application in the **Context Root** field. To execute the examples discussed here, you must specify `services` as the Context Root name.

      Figure 4 on page 42 shows the WebSphere Application Server Administrator's Console during the installation of the application:


Preparing for the application installation

Specify the EAR/WAR/JAR module to upload and install.

Path:	Browse the local machine or a remote server: [✓ Local path:] Browse... Server path:	Choose the local path if the ear resides on the same machine as the browser. Choose the server path if the ear resides on any of the nodes in your cell context.
Context Root:	Used only for standalone Web modules (*.war)	You must specify a context root if the module being installed is a WAR module.

Figure 4. Specification of the application or module

c. Click **Next**.

d. Accept all of the other defaults and click **Next** for the remainder of the Wizard. On the **Map virtual hosts for web modules** window, select the `.WAR` file and click **Next**. On the **Map modules to application servers** window, select the `.WAR` file and click **Next**. The configuration options specify a virtual host (for example: `default_host`) and an application server (for example: `Default Server`). At the end of the Wizard, click **Finish**.

e. The final window displays the Save to Configuration. Click **Save**.

5. Verify that the database settings are correct (especially user ID and password) in the group.properties files.

6. Issue *setup.cmd* in a DB2 Universal Database command window in a Windows environment (the DB2 UDB Command Line Processor window), or *setup.sh* in a UNIX environment command window in each of the database directories to create the database. For example, run setup.cmd in the *dxx_sales_db* directory to set up the SALES_DB database that uses DB2 XML Extender.

   Attention: If you issue the setup command in the dxx_sample directory, the command drops and then recreates the SAMPLE database. If you use the SAMPLE database that is shipped with the DB2 Universal Database product, be aware that you will lose modifications that you have made to the database.

7. If you deployed your own application, copy the worf-servlets.jar file from the WORF directory to WebSphere/AppServer/installedApps/<host>/<application WAR directory>/WEB-INF/lib .

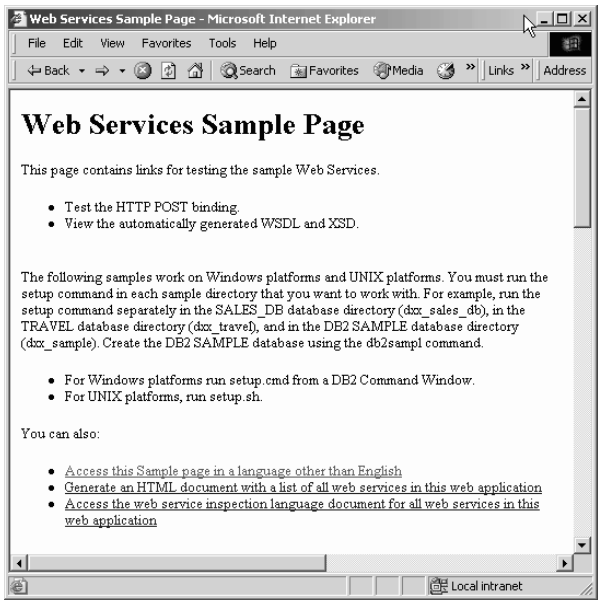
8. Stop the current server.

9. Restart the server.

10. Verify that services.war is already running by selecting Applications —> Enterprise Applications.

11. Open a browser window to test the installation by accessing the Web application welcome page.

    The specific port number varies according to the WebSphere Application Server configuration. If you have used the defaults, the services Web application welcome page might be *http://localhost:9080/services*. Remember that `services` is the name of the application that you created in an earlier step. The page should look like the screens shown in Figure 5 and Figure 6:



Web Services Sample Page - Microsoft Internet Explorer

Web Services Sample Page

This page contains links for testing the sample Web Services.

- Test the HTTP POST binding.
- View the automatically generated WSDL and XSD.

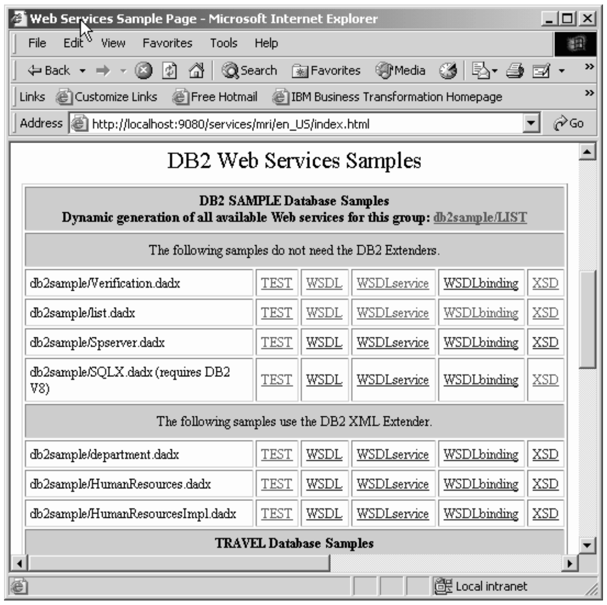
The following samples work on Windows platforms and UNIX platforms. You must run the setup command in each sample directory that you want to work with. For example, run the setup command separately in the SALES_DB database directory (dxx_sales_db), in the TRAVEL database directory (dxx_travel), and in the DB2 SAMPLE database directory (dxx_sample). Create the DB2 SAMPLE database using the db2sampl command.

- For Windows platforms run setup.cmd from a DB2 Command Window.
- For UNIX platforms, run setup.sh.

You can also:

- Access this Sample page in a language other than English
- Generate an HTML document with a list of all web services in this web application
- Access the web service inspection language document for all web services in this web application

Figure 5. WORF sample page



Web Services Sample Page - Microsoft Internet Explorer

Address http://localhost:9080/services/mri/en_US/index.html

DB2 Web Services Samples

DB2 SAMPLE Database Samples **Dynamic generation of all available Web services for this group:** db2sample/LIST					
The following samples do not need the DB2 Extenders.					
db2sample/Verification.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/list.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/Spserver.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/SQLX.dadx (requires DB2 V8)	TEST	WSDL	WSDLservice	WSDLbinding	XSD
The following samples use the DB2 XML Extender.					
db2sample/department.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/HumanResources.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/HumanResourcesImpl.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
TRAVEL Database Samples					

Figure 6. Web services sample page-the test links

12. Click on some of the links to verify that the sample services work. The test page consists of a tree view of the operations, an input view and a results view. You access the test page from the TEST link within the Welcome Page of

the samples, or by typing the following in your browser: <your-Web-server>:9080/<context_root_name>/<group_name>/<dadx file>/TEST.

Related tasks:

- "Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX" on page 39
- "Defining the web.xml and group.properties files" on page 59

Related reference:

- "Error checking by the DADX environment checker" on page 240
- "Indicating errors and warnings in the output text file" on page 239
- "Running the DADX environment checker" on page 238

Deploying WORF examples on WebSphere Application Server Version 4.0.4 for z/OS or OS/390

These steps are for deploying Web applications for use in WORF on the z/OS or OS/390 platform. You can also verify that you have correctly installed and configured WORF and its prerequisites.

Procedure:

To install WORF, complete the following steps:

1. Download and unpax *dxxworf.pax* to an empty directory, such as */u/USER/worf/*. You can unpax the file using the command:
   ```
   pax -rvf dxxworf.pax
   ```
   After you expand the file, the directory has the following contents:
   - *readme.txt*
   - *lib/apache-services.war* and *lib/axis-services.war* - sample Web applications that contain Web services which use WORF.

     Note: Note: the commands and instructions that are included here refer to services.war or services.ear. Please use the correct SOAP files for the SOAP engine you choose to run. For example, an example might refer to a services.war file, but if you installed the Apache SOAP engine, then the file is apache-services.war.
   - *lib/worf.jar* - WORF library.
   - lib/worf-servlets.jar
   - *schemas/* - Extensible Markup Language (XML) schemas for the DADX and NST XML files
   - *tools/ – The tools directory contains the DAD and DADX checker tools. See Installing the DADX environment checker for more explanation.*
2. Copy *worf.jar* to the application extensions directory of your J2EE server instance.
3. Start (or restart) the J2EE Server

To install and deploy the WORF examples, complete the following steps:

1. Configure the System Management Scripting application programming interface (API). For more information on configuring the scripting API on your OS/390 or z/OS system, see *IBM WebSphere Application Server V4 for z/OS and OS/390:*

Installation and Customization and *IBM WebSphere Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*.

2. Prepare an enterprise archive file (EAR). You use EAR files to deliver Java 2 platform enterprise edition (J2EE) applications. They consist of Web archive files (WAR) and Java archive files (JAR).
   a. In UNIX System Services (USS), copy the apache-services.war or the axis-services.war to a temporary, writable directory and change your current directory to that location.
   b. Enter the following command from the USS command line (all on one line):
      ```
      390fy -op "" -context_root "/services"
            -display_name "ServicesApp" services.war
      ```
      The command creates an initial EAR file with the name *services.ear* in the current directory. The EAR file has a Context Root of "/services" and a Display Name of "ServicesApp". The Context Root is the part of the Uniform Resource Locator (URL) that directs WebSphere Application Server to your application. The Display Name is a string that identifies your application in the Systems Management End User Interface (SM/EUI) and in USS. You can edit the Context Root and Display Name to any name you choose.
   c. Resolve the Java Naming and Directory Interface (JNDI) name mapping for services.ear. Do this by issuing the following command from the USS command line (all on one line):
      ```
      390fy -JNDIejbp "/<Sysplex>/<J2EE Server>"
                -op "_resolved" services.ear
      ```
      The terms used in the above example have the following definitions:

      <J2EE Server>
      The name of the J2EE server onto which you will deploy the application

      <Sysplex>
      The name of the Sysplex on which your J2EE server exists

      This command creates a new file with the name *services_resolved.ear* in the current directory.
3. Deploy the application

   Note: When executing the commands for this step, you must be logged into USS with a user ID that is registered as a Systems Management Administrator for WebSphere.
   a. Copy the following sample files from the WebSphere Application Server sample directory (<*WAS_Home*>*/samples/smapi/*) to the temporary directory that contains the EAR file you just created.
      - *inputcreateconversation.xml*
      - *inputprocessearfile.xml*
      - *inputcommitconversation.xml*
   b. Set the environment variable DEFAULT_CLIENT_XML_PATH to the temporary directory that contains the EAR file.
   c. Edit the file *inputcreateconversation.xml* and specify a conversation name and optionally a description of the name. The conversation name and description can be any text that you want. However, the conversation name

must remain the same when you process the input files in the next steps. Here is an example of the conversation name and description:

```
<inputcreateconversation conversationname="WORFSamples"
           conversationdescription="WORF Sample Test" />
```

Save the file *inputcreateconversation.xml*.

d. Type the following command (all on one line) from the USS command line:

```
CB390CFG -action createconversation
   -xmlinput inputcreateconversation.xml
   -output createconv.out
```

This command creates a file *createconv.out* in the temporary directory of your system and contains the results of the operation. This file is not needed except to verify the success of the application deployment.

e. Edit the file *inputprocessearfile.xml*. Specify the target J2EE server and the EAR file to deploy. The following is an example of specifying the J2EE server and the EAR file:

```
<inputprocessearfile conversationname="WORFSamples"
                  j2eeservername="BBOASR2"
earfilename="/tmp/worfsamp/services_resolved.ear"
                  processingmode="standard" />
```

Save the file *inputprocessearfile.xml*.

f. Type the following command (all on one line) from the USS command line:

```
CB390CFG -action processearfile
    -xmlinput inputprocessearfile.xml
    -output processear.out
```

This command creates a file *processear.out* in the temporary directory of your system and contains the results of the operation. This file is not needed except to verify the success of the application deployment.

g. Edit the file *inputcommitconversation.xml*. Specify the conversation name that you used in the previous steps. For example:

```
<inputcommitconversation conversationname="WORFSamples" />
```

Save the file *inputcommitconversation.xml*.

h. Type the following command (all on one line) from the USS command line:

```
CB390CFG -action commitconversation
  -xmlinput inputcommitconversation.xml
  -output commitconv.out
```

This command creates a file *commitconv.out* in the temporary directory of your system and contains the results of the operation. This file is not needed except to verify the success of the application deployment.

4. Set up the Web server.

   a. Ensure that the J2EE server allows the Context Root that you named in Step 2 on page 45. In the file *webcontainer.conf* of the server, ensure that at least one host has a specification as in the following example:

```
host.<host_alias>.contextroots=/somewebapp,/services
```

To accept any Context Root, you can use the following line of text:

```
host.<host_alias>.contextroots=/*
```

Note: Using this method to specify the context root allows you to skip this step when you deploy future applications

b. If you use the Web server plug-in to access your J2EE server, add a Service statement to the file *httpd.conf* of your Web server. This statement specifies the Context Root of your deployed application. As an example, if you specified "/services" as your Context Root in Step 2 on page 45, your new Service statement is like the following example:

```
Service  /services/*
<WAS_Home>/WebServerPlugIn/bin/was400plugin.so:service_exit
```

The <WAS_Home> is the directory in which WebSphere Application Server is installed.

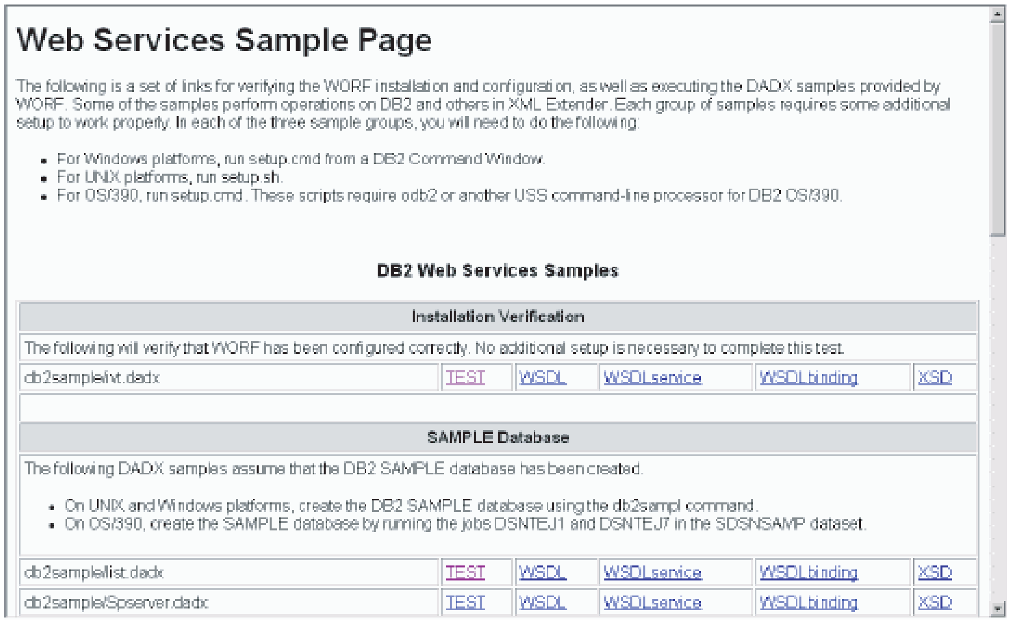
Note: The Service statement should be all on a single line

c. Restart the Web server.

5. Verify the WORF configuration.

a. Access the WORF Web Services Sample Page that is included in the sample WAR file. If you set the Context Root in Step 2 on page 45 to "/services", type the following URL:

```
http://<hostname>/services/
```



Web Services Sample Page

The following is a set of links for verifying the WORF installation and configuration, as well as executing the DADX samples provided by WORF. Some of the samples perform operations on DB2 and others in XML Extender. Each group of samples requires some additional setup to work properly. In each of the three sample groups, you will need to do the following:

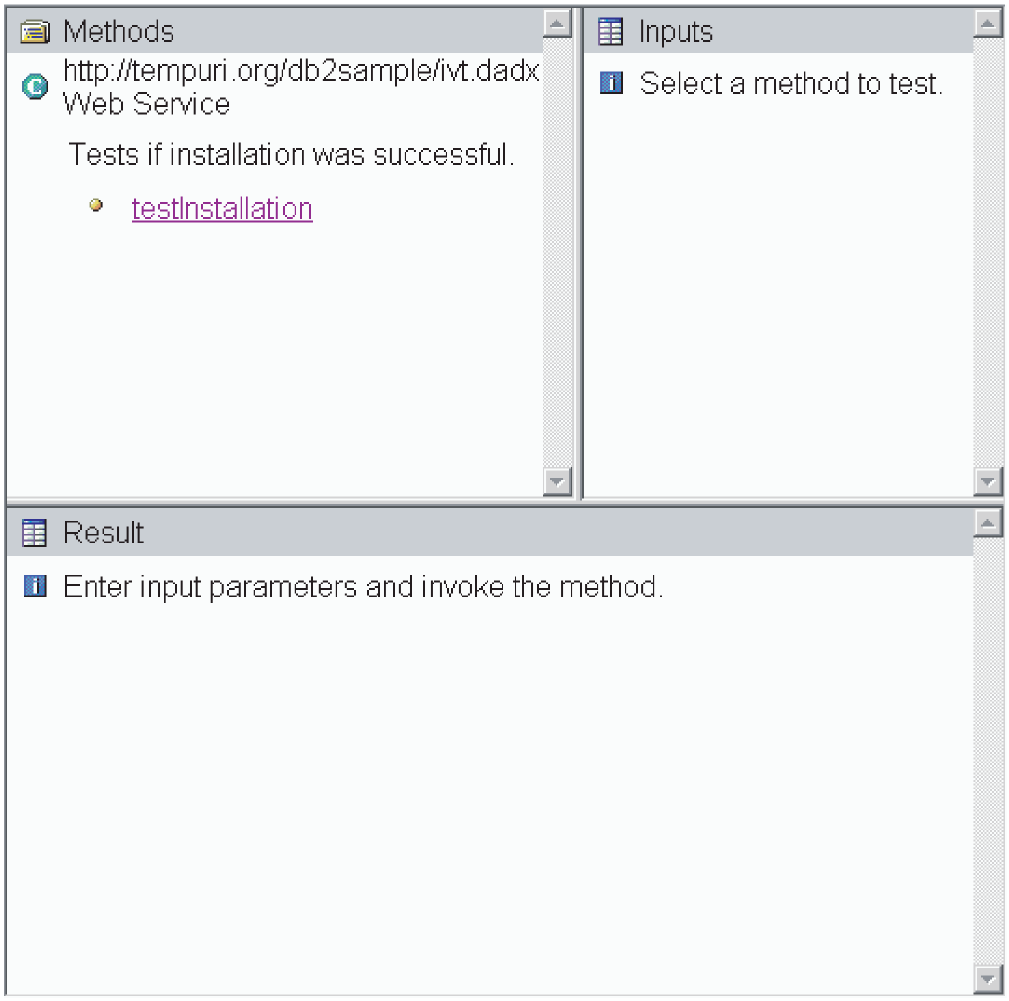
- For Windows platforms, run setup.cmd from a DB2 Command Window.
- For UNIX platforms, run setup.sh.
- For OS/390, run setup.cmd. These scripts require odb2 or another USS command-line processor for DB2 OS/390.

DB2 Web Services Samples

Installation Verification					
The following will verify that WORF has been configured correctly. No additional setup is necessary to complete this test.					
db2sample/ivt.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
SAMPLE Database					
The following DADX samples assume that the DB2 SAMPLE database has been created. • On UNIX and Windows platforms, create the DB2 SAMPLE database using the db2sampl command. • On OS/390, create the SAMPLE database by running the jobs DSNTEJ1 and DSNTEJ7 in the SDSNSAMP dataset.					
db2sample/list.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
db2sample/Spserver.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD

Figure 7. WORF sample page

b. In Figure 7, the first section of samples, titled **Installation Verification**, shows a single DADX file, *ivt.dadx*. Click on the **TEST** link. The built-in test facility of WORF opens.


Methods

http://tempuri.org/db2sample/ivt.dadx Web Service

Tests if installation was successful.

- testInstallation

Inputs

Select a method to test.

Result

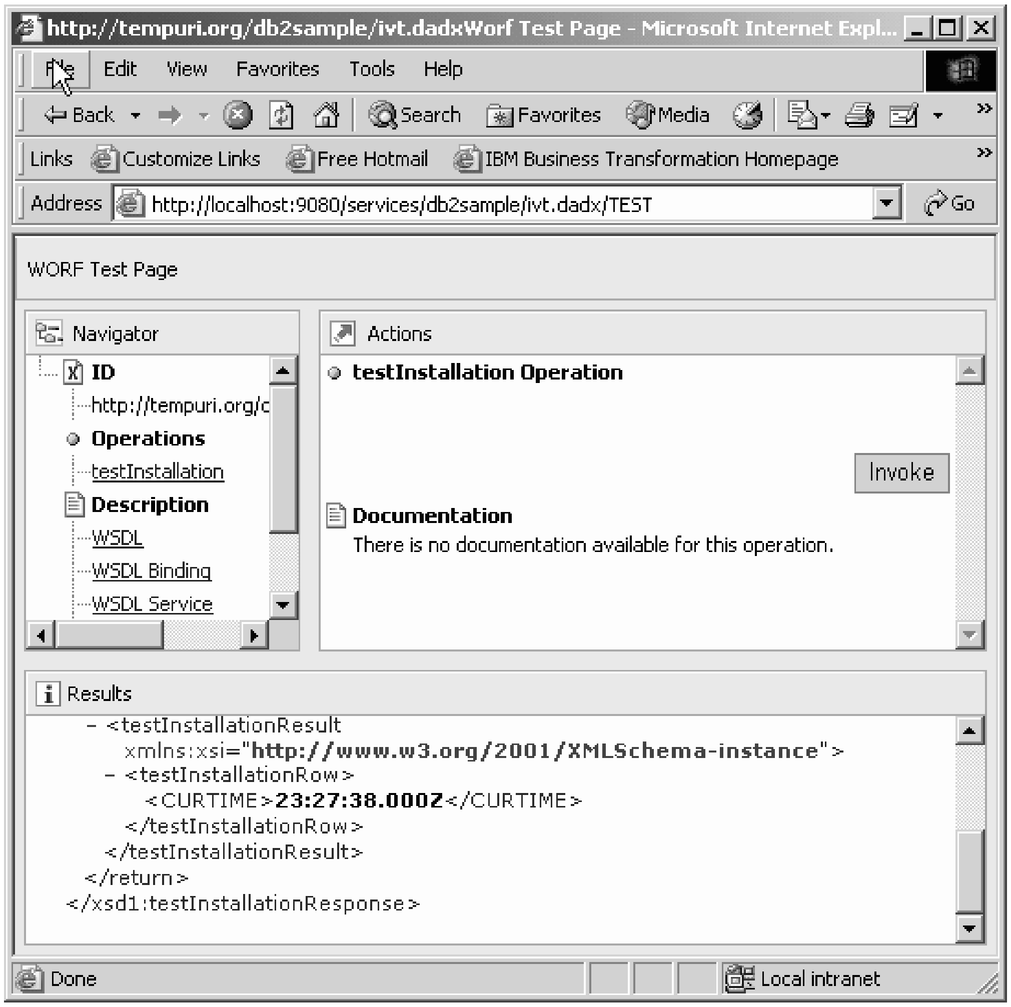
Enter input parameters and invoke the method.

Figure 8. WORF test facility

c. From the WORF test facility, Figure 8, select the “testInstallation” operation. Click on **Invoke**.

d. An XML document displays in the bottom frame of the window. Verify that the current time of day appears in the document, such as in the following example:

```
<CURTIME>14:38:26.000Z</CURTIME>
```


http://tempuri.org/db2sample/ivt.dadxWorf Test Page - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail IBM Business Transformation Homepage
Address http://localhost:9080/services/db2sample/ivt.dadx/TEST Go
WORF Test Page
Navigator
ID
http://tempuri.org/c
Operations
testInstallation
Description
WSDL
WSDL Binding
WSDL Service
Actions
testInstallation Operation
Invoke
Documentation
Results
Done
Local intranet

There is no documentation available for this operation.

```
  - <testInstallationResult
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    - <testInstallationRow>
        <CURTIME>23:27:38.000Z</CURTIME>
      </testInstallationRow>
    </testInstallationResult>
  </return>
</xsd1:testInstallationResponse>
```

Figure 9. Result of WORF test-expected output

   e. If this test fails, you have something wrong with the configuration. Verify that you correctly installed the software requirements. Also, verify that you configured the WebSphere Application Server and that you have the authorization to access DB2 Universal Database.

6. Prepare and run the examples.
   a. On the sample page (see Step 5a on page 47), there are DADX samples that are shipped with the *services.war* application. Before you run the samples, follow the setup instructions that are contained within each category.
   b. Execute an individual Web service by selecting the **TEST** link for each sample. Select the **WSDL**, **WSDLservice**, **WSDLbinding**, and **XSD** links to run those examples.

After you deploy the application, you can modify the application. You can also create additional WAR files for deployment. After you create a WAR file, deploy the Web application by using Step 2 on page 45, Step 3 on page 45, and Step 4 on page 46.

Related concepts:
- "Overview of the Web services process" on page 32

Related tasks:

- "Installing and deploying the WORF examples in iSeries" on page 54
- "Installing and deploying the WORF examples on Apache Jakarta Tomcat" on page 52

Related reference:

- "Installing the DADX environment checker" on page 237

Configuring Web services provider for Apache Jakarta Tomcat on UNIX and Windows

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

Installing the Web services provider software requirements for Apache Jakarta Tomcat on UNIX and Windows

Ensure that you have the required software installed. Verify your installation for your particular platform with the specific documentation.

Prerequisites:

You need the DB2 XML Extender for advanced mapping control between XML and relational data. Verify the installation by creating the DB2 SAMPLE database. WORF requires Java Database Connectivity (JDBC) 2.0, which is the default in DB2 Universal Database Version 8.

Procedure:

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. Run *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 4.0.6 or later binary from http://jakarta.apache.org/site/binindex.html. (Apache Jakarta Tomcat Version 4 standard comes with the appropriate Xerces parser. For earlier versions you must add the Xerces parser to your CLASSPATH to use it as the XML parser.)
     - Apache SOAP 2.3 or later binary from http://xml.apache.org/soap
     - Apache Axis 1.2 from http://www.apache.org/.
     - 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
     - j2ee.jar, version 1.3 or later.
     - qname.jar
   - *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.

Related tasks:

- "Web services provider software requirements for OS/390 and z/OS" on page 35

Related reference:

- "Web services provider software requirements for UNIX and Windows" on page 34

Installing or migrating WORF on Apache Jakarta Tomcat

Procedure:

To migrate to WORF Version 8.2 from an earlier version, refer to the Migration summary section. To install WORF Version 8.2 on Apache Jakarta Tomcat, complete the following steps: :

- To run WORF with Apache SOAP, or with Apache Axis, add the following JAR files to the class path on your application server:
  - *soap.jar for the Apache SOAP engine, or axis.jar for the Apache axis engine.*
  - *xerces.jar* (or the jars of your Java XML parser)
  - *mail.jar*
  - *activation.jar*
  - *worf.jar*
  - j2ee.jar, version 1.3 or later
  - qname.jar
  - *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.
  - *jaxrpc.jar*
  - *log4j-1.2.8.jar*
  - *commons-logging.jar*
  - *commons-logging-api.jar*
  - *commons-discovery.jar*
  - *db2java.zip*, or *jcc.jar* (or the JDBC implementation jar of your database server). The name of the driver class depends on the driver package that you use. You can modify the driver package that you use in the `group.properties` file.
- Modify the files listed in Table 4 on page 52. The modifications that you make depend on your Apache Jakarta Tomcat version and platform. Add a line for each of the jar files mentioned above. Replace <jarfile> with the actual location of the jar file. If you run Apache Jakarta Tomcat in an integrated development environment, make sure that all these jars are on the CLASSPATH that you use for starting Tomcat. The files that you need to modify are all in the directory in which you start Apache Jakarta Tomcat. You should start and stop the server with the startup.bat or shutdown.bat or startup.sh or shutdown.sh that is in the *app_server/bin* directory.

Table 4. Class path designations

Platform	Server software	File to modify	Command to add
UNIX	Apache Jakarta Tomcat 3.2.x	\bin\tomcat.sh (before ″export CLASSPATH″)	CLASSPATH = $CLASSPATH: <jarfile>
	Apache Jakarta Tomcat 3.3.x	\bin\tomcat.sh (before ″export CLASSPATH″)	CLASSPATH = $CLASSPATH: <jarfile>
	Apache Jakarta Tomcat 4.x	\bin\setclasspath.sh (before ″export CLASSPATH″)	CLASSPATH = $CLASSPATH: <jarfile>
Windows	Apache Jakarta Tomcat 3.2.x	\bin\tomcat.bat (:setClasspath section)	set CP = %CP%; <jarfile>
	Apache Jakarta Tomcat 3.3.x	\bin\tomcat.bat	set CLASSPATH = %CLASSPATH%; <jarfile>
	Apache Jakarta Tomcat 4.x	\bin\setclasspath.bat	set CLASSPATH = %CLASSPATH%; <jarfile>

- If your WebSphere server is a release earlier than WebSphere 5.0.2, you must download a file from http://java.sun.com/xml/downloads/saaj.html, named `saaj.jar`. Copy `saaj.jar` to *C:\WebSphere\AppServer\lib*.

Migration summary

To migrate to WORF Version 8.2 from an earlier version of WORF:

1. Replace the worf.jar file by copying the *lib\worf.jar* from the Version 8.2 dxxworf.zip to *C:\WebSphere\AppServer\lib*. The dxxworf.zip is located in the following path: *<DB2 UDB installed location>\samples\java\Websphere\dxxworf.zip*.
2. For each application that you deployed, replace the JSP files in the worf directory of that application, with the files in the worf directory of the apache-services.war, or the axis-services.war. Then re-deploy the application.

Related tasks:

- “Installing and deploying the WORF examples in iSeries” on page 54
- “Installing and deploying the WORF examples on Apache Jakarta Tomcat” on page 52
- “Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX” on page 39

Installing and deploying the WORF examples on Apache Jakarta Tomcat

Procedure:

To install the WORF examples, do the following tasks:

1. Unjar the *apache-services.war or axis-services.war* into your *tomcat\webapps* directory (depending on the SOAP engine that you install).

If you already have a *apache-services.war or axis-services.war* file installed, perform the following tasks:

a. Stop Apache Jakarta Tomcat.

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

Note: Any of your previously deployed Web services in the "services" web application will be lost with this action, so make sure that 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 uniform resource locator (URL). The port number, designated here by *8080* depends on your own current machine:

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

   The specific port address might vary depending on your environment. You should get a page that looks like Figure 5 on page 43. To learn more about the Web services sample page, see Testing the Web service.

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

5. To display the Extensible Markup Language (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.

Related concepts:

- "Introduction to using DB2 as a Web services provider – WORF" on page 25

Related tasks:

- "Testing the Web service" on page 109
- "Installing the Web services provider software requirements for Apache Jakarta Tomcat on UNIX and Windows" on page 50
- "Installing the Web services provider software requirements" on page 36

Installing the Web services provider software requirements for Apache Jakarta Tomcat on iSeries

Prerequisites:

Ensure that you have the required software installed. You need the DB2 XML Extender for advanced mapping control between XML and relational data. To use DB2 Universal Database XML Extenders, make sure that you install the product. You can verify that you have DB2 Universal Database XML Extenders on your system by issuing the CL command, **GO LICPGM**. For DB2 Universal Database for iSeries, V5R2, if you have DB2 Universal Database XML Extenders, the following entries display as a result of the **GO LICPGM** command:

- 5722DE1 *COMPATIBLE DB2 UDB Extenders
- 5722DE1 *COMPATIBLE DB2 UDB Text Extender
- 5722DE1 *COMPATIBLE DB2 UDB XML Extender
- 5722DE1 *COMPATIBLE Text Search Engine

Enable DB2 Universal Database XML extenders with the following CL command: **CALL PGM(QDBXM/QZXMADM) PARM(enable_db LOCALRDB)**. LOCALRDB is the *LOCAL database name in the relational database directory. To work with the relational database entries, issue the following CL command: **WRKRDBDIRE**. If you use the document type definition documents (DTDs) that are in the sample files, execute the script *setup-dxx.cmd*.

WORF requires Java Database Connectivity (JDBC) 2.0, which is the default in DB2 Universal Database Version 8.

Procedure:

The procedures to prepare the WORF environment are as follows:

1. If you are not running DB2 Universal Database Version 8, select JDBC 2.0. Run *C:\SQLLIB\java12\usejdbc2.bat*, assuming that you installed DB2 in C:\SQLLIB\ in a Windows environment.
2. Install the following Internet software:
   - From Apache:
     - Apache Jakarta Tomcat Version 4.0.3 or later binary from http://jakarta.apache.org/site/binindex.html. (Apache Jakarta Tomcat Version 4 standard comes with the appropriate Xerces parser. For earlier versions you must add the Xerces parser to your CLASSPATH to use it as the XML parser.)
     - 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
     - j2ee.jar, version 1.3 or later.
     - qname.jar
   - *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.

Related concepts:
- "Using Web services provider with iSeries" on page 28

Related tasks:
- "Defining the web.xml and group.properties files in the iSeries platform" on page 62
- "Installing and deploying the WORF examples in iSeries" on page 54
- "Installing the Web services provider software requirements on iSeries" on page 37

Installing and deploying the WORF examples in iSeries

Procedure:

To install the WORF examples, do the following tasks:

1. Make sure that the worf.jar file is in

   ```
   /QIBM/UserData/WebASAEs4/worf/lib/app
   ```

   where **WebASAEs4** is the version of WebSphere, and **worf** is the name of the WebSphere instance.
2. If file *runtime.zip* is not in directory /QIBM/UserData/java400/ext, execute the following commands:
   a. >qsh
   b.
   ```
    >ln -s  /QIBM/ProdData/OS400/Java400/ext/runtime.zip
                /QIBM/UserData/Java400/ext/runtime.zip
   ```
3. Copy the *services.war* into your *tomcat\webapps* directory.

   If you already have a *services.war* file installed, then perform the following tasks:
   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 in the "services" web application will be lost with this action, so make sure this is acceptable.
   c. Restart Apache Jakarta Tomcat.
4. 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)
   ```
5. Invoke the examples in the sample application by accessing the test page at http://<system>:<port>/services
   - Invoke a sample with no parameters: http://<system>:<port>/services/travel/ZipCodes.dadx/findAll
   - Invoke a sample with parameters:

     ```
     http://<system>:<port>/services/
         travel/ZipCodes.dadx/findCityByZipCode?zipcode=55901
     ```
6. Verify that your database settings are correct, especially user ID and password, in group.properties. If you do not use a value for user ID, the Web services code runs under QEJBSVR. Therefore, authorize this profile to any database objects that you want to access. Try the verification.dadx on your system (the dynamic test page and the WSDL).

Related concepts:
- "Introduction to using DB2 as a Web services provider – WORF" on page 25

Related tasks:
- "Installing the Web services provider software requirements for Apache Jakarta Tomcat on iSeries" on page 53
- "Installing the Web services provider software requirements" on page 36

Administering and troubleshooting the Web services provider

| This chapter describes some performance suggestions and a troubleshooting guide
| for the Web services provider.

Using connection pooling to improve performance

Each time a resource attempts to access a database, it must connect to that database. A database connection requires resources to create the connection, maintain it, and then release it when it is no longer required. The database resources required for a Web-based application can be high because Web users connect and disconnect more frequently. You can use IBM WebSphere Application Server to help create and maintain a pool of database connections. These database connections can be shared by applications on an application server to address the resource problems. Connection pooling spreads the connection overhead across several user requests, thereby conserving resources for future requests and improving performance. You can configure a pool for each unique data source. You can read more about connection pooling in Chapter 10 of the WebSphere handbook.

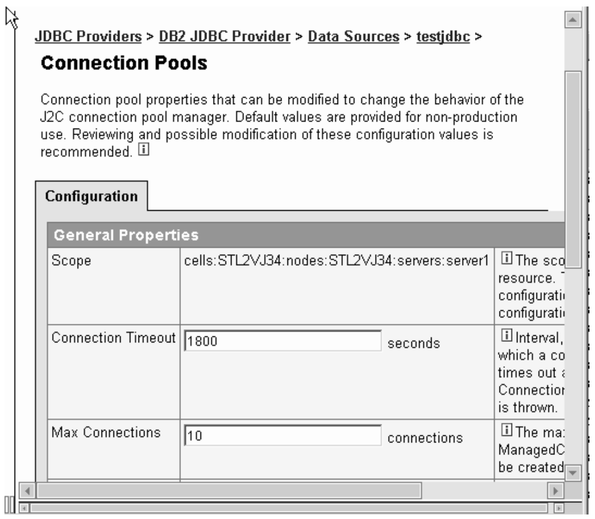
Prerequisites:

1. Create the JDBC provider if one does not exist that you want to use.
2. Create a data source.

Procedure:

Installed applications use JDBC providers to access data from databases. To adjust some of the connection pooling parameters for a particular data source within a JDBC provider from WebSphere Application Server, Version 5, perform the following steps:

1. Configure the data source parameters.
2. Update the connection pooling information, as in Figure 10. WebSphere Application Server provides a Java Naming service (JNDI) to facilitate the connection to DB2 Universal Database. The pool is shared by all applications connecting to the same data source.



JDBC Providers > DB2 JDBC Provider > Data Sources > testjdbc >

Connection Pools

Connection pool properties that can be modified to change the behavior of the J2C connection pool manager. Default values are provided for non-production use. Reviewing and possible modification of these configuration values is recommended.

Configuration

General Properties		
Scope	cells:STL2VJ34:nodes:STL2VJ34:servers:server1	The sco resource. configurati configurati
Connection Timeout	1800 seconds	Interval, which a co times out Connectior is thrown.
Max Connections	10 connections	The ma ManagedC be created

Figure 10. Adjusting connection pooling parameters

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.websphere.naming.WsnInitialContextFactory
datasourceJNDI=jdbc/sampleDataSource
```

4. Restart the Web application if you have made any changes to the group.properties file so they will take effect.

Related tasks:

- "Defining the web.xml and group.properties files" on page 59

Troubleshooting Web services

Table 5 describes problems that can occur when you use WORF on WebSphere Application Server 5.1. The table provides recommended solutions.

Table 5. 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.
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.
Error 400: Unable to get DAD;unable to get Input Stream for: xxxxxx	There is no access to the specified XML Extender DAD file (for example, the DAD that is specified in the *.nst file)
Error 400: database connection error	• Database is not started. • The database objects that are referenced in the DADX file do not exist. • The JDBC driver is not found.
Error 400: unable to get input stream for /groups/xxx/yyy.dadx	The DADX file is not in the group folder, or it is not accessible.
Error 404: File not found: aaaa/abc.dadx	The servlet mapping 'aaaa' does not exist in the web.xml file.
blank page results	If you are using a version of WebSphere Application Server that is earlier than Version 5.0.2, you might be missing `jaas.jar`. Open SystemErr.log in the server directory to determine if other JAR files are missing.

To obtain information about runtime events and diagnostics from the Web service provider to troubleshoot your Web service after it is deployed, you can use the trace facility of the Web application server on which your application runs. The trace information that you receive from the Web application server includes messages and event activity. Even if the tracing is not enabled, errors are captured in the application server error logs. To learn more about how to trace your Web service provider events, see Web services provider tracing

Related concepts:
- "Web services provider tracing" on page 137

Related tasks:
- "Generating deployment descriptors" on page 133

Related reference:
- "Web services provider software requirements for UNIX and Windows" on page 34

Developing applications that use the Web services provider

The following sections describe the overview and details of using Web services provider.

Defining a group of Web services

Groups are containers for Web services that share common configuration options. Configuration options can be the following:
- database configuration
- namespace setup
- message encoding setup

The *groups* directory contains the resources for all DADX Web service groups. The WORF Web application creates this directory during the application configuration. This directory is in the *WEB-INF\classes\groups\* subdirectory of the Web application's base directory.

DADX files contain a description of the Web services. WORF contains 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 the Java class loader can load your DADX files.

Within the *groups* directory, WORF stores each group of DADX Web services in a directory with the same name as its servlet instance. The application server looks for the right servlet instance to call by the given URL which is based on the web.xml file.

Another resource that is used by WORF is the group.imports file. This is an optional resource that helps the various Web service consumers or tools that use the generated WSDL to find the schemas that are used. If the group.imports file exists, then the WSDL generates the imports elements based on the content of the group.imports file and the scope of the element. If no group.imports file exists, then no import elements are generated for WSDLs for non-dynamic query services. For dynamic query services, the WSDL contains some data types that are in db2WebRowSet.xsd. With no group.imports to define a location of db2WebRowSet.xsd, WORF assumes that this schema file is in the default location, such as in the following example:

```
http://<server>:<port>/<contextRoot>/db2WebRowSet.xsd
```

In the examples relating to the DB2® Web services, the WORF application stores the DADX files in the *WEB-INF\classes\groups\dxx_sample\* directory, the *WEB-INF\classes\groups\dxx_sales_db\* directory, and the *WEB-INF\classes\groups\dxx_travel\* directory.

Related tasks:

- "Preparing and creating the Web archive file" on page 136
- "Customizing the group.properties file" on page 64
- "Generating deployment descriptors" on page 133

Related reference:

- "Error checking by the DADX environment checker" on page 240

Defining the web.xml and group.properties files

Procedure:

To define a new group of DADX Web services, complete the following steps. (You can also create these files by using WebSphere Studio version 5. For more information see the WebSphere Studio Information Center (http://publib.boulder.ibm.com/infocenter/wsphelp/index.jsp). If you are migrating to a new version of WORF, make sure that the web.xml and group.properties files contain the values that you expect for your environment.

1. Choose a group name for the DADX group that reflects your application. These instructions use the name `myapp_group`.
2. Edit the *web.xml* file in the *WEB-INF* directory, to define the group name. If you want to use WebSphere Version 5 data sources (WebSphere Studio or WebSphere Application Server), that conform to Java 2 Enterprise Edition Version 1.3, make sure that you change the web-app_2.2.dtd to web-app_2.3.dtd in the web.xml file.

   You can have multiple group names in the same *web.xml* file. The following figure shows an example of the *web.xml* file. The servlet-mapping element is 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 id=InitParam_1076524994485>
          <param-name>faultListener</param-name>
          <param-value>
             org.apache.soap.server.DOMFaultListener
          </param-value>
        </init-param>
        <init-param id=InitParam_1076524994488>
          <param-name>soap-engine</param-name>
          <param-value>apache-soap</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 11. web.xml

The <servlet> section defines a new servlet instance for the group. At least one <servlet> element must exist for each group, but a group can have multiple <servlet-mapping> elements. See the Java servlet specification at http://java.sun.com/products/servlet/. In this example, the <servlet-name> element defines a group named `myapp_group`.

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

<servlet-name>
This is a child tag in the <servlet> section and in the <servlet-mapping> section and defines the name of the group. The servlet name must be a valid directory name under the *groups* directory. You use this name to store the DADX resources for this group of Web services. For example: `myapp_group` is defined in both the <servlet> and <servlet-mapping> elements.

<servlet-mapping>
You must have at least one <servlet-mapping> section to introduce a mapping between a URL and the group. The child tag <servlet-name>, defines the group name and must be the same as the directory name for the group, which also means that the <servlet-name> must be the same as in the <servlet> group. The <servlet-name> tag is the link between the <servlet> and the <servlet-mapping> tag.

<url-pattern>
The uniform resource locator (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/*. The servlet name in this example is myapp_group.

<init-param>
You can update the name of the SOAP engine that you want to use. The parameter name to specify the soap engine is <soap-engine>. If you want to use Apache SOAP, then the parameter value is *apache-soap*. If you want to use Apache Axis, then the parameter value is *apache-axis*. If you do not specify a <soap-engine> parameter, the default soap engine is *apache-soap*.

Note: The default encoding for the *web.xml* file is UTF-8. You update the file in UTF-8 when on OS/390 or z/OS platforms, by sending the *web.xml* file to a UNIX or Windows system. You can send the file by using the File Transfer Protocol (FTP) binary transfer. Then update the file, and return the file to the original system.

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 subdirectory eventually contains the resources for this group.
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=
For OS/390 and z/OS, use the following database driver and URL:
dbDriver=COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
dbURL=jdbc:db2os390:
```

Figure 12. group.properties example

Related concepts:
- “Web services description language” on page 115
- “Dynamic database queries that use the Web services provider” on page 90

Related tasks:
- “Customizing the group.properties file” on page 64

Related reference:
- “Error checking by the DADX environment checker” on page 240
- “Checking errors in the web.xml file” on page 241

Defining the web.xml and group.properties files in the iSeries platform

Procedure:

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. These instructions use the name `myapp_group`.
2.
   - For iSeries, 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. If you want to use WebSphere Version 5 data sources (WebSphere Studio or WebSphere Application Server), that conform to Java 2 Enterprise Edition Version 1.3, make sure that you change the web-app_2.2.dtd to web-app_2.3.dtd in the web.xml file.

     The sample application includes a servlet that applies a style sheet to the generated Extensible Markup Language (XML). The servlet class and source are in the following directory:

     ```
      /QIBM/UserData/WebASAEs4/worf/
     installedApps/servicesApp.ear/
     services.war/WEB-INF/classes
     ```

     The sample file contains the following setting: serveServletsByClassnameEnabled="true". Invoke the servlet by executing the following file:

     ```
     /QIBM/UserData/WebASAEs4/worf/
     installedApps/servicesApp.ear/
     services.war/WEB-INF/ibm-web-ext.xmi
     ```

     You can recompile the servlet inside qshell by executing the following compile statement:

     ```
     javac -J-Djava.ext.dirs=/qibm/proddata/webasaes4/lib
            -d . SampleXSLTServlet.java
     ```

     Invoke the servlet from the Internet by executing the following file:

     ```
     http://<system>:<port>/services/servlet/
     SampleXSLTServlet?XML=
     http://<system>:<port>/services/travel/ZipCodes.dadx/
        findAll&XSL=
     file:///home/zipcodes.xsl
     ```

     The above example assumes that the *zipcodes.xsl* is in the /home directory. You can locate the file anywhere. You can use this servlet example for any combination of XML Web services and Extensible Stylesheet Language (XSL) style sheets.

     The following figure shows an example of the *web.xml* file. The servlet-mapping element is 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 13. web.xml

The <servlet> section defines a new servlet instance for the group. At least one <servlet> element must exist for each group, but a group can have multiple <servlet-mapping> elements. See the Java servlet specification at http://java.sun.com/products/servlet/. In this example, the <servlet-name> element defines a group named `myapp_group`.

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

<servlet-name>
: This is a child tag in the <servlet> section and in the <servlet-mapping> section and defines the name of the group. The servlet name must be a valid directory name under the *groups* directory. You use this name to store the DADX resources for this group of Web services. For example: `myapp_group` is defined in both the <servlet> and <servlet-mapping> elements.

<servlet-mapping>
: You must have at least one <servlet-mapping> section to introduce a mapping between a URL and the group. The child tag <servlet-name>, defines the group name and must be the same as the directory name for the group, which also means that the <servlet-name> must be the same as in the <servlet> group. The <servlet-name> tag is the link between the <servlet> and the <servlet-mapping> tag.

<url-pattern>
: The uniform resource locator (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/*`. The servlet name in this example is `myapp_group`.

<init-param>
You can update the name of the SOAP engine that you want to use. The parameter name to specify the soap engine is <soap-engine>. If you want to use Apache SOAP, then the parameter value is *apache-soap*. If you want to use Apache Axis, then the parameter value is *apache-axis*. If you do not specify a <soap-engine> parameter, the default soap engine is *apache-axis*.

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 subdirectory contains the resources for this group.
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:*local/SAMPLE
userID=
password=
namespaceTable=myapp.nst
autoReload=true
reloadIntervalSeconds=5
```

In the example in Figure 14 on page 64, the use of **SAMPLE** in the dbURL parameter is the database or collection that you want to use.

Figure 14. group.properties example

Related tasks:
- "Customizing the group.properties file" on page 64

Related reference:
- "Error checking by the DADX environment checker" on page 240

Customizing the group.properties file

The *group.properties* file is a standard Java properties file. You must define the group.properties with at least one of the following parameters:

When using group.properties for connection pooling
Define group.properties using parameter initialContextFactory with datasourceJNDI

When using group.properties for regular JDBC database connections
Define group.properties using parameter dbURL with dbDriver

If you define both types of connections, the application tries the DataSource first. If WORF cannot obtain the DataSource, then it tries the JDBC. The complete set of properties is listed below.

Procedure:

To modify the group.properties file, use the following definitions for your environment:

Database configuration parameters

initialContextFactory
: This parameter is used with `datasourceJNDI`, and is required for WebSphere connection pooling. The parameter specifies the Java class name of the JNDI initial context factory that is used to locate the DataSource for the database. This property, along with the datasourceJNDI property, enables connection pooling.

datasourceJNDI
: This parameter is used with `initialContextFactory`, and is required for WebSphere connection pooling. The parameter specifies the JNDI name of the DataSource for the database. When you use this with `initialContextFactory`, it defines a DataSource for the database connection. This property enables connection pooling. You must define either the DataSource or the Java Database Connectivity (JDBC) connection.

dbDriver
: This parameter specifies the Java class name of the Java Database Connectivity (JDBC) driver for connecting to the database.

dbURL
: This parameter is used with dbDriver, and specifies the JDBC uniform resource locator (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. This specifies the user ID for the database.

password
: Optional, but used with user ID. This specifies the password for the database. There are algorithms that are available to help you encode and decode your password.

enableXmlClob
: Optional. This specifies whether retrieveXML operations will use the CLOB-based XML Extender stored procedures. The default value is *true*. This parameter is available only for backward compatibility. For OS/390 and z/OS platforms, either do not define this property, or always set the value to *true*.

Web Service configuration parameters

groupNamespaceUri
: Optional. This parameter defines the target namespace that is used in the generated Web service description language (WSDL) and Extensible Markup Language (XML) schema files (XSD). The target namespace is for Web services in this group.

useDocumentStyle
: Optional. The default value is *false*, which means that the Web services at runtime use RPC encoding. If you set this value to true, then the Web services at runtime use document style and literal encoding. The IBM DB2 Information Integrator Web services provider contains samples that are set to use RPC style. For new applications, you should use document style for maximum interoperability.

namespaceTable
Optional. This specifies the resource name of the namespace table. It references a Namespace Table (NST) resource that defines the mapping from DB2 XML Extender DTDIDs to XML Schema (XSD) namespaces and locations. See Figure 24 on page 83 for an example of an NST file.

Runtime configuration parameters:

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

reloadIntervalSeconds
Optional, but used with `autoReload`. This controls resource loading and caching. It 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.

The options *autoReload* and *reloadIntervalSeconds* control resource loading and caching. If autoReload is absent or false, then there is no resource reloading, and the application ignores reloadIntervalSeconds. If autoReload is true, then, when WORF accesses a resource (such as one of the following files: DAD, DADX, document type definition (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 set `reloadIntervalSeconds` to zero. If the Web services are in production, set `autoReload` to false, or set `reloadIntervalSeconds` to a large value to avoid impacting server performance.

Related concepts:
- "Web services description language" on page 115
- "Defining a group of Web services" on page 58

Related tasks:
- "Defining the web.xml and group.properties files in the iSeries platform" on page 62
- "Defining the web.xml and group.properties files" on page 59

The DADX file

This section describes the properties, syntax, and operation of the DADX file.

Defining the Web service with the document access definition extension file

| The document access definition extension (DADX) file specifies a Web service. It
| does this by using SQL statements, a list of parameters, and optionally, document
| access definition (DAD) file references that define a set of operations. You can
| define a set of dynamic Web service operations with a DADX file that contains
| only the dynamic query service tag (<DQS/>). Operations are similar to methods
| that you can invoke. You can define the operations in a DADX Web service by the
| following operation types:
| • SQL Operations (non-dynamic)

<query>
Queries the database by using a select operation

<update>
Performs an update, insert or delete operation on the database

<call> Calls stored procedures

- SQL Operations (dynamic query services)

<getTables>
Retrieves a description of available tables.

<getColumns>
Retrieves a description of columns.

<executeQuery>
Issues a single SQL statement.

<executeUpdate>
Issues a single INSERT, UPDATE, DELETE.

<executeCall>
Calls a single stored procedure.

<execute>
Issues a single SQL statement.

- Extensible Markup Language (XML) collection operations (requires DB2® XML Extender)

<retrieveXML>
Generates XML documents

<storeXML>
Stores XML documents

Related concepts:
- "Web service provider operations used with DADX files" on page 31

Related reference:
- "Syntax of the DADX file" on page 67

Syntax of the DADX file

The DADX file is an Extensible Markup Language (XML) document. "DADX syntax definitions" on page 68 and Figure 15 on page 68 describe the elements of the DADX. The DADX schema is in XML schema for the DADX file. The numbers next to the nodes and elements in "DADX syntax definitions" on page 68 identify the child groupings. The numbering scheme expresses the XML document hierarchy. For example, when the identifiers change from 1.3 (result_set_metadata) to 1.3.1 (column), this means that the column is a child of result_set_metadata. A change from 1.1 (documentation) to 1.2 (implements) means that these elements are siblings.

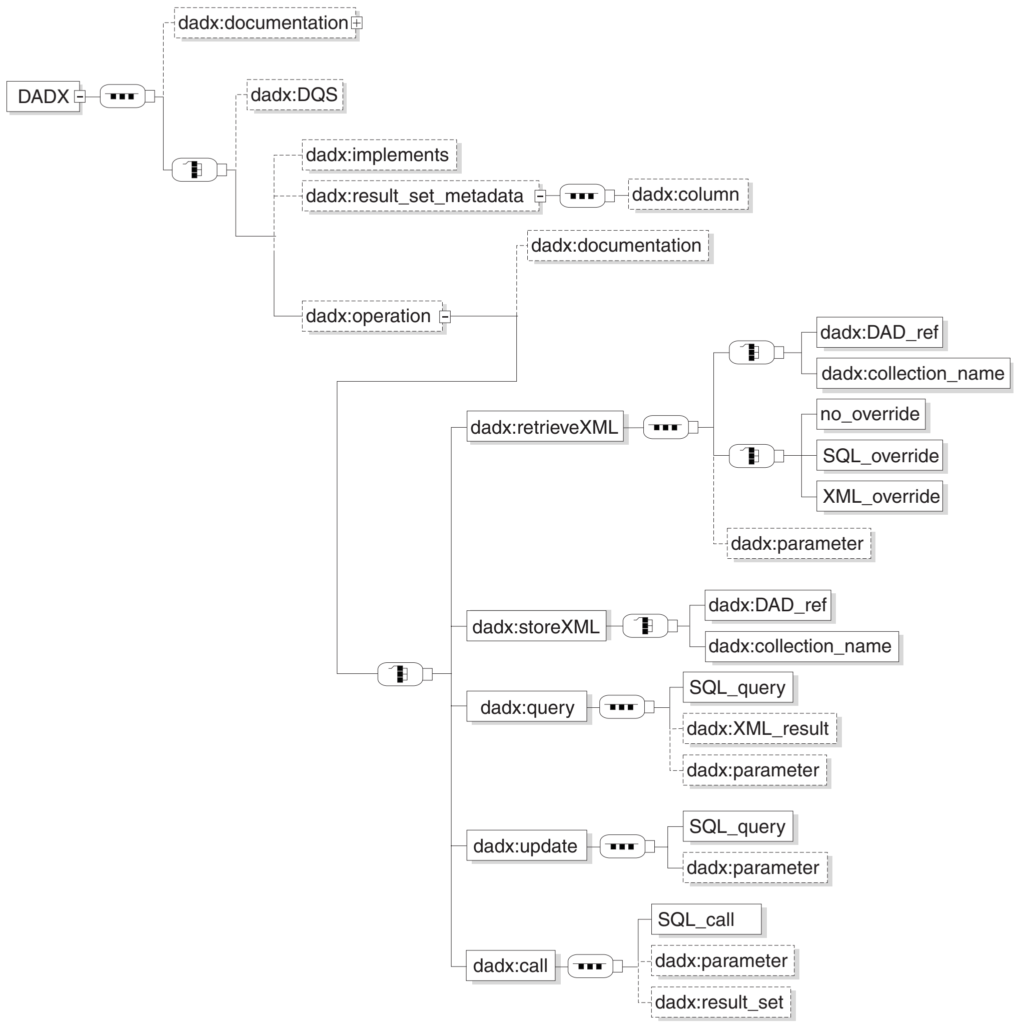


Figure 15. DADX syntax

0. Root element: <DADX>

Attributes:

xmlns:dadx
The namespace of the DADX.

xmlns:xsd
The namespace of the Extensible Markup Language (XML) Schema specification

Children:

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

1. DADX functions that specify non-dynamic operations

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

1.3 <result_set_metadata>

Stored procedures can return one or more result sets. You can include them in the output message. Metadata for a stored procedure result set must be defined explicitly in the non-dynamic DADX using the <result_set_metadata> element. At run-time, you obtain the metadata of the result set. The metadata must match the definition contained in the DADX file.

Note: You can only invoke stored procedures that have result sets with fixed metadata.

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. This specifies the name of the column.

type Required if you do not specify **element**. It specifies the type of column.

element
Required if you do not specify **type**. It specifies the element of column.

as Optional. This 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 performs. Web services can 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 <dadx:documentation>
  Specifies a comment or statement about the purpose and content of the operation. You can use XHTML tags.

  1.4.2 <retrieveXML>

  This element specifies to generate zero or one XML documents from a set of relational tables when using the XML collection access method. Depending on whether you specify a DAD file or an XML collection name, the operation calls the appropriate XML Extender composition stored procedure.

  Children:

  – Specify which of these stored procedures you want to use. You do this by passing either the name of a DAD file, or the name of the collection by 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 you specify a relative path for the DAD file, then the application assumes that the current working directory is the group directory.

    1.4.2.2 <collection_name>
    The content of this element is the name of the XML collection. You define collections by 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 you do not specify either <SQL_override> or <XML_override>.

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 by using the following element:

1.4.2.6 <parameter>
Required when referencing a parameter in an <SQL_override> or an <XML_override> element. This 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. A parameter must have its contents defined by either an XML Schema element (a complex type) or a simple type.

Attributes:

name The unique name of the parameter.

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>

This element specifies to store (decompose) an XML document in a set of relational tables using the XML collection access method. Depending on whether you specify a DAD file or an XML collection name, the operation calls the appropriate XML Extender decomposition stored procedure. Children:

– Specify which of these stored procedures you want to use. You do this by passing either the name of a DAD file, or the name of the collection by 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 you specify a relative path for the DAD file, the application assumes that the current working directory is the group directory.

1.4.3.2 <collection_name>

The content of this element is the name of an XML collection. You define collections by using the XML Extender administration interfaces, as described in *DB2 XML Extender Administration and Programming*.

1.4.4 <query>

Specifies a query operation. You define the operation by using 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 document type definition (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. This defines a named column that contains XML documents. The XML Schema element of its root must define the document type.

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

Attributes:

name The unique name of the parameter.

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>
This specifies an SQL INSERT, UPDATE, or DELETE statement.

1.4.5.2 <parameter>
Required when referencing a parameter in the <SQL_update> element. It 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. A parameter must have its contents defined by one of the following: an XML Schema element (a complex type) or a simple type.

Attributes:

name The unique name of the parameter.

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

Attributes:

name The unique name of the parameter.

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>

This 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. It 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 metadata definition in the DADX file. The identifier must refer to the name of an element.

2. <DQS>
Dynamic query services.

Related concepts:

- “Defining the Web service with the document access definition extension file” on page 66
- “Definition of a DADX file” on page 29

Related reference:

- “A simple DADX file” on page 75
- Appendix C, “XML schema for the DADX file,” on page 247

A simple DADX file

The following example is a simple DADX file that contains one operation with an SQL query. This DADX file is for non-dynamic queries. See Configuring and running dynamic database queries as part of Web services provider for an example of a DADX file used to enable dynamic query services.

Figure 16 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"
  >
  <documentation>
    Simple DADX example that accesses the SAMPLE database.
  </documentation>
  <operation name="listDepartments">
    <documentation>
      Lists the departments.
    </documentation>
    <query>
      <SQL_query>SELECT * FROM DEPARTMENT</SQL_query>
    </query>
  </operation>
</DADX>
```

Figure 16. 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.

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66
- "Definition of a DADX file" on page 29
- "Dynamic database queries that use the Web services provider" on page 90

Related tasks:

- "Configuring and running dynamic database queries as part of Web services provider" on page 91

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 by 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 by using the <RetrieveXML> element. The <DAD_ref> element specifies the name of a DAD file.
The Web service generated from this DADX file calls the dxxGenXML stored

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

Figure 17. DADX file that generates an XML document

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.

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66
- "Testing Web services applications – a scenario" on page 109

Related reference:

- "DADX operation examples" on page 80

Using overrides in the DADX file

The DADX file can override Extensible Markup Language (XML) values and SQL statements in the DAD file by 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, shown in Figure 17 on page 76.

The following example uses an SQL override statement.
Although you can override the SQL statement, the new SQL statement must

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <documentation >
    Provides queries for part order information at myco.com.
    See <xhtml:a href="../documentation/PartOrders.html" target="_top">
         PartOrders.html</xhtml:a> for more information.
  </documentation>
  <operation name="findAll">
    <documentation >
      Returns all the orders with their complete details.
    </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 18. Example of a DADX file that generates an XML document with an SQL override

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, you have to override the RDB nodes by using the <XML_override> element. RDB node elements define DB2 Universal Database tables, columns, and conditions that are to contain XML data. The example in Figure 19 on page 78 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. You must define the name and type of all parameters 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"
      xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <documentation >
    Provides queries for part order information at myco.com.
    See <xhtml:a href="../documentation/PartOrders.html" target="_top">
         PartOrders.html</xhtml:a> for more information.
  </documentation>
  <operation name="findByExtendedPriceAndShipDate">
    <documentation >
      Returns all the orders with an extended price greater than $50.00
      and a ship date later than 1998-12-01.
    </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 19. Example of a DADX file that generates an XML document with an XML override

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66

Related reference:

- "DADX operation examples" on page 80
- "A simple DADX file" on page 75

Declaring and referencing parameters in the DADX file

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. The Extensible Markup Language (XML) and SQL overrides that you use in the <retrieveXML> and <storeXML> operations can also reference parameters. You declare the parameters by using the <parameter> element. The parameters have simple XML Schema types that correspond to the built-in SQL data types. Table 6 describes the supported types.

Table 6. Supported XML Schema and SQL types

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

Table 6. Supported XML Schema and SQL types (continued)

XML Schema Simple Type	SQL Type
timestamp	TIMESTAMP
long	BIGINT
byte	TINYINT

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, as in the following example:

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

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

The example in Figure 20 on page 79 shows a query operation that retrieves a set of relational data by using an SQL SELECT statement. The statement contains one input parameter by using the parameter syntax.

```
<operation name="findCustomerOrders">
    <documentation>Returns all the orders for a given customer.
    </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 20. Query operation with a parameter

The example in Figure 21 on page 80 shows parameters in an SQL override that are used by a retrieveXML operation:

```
<operation name="findByColorAndMinPrice">
    <documentation>Returns all the orders that have the specified color and
       at least the specified minimum price.
    </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 21. SQL override used by a retrieveXML operation

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

Related concepts:
- "XML schema definitions" on page 119

Related reference:
- "A simple DADX file" on page 75
- "Syntax of the DADX file" on page 67
- Appendix C, "XML schema for the DADX file," on page 247

DADX operation examples

The following samples show DADX files with the various operations.

Example 1: Query operation

This example shows a Query operation, using the default tags. This example does not need 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">
     <documentation>
      mycompany part orders service.
    </documentation>
  <implements namespace="http://www.poia.org/part_orders.wsdl"
      location="http://www.poia.org/part_orders.wsdl"/>
<operation name="findCustomerOrders">
    <documentation>Returns all the orders for a given customer.
    </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 22. DADX with Query operation

A list of parameter elements that are uniquely named within this operation must define the input parameters. If you need more control over the mapping, then you can use a DAD file.

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

The example in Figure 23 on page 82 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 by using the UDF db2xml.varchar. You store the documents by 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">
  <documentation>
       Queries part orders at myco.com.
  </documentation>

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

Figure 23. 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 Figure 23 on page 82 the example specifies that the ORDER column contains fragments of element *dtd1:Order*. The element <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 line, <XML_result name = "ORDER" element = "dtd1:Order"/> 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. Refer to Figure 24 on page 83 as an 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 24. NST file

You must reference this file in the *group.properties* file. See the example in Figure 12 on page 61 to learn more about this file.

Example 2: Update operation

The example in Figure 25 on page 83 shows an operation that updates the electronic mail (e-mail) address of a customer for a given order. The update operation can contain SQL INSERT, DELETE, or UPDATE statements in the <SQL_update> element.

```
  <operation name="updateOrderEmail">
    <documentation>Updates the email address for an order.
    </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 25. Update operation

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. This is to let WORF generate the WSDL and XML schema files (XSD) for this Web service operation. Figure 26 on page 84 shows the definition of a result set metadata 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 26. Definition of a result set metadata referenced two times

You can also call a stored procedure by using the format shown in Figure 27 on page 84.

```
<operation name="callProc1">
    <documentation>Call the Proc1 stored procedure.
    </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 27. DADX with alternate Call operation

Example 4: RetrieveXML operation

The DADX file in Figure 28 on page 85 implements one retrieveXML operation by using the stored procedure dxxGenXMLCLOB.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <documentation>
      mycompany part orders service.
    </documentation>
  <operation name="findByColorAndMinPrice">
    <documentation>Returns all the orders that have the specified color
        and at least the specified minimum price.</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 28. DADX with retrieveXML operation

The operation in Figure 28 on page 85 generates XML documents that are based on the mapping in the *getstart_xcollection.dad* file. The operation specifies an SQL override. The operation 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 in the appendix of *DB2 XML Extender Administration and Programming*.

Example 5: StoreXML operation

This example in Figure 29 on page 86 shows a DADX file that references a DAD by 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">
     <documentation>
      mycompany part orders service.
    </documentation>

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

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

Figure 29. 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.

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66

Related reference:

- "A simple DADX file" on page 75

Converting a document type definition to an XML schema

| XML Extender currently uses document type definition files (DTDs) to define
| document structure, while Web services description language (WSDL) uses XML
| schemas (XSD files). Web services object runtime framework (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. This also enables
| conversion of the DTD to XSD.

Procedure:

Request an XSD file by using the following uniform resource locator (URL) syntax:

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

For example:

```
http://host_name:port/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 document type definition
| identifier (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 might 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')"
```

Related concepts:

- "XML schema definitions" on page 119

Related reference:

- Appendix C, "XML schema for the DADX file," on page 247

WSDL from a DADX file

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.

The Web services description language (WSDL) document is an Extensible Markup Language (XML) vocabulary that is used to describe the interface of business services. You can use it to publish services to a UDDI registry. WSDL allows development tools to programmatically create requester code and provider code for use in binding to a Web service. It also enables preconditioned applications to dynamically bind to a Web service. You can use WSDL to specify the data that are required for requests and responses. WSDL uses XML Schema for precise data definition.

A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either an RPC style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use.

To generate the WSDL, submit the following uniform resource locator (URL). The *localhost* port number, designated here by <*yourWebAppServer*> depends on your own current machine:

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

Web services object runtime framework (WORF) dynamically generates the WSDL document. You can publish this in UDDI or some other Web service directory.

If you use the samples that WORF includes during the installation, you can submit the following URL:

```
http://yourWebAppServer/services/sales/PartOrders.dadx/WSDL
```

Related concepts:

- "WSDL for UDDI registration" on page 88
- "Defining the Web service with the document access definition extension file" on page 66
- "Web services description language" on page 115

Related tasks:

- "Testing the Web service" on page 109

WSDL for UDDI registration

The Universal Description, Discovery, and Integration (UDDI) best practices document explains the use of Web services description language (WSDL) with UDDI registries. This document recommends that you split the WSDL document 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. 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://yourWebAppServer:port/webapp_name/
     group_name/DADX_file.dadx/WSDLservice
  ```
- To generate the reusable part, submit a URL with `WSDLbinding` key words:
  ```
  http://yourWebbAppServer: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 in the following example:

```
http://yourWebAppServer/sales_db/part_orders.dadx/WSDLservice
```

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

```
http://yourWebAppServer/sales_db/part_orders.dadx/WSDLbinding
```

The above example deals with the case in which the service implementer creates a Web service that is unique to a company. One of the usage scenarios that UDDI is designed to handle is one where a standards body or vendor defines a Web service interface tModel. Then, service implementers use the Web service. For example, the airline industry might define a Web service that provides flight schedules, that airlines can implement. UDDI allows users to search for all registered services that implement a given tModel. Then, a travel planning application can 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 that is defined elsewhere. An example of an <implements> element is shown in Figure 30 on page 89.

```
<?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 30. Element <implements>

The following example shows how the <implements> tag is used in a DADX file:

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

<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx"

  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  xmlns:xhtml="http://www.w3.org/1999/xhtml">

<documentation>

  Provides queries for part order information at myco.com.

  This Web Service is compliant with the Part Ordering Industry
   Association standard.

  </documentation>


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

  location="http://www.poia.org/PartOrders.wsdl"/>


<operation name="findAll">

<documentation%gt;

Returns an order with its complete details.

</documentation>

    ...



  </operation>
...
```

Figure 31. Example DADX file using an implements tag

Related concepts:
- "WSDL from a DADX file" on page 87
- "Web services description language" on page 115

Dynamic database queries that use the Web services provider

With dynamic query services you can dynamically build and submit queries at run time that select, insert, update application data, and call stored procedures rather than run queries that are predefined at deployment time.

A Web application can use the Web services interface to access a database and extract information about the tables and columns that are available. Then, the application can query the tables and modify the data in the database through Web services. The Web application can also perform data definition language actions on the database, such as creating tables. By using the dynamic query services of the Web services provider, Web applications can be more flexible.

WORF can generate two styles of Web services description language files (WSDL) from the DADX files that contain a dynamic query service tag (<DQS/>):

- A WSDL file that uses the document-oriented information style
- A WSDL file that uses the procedure-oriented information style (RPC)

The style that is generated is defined on a group level and depends on the existence of `useDocumentStyle=true` in the group.properties file. For more information about the Web services description language information styles, look in the Web services description language specifications on your browser. The WSDL file contains service, port, and definition information. Dynamic query tags in the DADX files do not affect static DADX functions.

Consider using the dynamic query service when you do not know the query search criteria until you run your application.

The dynamic query component of the Web services provider supports Web service operations that are generally defined by the following categories:

Obtain metadata
You can retrieve the tables that exist in a database and the column information for those tables.

Execute DDL
You can issue a CREATE TABLE statement.

Execute DML
You can issue SELECT, INSERT, UPDATE and DELETE statements, and the CALL statement to run stored procedures.

The server administrator controls access to a specific database by defining a group with specific user ID and password settings in the group.properties file. The administrator can also create a separate WORF instance to handle access to a database.

Related concepts:
- “WSDL from a DADX file” on page 87
- “Web services provider features” on page 30
- “Web service provider operations used with DADX files” on page 31
- “Web services description language” on page 115
- “Dynamic query services-example queries” on page 93

Related tasks:
- “Customizing the group.properties file” on page 64

Related reference:
- “Dynamic query service operations in the Web services provider” on page 99

Configuring and running dynamic database queries as part of Web services provider

With dynamic query services, you can build, execute stored procedures and submit database queries at run time that access a previously deployed Web service. You no longer need to define all of your database queries in your Document Access Definition Extension (DADX) file. You can run dynamic queries at the group level, or within the scope of the group directory, based on the information in the group.properties file.

Prerequisites:

- Ensure that a *group.properties* file exists for the group in which you want to run dynamic Web queries.
- The Web application must establish a connection to the target database for each Web service operation that is defined in the Web services description language (WSDL) document.
- You must ensure that the XML schema description file, *db2WebRowSet.xsd* is included in the context root of your Web application, unless you define an import definition in the WSDL. File *db2WebRowSet.xsd* is included in the *dxxworf.zip* file.

Restrictions:

- When your application uses the dynamic query services of the Web services provider, the application cannot use cursors or perform any operation that assumes a state on the server. You must obtain your results in a single query.
- The XML tag (<DQS/>) that identifies a dynamic query service operation cannot coexist with any DADX-specific Web service definitions within the same file.

Procedure:

To prepare your Web services environment to run dynamic queries on a DB2 Universal Database with Web services provider:

1. Create a DADX file that includes the XML tag <DQS/>.

   This tag enables a group to perform dynamic queries. No other tag is needed in the DADX file.
2. Save the file in the directory of the group for which you will run dynamic queries.
3. Using the WSDL, develop a client for the application. The client must contain at least the following information:
   - A group name
   - The name of the DADX file, such as *mydqs.dadx*
   - A Web service operation, such as `getTables`
4. Modify the client to issue one of the accepted DQS operations, such as the getTables operation.
5. Run the client that issues the `getTables` operation.

   The result of the query is metadata that describes the rows and columns of the table, and the data that is contained in the tables. The SQL statements run in autocommit mode. The client can also call a dynamic query service in other groups. The only information that needs to change is the endpoint URL. However, clients are only compatible for either an RPC style WSDL or a document style WSDL. You cannot use a dynamic query services client that is defined by an RPC style WSDL for a group that uses a document style WSDL.

Related concepts:

- "Web services description language" on page 115
- "Dynamic query services-example queries" on page 93
- "Dynamic database queries that use the Web services provider" on page 90
- "Defining a group of Web services" on page 58

Related reference:

- "A simple DADX file" on page 75

Dynamic query services-example queries

Example: the DADX file

In the following example, the DADX file is named *mydqs.dadx*. The file mydqs.dadx is in the directory of the group for which you will execute dynamic queries.

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx">
    <documentation>
        This is optional documentation about DQS
    </documentation>
      <DQS/>
</DADX>
```

Example: Running the dynamic query from a browser

The following example is a simple dynamic query that you can run from a browser. You can also include this statement in an application. The required information for a dynamic query in Web services provider is in **bold** print. The Web service operation in this example is `executeQuery`. The parameter associated with the operation is `queryInput`. The statement fetches all rows of column `lastname` from table `employee`:

```
http://localhost:9080/services/<group_name>
  /somefile.dadx/executeQuery?queryInputParameter
     =select%20lastname%20from%20employee
```

The example issues a GET binding request rather than a complete SOAP envelope. See Accessing the Web service with GET, POST, and SOAP bindings for more information on the GET, POST, and SOAP bindings. The following output is from the executeQuery operation and it is defined by the db2WebRowSet schema definition:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:executeQueryResponse
    xmlns:ns1="http://schemas.ibm.com/db2/dqs">
  <queryOutputParameter>
   <db2WebRowSet
       xmlns="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<metadata>
 <column-count>14</column-count>
 <column-definition>
  <column-index>1</column-index>
  <nullable>0</nullable>
  <column-name>EMPNO</column-name>
  <column-precision>6</column-precision>
  <column-scale>0</column-scale>
  <column-type>CHAR</column-type>
  <column-type-name>CHAR</column-type-name>
  <xml-type>string</xml-type>
 </column-definition>
 <column-definition>
  <column-index>2</column-index>
  <nullable>0</nullable>
  <column-name>FIRSTNME</column-name>
  <column-precision>12</column-precision>
  <column-scale>0</column-scale>
```

```
    <column-type>VARCHAR</column-type>
    <column-type-name>VARCHAR</column-type-name>
    <xml-type>string</xml-type>
  </column-definition>
...
  <column-definition>
    <column-index>14</column-index>
    <nullable>1</nullable>
    <column-name>COMM</column-name>
    <column-precision>9</column-precision>
    <column-scale>2</column-scale>
    <column-type>DECIMAL</column-type>
    <column-type-name>DECIMAL</column-type-name>
    <xml-type>decimal</xml-type>
  </column-definition>
...
  <column-definition>
    <column-index>14</column-index>
    <nullable>1</nullable>
    <column-name>COMM</column-name>
    <column-precision>9</column-precision>
    <column-scale>2</column-scale>
    <column-type>DECIMAL</column-type>
    <column-type-name>DECIMAL</column-type-name>
    <xml-type>decimal</xml-type>
  </column-definition>
</metadata>
</data>
</db2WebRowSet>
</queryOutputParameter>
    </ns1:executeQueryResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Example: Importing db2WebRowSet.xsd

When the group contains a dynamic query services DADX, the db2WebRowSet.xsd file must be accessible to Web services consumers. To ensure the location of the db2WebRowSet.xsd file, the group.imports file defines the necessary schema locations. The following is an example of a group.imports file to import db2WebRowSet.xsd. This example assumes that you do not have file db2WebRowSet.xsd in your local groups directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<imports
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import namespace="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
      schemaLocation="http://myServer.myCo.com/schemas/misc/ibm/db2WRS.xsd"/>
</imports>
```

If no group.imports file exists, then WORF generates the default import elements in the WSDL only for the dynamic query services. In this case, WORF assumes that the db2WebRowSets.xsd file is in the following location:

```
http://<server>:<port>/<contextRoot>/db2WebRowSet.xsd
```

Example: getTables

The following is an example of the getTables operation:

```
<?xml version="1.0" encoding="UTF-8"?>
 <SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
```

```
    <ns0:getTables
      xmlns:ns0="http://schemas.ibm.com/db2/dqs">
     <tablesInputParameter>
       <tablesInputData>
        <schemaPattern>MSCHENK</schemaPattern>
        <tableNamePattern>EMPLOYEE</tableNamePattern>
       </tablesInputData>
      </tablesInputParameter>
    </ns0:getTables>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: getColumns

The following is example of the getColumns operation:

```
<?xml version="1.0" encoding="UTF-8"?>
 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Body>
    <ns0:getColumns
      xmlns:ns0="http://schemas.ibm.com/db2/dqs">
      <columnsInputParameter>
        <columnsInputData>
          <schemaPattern>MSCHENK</schemaPattern>
          <tableNamePattern>EMPLOYEE</tableNamePattern>
          <columnNamePattern>EMPNO</columnNamePattern>
        </columnsInputData>
      </columnsInputParameter>
    </ns0:getColumns>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: executeQuery

The following example query fetches all rows from table `employee` and specifies several parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
     <ns0:executeQuery
       xmlns:ns0="http://schemas.ibm.com/db2/dqs">
      <queryInputParameter>
       select * from employee
      </queryInputParameter>
      <extendedInputParameter>
      <properties>
         <loginInfo>
           <userid>userid</userid>
           <password>some_password</password>
         </loginInfo>
         <readOnly>true</readOnly>
         <isolationLevel>READ_UNCOMMITTED</isolationLevel>
         <escapeProcessing>true</escapeProcessing>
         <startAtRow>4</startAtRow>
         <fetchSize>80</fetchSize>
         <maxFieldSize>20</maxFieldSize>
         <maxRows>100</maxRows>
         <queryTimeout>2000</queryTimeout>
      </properties>
      </extendedInputParameter>
    </ns0:executeQuery>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: executeUpdate

The following example shows a dynamic query services update statement:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
   <ns0:executeUpdate
     xmlns:ns0="http://schemas.ibm.com/db2/dqs">
     <queryInputParameter>
        update bo_events set OBJECTEVENTID=&'testestest&'
     </queryInputParameter>
     <extendedInputParameter>
       <properties/>
     </extendedInputParameter>
   </ns0:executeUpdate>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following example shows the response document that is returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:executeUpdateResponse
   xmlns:ns1="http://schemas.ibm.com/db2/dqs">
    <updateOutputParameter xsi:type="xsd:int">
        1
    </updateOutputParameter>
   </ns1:executeUpdateResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Example: executeCall

The example request calls stored procedure `multipleResultSets`:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns0:executeCall
      xmlns:ns0="http://schemas.ibm.com/db2/dqs">
    <callInputParameter>
      <callInputData>
          <spName>
             multipleResultSets
          </spName>
          <parameters>
            <parameter>
              <inParam>
                 <kind>IN</kind>
                 <type>string</type>
                 <value>000130</value>
              </inParam>
            </parameter>
            <parameter>
              <inParam>
                <kind>INOUT</kind>
                <type>string</type>
                <value>000130</value>
              </inParam>
             </parameter>
             <parameter>
               <outParam>
                 <kind>OUT</kind>
                 <type>string</type>
               </outParam>
             </parameter>
          </parameters>
       </callInputData>
    </callInputParameter>
   <extendedInputParameter>
    <properties/>
   </extendedInputParameter>
  </ns0:executeCall>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following example shows the sample output:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:executeCallResponse
     xmlns:ns1="http://schemas.ibm.com/db2/dqs">
  <callOutputParameter>
  <dqs:callOutputData
     xmlns:dqs="http://schemas.ibm.com/db2/dqs/types/soap">
   <dqs:outputResultSequences>
   <db2WebRowSet
     xmlns="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <metadata>
      <column-count>5</column-count>
      <column-definition>
        <column-index>1</column-index>
        <nullable>0</nullable>
        <column-name>DEPTNO</column-name>
        <column-precision>3</column-precision>
        <column-scale>0</column-scale>
        <column-type>CHAR</column-type>
        <column-type-name>CHAR</column-type-name>
        <xml-type>string</xml-type>
      </column-definition>
        ...
      <column-definition>
        <column-index>5</column-index>
        <nullable>1</nullable>
        <column-name>LOCATION</column-name>
        <column-precision>16</column-precision>
        <column-scale>0</column-scale>
        <column-type>CHAR</column-type>
        <column-type-name>CHAR</column-type-name>
        <xml-type>string</xml-type>
 </column-definition>
</metadata>
<data>
```

```
 <row>
  <column>A00</column>
  <column>
     SPIFFY COMPUTER SERVICE DIV.
  </column>
  <column>000010</column>
  <column>A00</column>
  <column xsi:nil="true"/>
 </row>
...
 <row>
   ...
 </row>
</data>
</db2WebRowSet>
</dqs:outputResultSequences>
   <dqs:outputParameterSequences>
    <dqs:callOutputParam>
     <position>2</position>
     <type>string</type>
     <value>xxxxxx</value>
    </dqs:callOutputParam>
    <dqs:callOutputParam>
     <position>3</position>
     <type>string</type>
     <value>This is the value of name3</value>
    </dqs:callOutputParam>
    </dqs:outputParameterSequences>
    </dqs:callOutputData>
   </callOutputParameter>
  </ns1:executeCallResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Example: execute

The following example creates a table with one column:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
  <ns0:execute
    xmlns:ns0="http://schemas.ibm.com/db2/dqs">
     <queryInputParameter>
       create table temptable(in varchar(500))
     </queryInputParameter>
     <extendedInputParameter>
       <properties/>
     </extendedInputParameter>
  </ns0:execute>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following is the output from the execute operation:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
   <ns1:executeResponse
     xmlns:ns1="http://schemas.ibm.com/db2/dqs">
    <executeOutputParameter>
    <dqs:executeOutputData
       xmlns:dqs="http://schemas.ibm.com/db2/dqs/types/soap">
      <resultsPresent>false</resultsPresent>
     <dqs:outputResultSequences>
     </dqs:outputResultSequences>
    </dqs:executeOutputData>
    </executeOutputParameter>
  </ns1:executeResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Related concepts:

- "Dynamic database queries that use the Web services provider" on page 90

Related tasks:

- "Configuring and running dynamic database queries as part of Web services provider" on page 91

Related reference:

- "Dynamic query service operations in the Web services provider" on page 99

Dynamic query service operations in the Web services provider

The following tables describe the dynamic query operations that are supported in the DB2 Web services provider.

Table 7. Operations for metadata retrieval

Web service operation	Description
getTables **tablesInputParameter** input; type = tablesInputData (see Table 11 on page 102) **tablesOutputParameter** output; type = db2WebRowSet	Retrieves a description of the tables in the specified catalog and schema, such as the name of the catalog, the name of the schema, and the name of the table. If you use schema as an input parameter, the Java database connectivity might require case sensitivity for schema.
getColumns **columnsInputParameter** input; type = columnsInputData (see Table 12 on page 103) **columnsOutputParameter** output; type = db2WebRowSet	Retrieves a description of the columns in the specified catalog, schema, and table. If you use schema as an input parameter, the Java database connectivity might require case sensitivity for schema.

Table 8. Operations to run queries and stored procedures

Operations	Description
executeQuery **queryInputParameter** required input; type = string **extendedInputParameter** required input; type = properties (see Table 9) **queryOutputParameter** output; type = db2WebRowSet	Issues a single SQL SELECT statement on the database server and returns a single result set.
executeUpdate **queryInputParameter** required input; type = string **extendedInputParameter** required input; type = properties (see Table 9) **updateOutputParameter** output; type = int	Issues a single INSERT, UPDATE, DELETE statement on the database server and returns a completion code.
executeCall **callInputParameter** input; type = callInputData **extendedInputParameter** input; type = properties (see Table 9) **callOutputParameter** output; type = callOutputData	Calls a single stored procedure on the database server and returns a set of output parameters and a sequence of result sets.
execute **queryInputParameter** required input; type = string **extendedInputParameter** required input; type = properties (see Table 9) **executeOutputParameter** output; type = executeOutputData	Issues a single SQL statement on the database server and returns a completion code and a sequence of result sets.

You can use the optional parameters that are listed in Table 9 with the operations that are listed in Table 8.

Table 9. Input data types for the extended parameters

Properties type	Description
loginInfo • userid • password	The loginInfo includes the user ID that is passed to the database for access control. It also includes the password that is associated with the user ID that is passed to the database for access control. These properties have a type of string. If you specify a user ID, then you must specify a password.
readOnly	Allows the Web application to specify that it will use the database for read-only purposes. This is a binary type and can be either true or false.
escapeProcessing	Allows the Web application to control escape processing on the query string. If escape scanning is enabled (true), the driver performs escape substitution before it sends the SQL to the database. This is a binary type and can be either true or false. The default value is true.

Table 9. Input data types for the extended parameters (continued)

Properties type	Description
fetchSize	Specifies the number of rows to be fetched back to the Web application on any given fetch operation. This is type integer. The default value is 0.
maxFieldSize	Sets the limit for the maximum number of bytes in a column to the specified number of bytes. The value is the maximum number of bytes that can be returned for any column value. The is type integer.
maxRows	Specifies the maximum number of rows to fetch back to the Web application. This is type integer. If the maxRows parameter is not specified, then a maximum of 1000 rows can be returned.
startAtRow	Allows the Web application to skip a specified number of rows in the result set. This is type integer.
queryTimeout	Allows the Web application to specify a timeout value for the query. Sets the number of seconds that the driver waits for a statement object to run to the given number of seconds. If the limit is exceeded, an exception occurs. A value of 0 seconds indicates that the driver can wait an unlimited number of seconds.
isolationLevel	Allows the Web application to control the isolation level of the query. • READ_UNCOMMITTED • READ_COMMITTED • REPEATABLE_READ • SERIALIZABLE • NONE

Table 10. Input data types for the callInputParameter

callInputData type	Description
spName type: string	The name of the stored procedure to invoke. This parameter is mandatory.
schema type: string	The schema of the stored procedure. This parameter is optional. If the parameter is not supplied, the value is the current schema.

Table 10. Input data types for the callInputParameter (continued)

callInputData type	Description
parameters type: sequence of parameters, each one consisting of either an inParam or an outParam **inParam** type defined as: • **kind**: either ′IN′ or ′INOUT′ • **type**: the type of the parameter (such as int, or string) • **value**: the value of the parameter **outParam** type defined as: • **kind**: either ′IN′ or ′INOUT′ • **type**: the type of the parameter	Stored procedures can have three kinds of parameters: IN, OUT, and INOUT. This parameter type is an extensible type. It allows any number of any combination of the inParam and outParam types. The Web application must know if the stored procedure that it plans to invoke needs any parameters. If it needs parameters, it needs to know how many parameters, and their type. If the stored procedure takes one of the unsupported data types as a stored procedure parameter, then this stored procedure cannot be executed through WORF. WORF accepts several XML types for the stored procedure parameters. The parameters correspond to the built-in SQL data types. Table 6 on page 78 describes the supported types. An input parameter can be set to NULL by using one of the following values: **absent** The <value/> tag for the input parameter is not provided. **nil = true** The tag is marked with the attribute nil, which is set to true, such as <value xsi:nil="true"/> The order of the input parameter must be the same as the order expected by the stored procedure.

Table 11. Input data types for the tablesInputData type

tablesInputData type	Description
catalogPattern type = ″string″ **schemaPattern** type = ″string″ **tableNamePattern** type = ″string″	Each of the pattern values is optional. If the value is not specified, the value defaults to the blank value. The description and behavior of each is specified in JDBC. Use the getTables Web service operation to return the list of tables that are satisfy the catalogPattern, schemaPattern, and tableNamePattern that are specified.

Example (note that such things as the namespace definitions are not shown here for simplicity):

```
<tablesInputData>
 <catalogPattern></catalogPattern>
 <schemaPattern>userSchema
 </schemaPattern>
 <tableNamePattern>EMPLOYEE
 </tableNamePattern>
</tablesInputData>
```

Table 12. Input data types for columnsInputData types

columnsInputData type	Description
catalogPattern type = ″string″ **schemaPattern** type = ″string″ **tableNamePattern** type = ″string″ **columnNamePattern** type = ″string″	Each of the pattern values is optional. If the value is not specified, the value defaults to the blank value. The description and behavior of each is specified in JDBC. Use the getColumns Web service operation to receive a list of columns that satisfy the catalog string pattern, schemaPattern, table name, and columnNamePattern that is specified.

Example (note that such things as the namespace definitions are not shown here for simplicity):

```
<columnsInputData>
 <catalogPattern></catalogPattern>
 <schemaPattern>userSchema
 </schemaPattern>
 <tableNamePattern>EMPLOYEE
 </tableNamePattern >
 <columnNamePattern>LASTNAME
 </columnNamePattern>
</columnsInputData>
```

Table 13. Output data types for the callOutputData types

callOutputData type

outputResultSequences
: contains a sequence of all result sets returned by the stored procedure as type db2WebRowSet

outputParameterSequences:
: contains a sequence of callOutputParam (parameters that were returned from the stored procedure that can be either kind=INOUT or kind=OUT)

callOutputParam
: returned Parameter: contains

- <position>

  type: int - the position of the parameter in the stored procedure parameter list
- <type>

  type: string - the XML data type (see callInputData for type information)
- <value>

  type: any - the value of the parameter

If an output parameter is NULL the absent method is used.

The result contains

```
<value xsi:nil="true"/>
```

Table 13. Output data types for the callOutputData types (continued)

callOutputData type
Example (note that such things as the namespace definitions are not shown here for simplicity):

```
<callOutputParameter>
   <dqs:callOutputData
      xmlns:dqs="http://schemas.ibm.com/db2/dqs/types/soap">
      <dqs:outputResultSequences>
         <db2WebRowSet
           xmlns="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <metadata>
             ....
         </db2WebRowSet>
     </dqs:outputResultSequences>
     <dqs:outputParameterSequences>
            <dqs:callOutputParam>
                <position>1</position>
                <type>short</type>
                <value>123</value>
            </dqs:callOutputParam>
            <dqs:callOutputParam>
                <position>2</position>
                <type>int</type>
                <value xsi:nil="true" />
            </dqs:callOutputParam>
         </dqs:outputParameterSequences>
      </dqs:callOutputData>
</callOutputParameter>
```

Table 14. Output data types for the executeOutputData types

executeOutputData type	**Description**
resultsPresent type = ″boolean″ **outputResultSequences** 0 or more occurrences of db2WebRowSet	If the **execute** Web service operation is invoked with a query string that returns result sets, the boolean indicates that this, and outputResultSequences will each contain one of those result sets.

Example (note that such things as the namespace definitions are not shown here for simplicity):

```
<executeOutputParameter>
   <dqs:executeOutputData
      xmlns:dqs="http://schemas.ibm.com/db2/dqs/types/soap">
      <resultsPresent>true</resultsPresent>
      <dqs:outputResultSequences>
         <db2WebRowSet
           xmlns="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <metadata>
             ....
         </db2WebRowSet>
      </dqs:outputResultSequences>
   </dqs:executeOutputData>
</executeOutputParameter>
```

Related concepts:

- "Dynamic query services-example queries" on page 93
- "Dynamic database queries that use the Web services provider" on page 90

Related tasks:

- "Configuring and running dynamic database queries as part of Web services provider" on page 91

Related reference:

- "db2WebRowSet" on page 105

db2WebRowSet

The dynamic query service type, db2WebRowSet, describes a generic way for generating an XML document from an SQL result set. The schema document, `db2WebRowSet.xsd` does not contain any metadata information about a particular result set. It contains generic metadata information about result set metadata. The actual result set metadata and the result set data is in the XML instance document. An instance document contains a metadata section and a data section.

metadata section

Contains metadata information about all of the columns that are in the result.

The first tag is a column count tag. It contains the number of columns in the result set. Then there is a column definition tag for every column. The column definition contains the following metadata information:

Table 15. Column definition metadata

Tag name	Description
<column-index>	The position of the column in the result set, starting with 1.
<nullable>	If the column can be NULL, then the value is 1. If the column cannot be NULL, then the value is 0.
<column-name>	The name of the column.
<column-precision>	The description of this tag depends on the SQL data type. For example, if the SQL data type is a character, then the column-precision is length. If the SQL data type is a decimal, then the column- precision is precision.
<column-scale>	The column-scale is a decimal data type.
<column-type>	The column-type corresponds to the Java database connectivity type, such as BINARY, VARBINARY, CHAR, and VARCHAR.
<column-type-name>	The DB2 Universal Database data type name, such as CHAR FOR BIT DATA, VARCHAR FOR BIT DATA, CHAR, and VARCHAR.
<xml-type>	The XML data type, such as base64binary, int, string, and dateTime.

data section

Contains the actual data. Each row is mapped to a row tag. A row tag contains as many column tags as there are columns in the result set, and ordered by the column index. The row tag contains the actual data as an XML data type.

Table 16. Data type mapping conventions

DB2 UDB data type <column-type-name>	**JDBC data type <column-type>**	**XML data type <xml-type>**
BLOB	BLOB	base64Binary
CLOB	CLOB	string
LONGVARCHAR	LONGVARCHAR	string
VARCHAR	VARCHAR	string
CHAR	CHAR	string
CHAR FOR BIT DATA	BINARY	base64Binary
VARCHAR FOR BIT DATA	VARBINARY	base64Binary
LONGVARCHAR FOR BIT DATA	LONGVARBINARY	base64Binary
DATE	DATE	date
TIME	TIME	time
TIMESTAMP	TIMESTAMP	dateTime
-	BOOLEAN	boolean
-	BIT	boolean
TINYINT	TINYINT	int
SMALLINT	SMALLINT	int
INTEGER	INTEGER	int
BIGINT	BIGINT	int
DOUBLE	DOUBLE	double
FLOAT	FLOAT	double
REAL	REAL	float
DECIMAL	DECIMAL	decimal
NUMERIC	NUMERIC	decimal
DATALINK	DATALINK	string
-	ARRAY	anyType
DISTINCT	DISTINCT	string
-	JAVA_OBJECT	string
-	NULL	string
-	OTHER	string
-	STRUCT	string
-	REF	string
	other number of the type	string

The following is the db2WebRowSet.xsd file. The default location of this file is the <contextRoot> directory.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema targetNamespace="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:db2wrs="http://schemas.ibm.com/db2/dqs/db2WebRowSet"
  elementFormDefault="qualified">
   <xs:element name="db2WebRowSet">
  <xs:complexType>
   <xs:sequence>
      <xs:element ref="db2wrs:metadata"/>
    <xs:element name="data">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="row"
                              minOccurs="0"
                              maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="db2wrs:column"
                                    minOccurs="0"
                                    maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Figure 32. db2WebRowSet.xsd (Part 1 of 2)

```
<xs:element name="column"
               type="xs:anyType"
               nillable="true"/>
 <xs:element name="metadata">
  <xs:complexType>
   <xs:sequence>
    <xs:element
                name="column-count"
                type="xs:string" />
    <xs:choice>
     <xs:element name="column-definition"
                       minOccurs="0"
                       maxOccurs="unbounded">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="column-index"
                              type="xs:string" />
        <xs:element name="nullable"
                              type="xs:string" />
        <xs:element name="column-name"
                              type="xs:string" />
        <xs:element name="column-precision"
                              type="xs:string" />
        <xs:element name="column-scale"
                              type="xs:string" />
        <xs:element name="column-type"
                              type="xs:string" />
        <xs:element name="column-type-name"
                              type="xs:string" />
        <xs:element name="xml-type"
                              type="xs:string"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 32. db2WebRowSet.xsd (Part 2 of 2)

Related concepts:

- "Dynamic query services-example queries" on page 93
- "Dynamic database queries that use the Web services provider" on page 90

Related tasks:

- "Configuring and running dynamic database queries as part of Web services provider" on page 91

Related reference:

- "Dynamic query service operations in the Web services provider" on page 99

Verifying and testing Web services provider (WORF)

This section provides an overview of WORF by using the SAMPLE database that comes with DB2. As a starting point, make sure that your application server is ready to run (see "Configuring the Web services provider for WebSphere Application Server on UNIX, Windows, z/OS, and OS/390" on page 36 and "Configuring Web services provider for Apache Jakarta Tomcat on UNIX and Windows" on page 50 for more details). You 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*. The scenario also assumes that you configured *services* on your application server.

Testing Web services applications – a scenario

WORF supports the creation of Web services by using the document access definition extension (DADX) file. 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 33. Simple DADX file: HelloSample.dadx

For the OS/390® and z/OS™ platforms, you might need to modify the name of the table to correspond with the sample DEPARTMENT table that is installed. This table has a default name of DSN8710.DEPT.

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.

Related concepts:
- "Introduction to using DB2 as a Web services provider – WORF" on page 25
- "Defining the Web service with the document access definition extension file" on page 66
- "Overview of the Web services process" on page 32

Related tasks:
- "Testing the Web service" on page 109

Testing the Web service

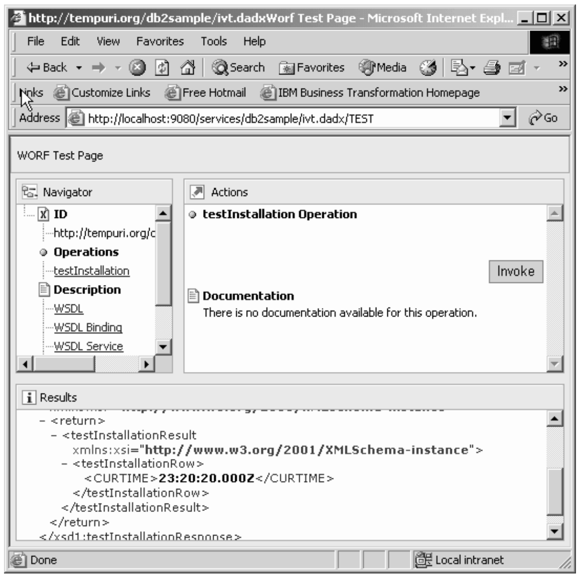
You can test your Web service by completing the following tasks:

Procedure:
1. Ensure that the you have installed the WORF samples in the directory \WEB-INF\classes\groups\dxx_sample.
2. Ensure that you deployed the sample application in a server such as WebSphere Application Server (WAS).

3. Open a browser window and type the following uniform resource locator (URL) to begin the test:

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

   Remember that the *your WebAppServer* identifier depends on your Web server configuration. When you type the address, you see the following automatically generated documentation and test page:



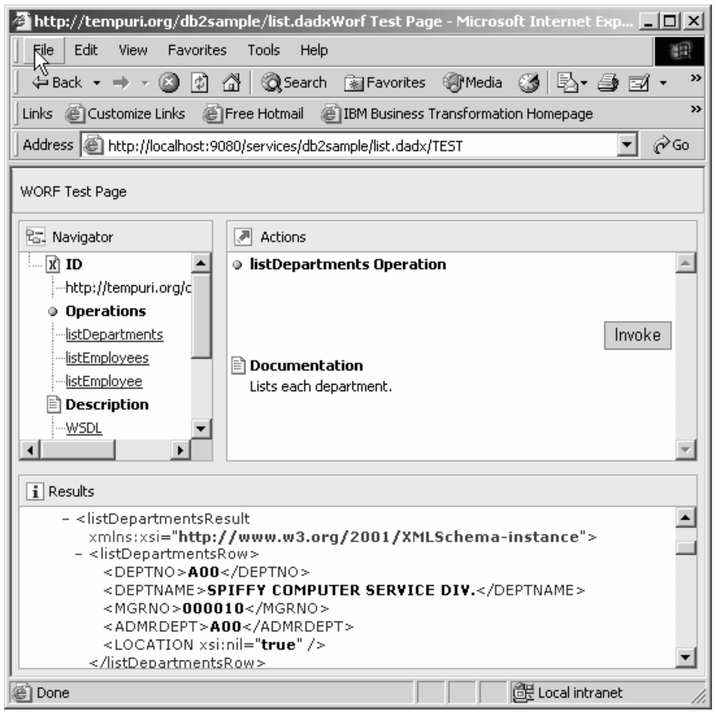
There is no documentation available for this operation.

```
  - <return>
    - <testInstallationResult
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      - <testInstallationRow>
          <CURTIME>23:20:20.000Z</CURTIME>
        </testInstallationRow>
      </testInstallationResult>
    </return>
  </xsd1:testInstallationResponse>
```

Figure 34. The WORF test page

4. Test the `listDepartments` operation:
   a. Click the `listDepartments` link in the **Methods** pane.
   b. Click the **Invoke** push button in the **Inputs** pane.

      You can see the Extensible Markup Language (XML) result of the operation in the Result pane:


http://tempuri.org/db2sample/list.dadxWorf Test Page - Microsoft Internet Exp...
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail IBM Business Transformation Homepage
Address http://localhost:9080/services/db2sample/list.dadx/TEST
Go
WORF Test Page
Navigator
ID
http://tempuri.org/c
Operations
listDepartments
listEmployees
listEmployee
Description
WSDL
Actions
listDepartments Operation
Invoke
Documentation
Lists each department.
Results
Done
Local intranet

```
  - <listDepartmentsResult
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    - <listDepartmentsRow>
        <DEPTNO>A00</DEPTNO>
        <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
        <MGRNO>000010</MGRNO>
        <ADMRDEPT>A00</ADMRDEPT>
        <LOCATION xsi:nil="true" />
      </listDepartmentsRow>
```

Figure 35. Result of the query

Related concepts:

- "Introduction to using DB2 as a Web services provider – WORF" on page 25
- "Testing Web services applications – a scenario" on page 109
- "Installing Web applications" on page 131

Related tasks:

- "Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX" on page 39

Accessing the Web service with GET, POST, and SOAP bindings

The Web object runtime framework (WORF) test page acts as a simple Hypertext Markup Language (HTML) client of the Web Service and uses the Hypertext Transfer Protocol (HTTP) POST binding. You can access the Web service by using the HTTP GET, and SOAP bindings. You can invoke the `listDepartments` operation with the HTTP GET, and POST bindings. The following example shows the basic syntax of the GET or POST binding:

```
http://server:port/contextRoot/group/dadx_file/operationName
```

If you have installed the WORF samples, you can type the following uniform resource locator (URL) to issue a GET request:

```
http://<yourWebAppServer:9080>/services/db2sample/HelloSample.dadx/listDepartments
```

The *localhost* port number, designated here by <*yourWebAppServer*> depends on your own current machine. The WORF `listDepartments` operation returns an Extensible Markup Language (XML) response that you can save to a file. The HTTP response is the same for GET and POST.

```
<?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 36. XML response document

The GET binding request does not send a request document. Instead, you attach all of the necessary parameters in the query string to the URL You can attach a query string to the URL with a question mark (?). The delimiter between any parameter=value pair is the ampersand (&). Any special characters must be URL encoded. The following example is a GET binding request that uses a query string with the question mark, and a delimiter.

```
http://server:port/contextRoot/group/dadx/
   operationName?param1=abc&param2=1234&param3=thi&20is&20a&20parameter
```

The following example is a dynamic query service. This is a GET binding request.

```
http://localhost:9080/services/db2sample/dqs.dadx/
   executeQuery?queryInputParameter=select+*+from+employee&extendedInputParameter=
      %3Cproperties%3E%0D%0A%3C%2Fproperties%3E%0D%0A
```

A POST binding issues an HTTP POST request. A POST bind request sends a request document. The document contains the request parameter, but the parameter is not in XML format. An HTTP client application, such as a Web browser, creates the request document. A Web browser usually creates a request document from input forms that are sent to the server. The following syntax is a typical POST bind request:

```
http://server:port/contextRoot/group/dadx/operationName
```

The following example is a dynamic query service. This is a POST binding request.

```
http://localhost:9080/services/db2sample/dqs.dadx/executeQuery
```

The query is the same as the GET binding request, except that the information that follows the question mark (?) is in the request document, and not part of the URL. In the following example, the content type is www-urlencoded:

```
queryInputParameter=
   select+*+from+employee&extendedInputParameter=
%3Cproperties%3E%0D%0A%3C%2Fproperties%3E%0D%0A
```

The GET and POST response for the dynamic query service request is:

```
<?xml version="1.0"?>
<xsd1:executeQueryResponse
  xmlns:xsd1="http://schemas.ibm.com/db2/dqs/types/soap"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <queryOutputParameter>
      ...
   </queryOutputParameter>
</xsd1:executeQueryResponse>
```

HTTP GET, and POST bindings are generally the same as any other HTTP GET, and POST requests. The HTTP GET binding adds any input parameters to the operation to the uniform resource locator (URL). But, the HTTP POST binding sends the parameters in the request body.

Related concepts:
- "Web services provider features" on page 30
- "Apache SOAP configurations" on page 135
- "SOAP binding" on page 113

Related reference:
- "Dynamic query service operations in the Web services provider" on page 99
- "Web services samples – PartOrders.dadx" on page 127

SOAP binding

The simple object access protocol (SOAP) binding also uses Hypertext Transfer Protocol (HTTP) POST, but it sends the operation name, input parameters, and other information as an Extensible Markup Language (XML) request body.

The SOAP request binding issues an SOAP request over HTTP. SOAP is used as a message protocol, for request and response messages. The SOAP request specifies how the request and response message should appear. The SOAP binding requests are XML documents that follow a certain schema.

SOAP operates on top of an HTTP POST request. HTTP is the transport protocol, and SOAP is the message protocol. A client application must know how to build SOAP request documents. The definition and the format of parameters, and other information is defined in separate Web services description language (WSDL) document.

Use the following uniform resource locator (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
```

There is no operation name in the request. The information is now in the SOAP request document.

The following example which is for an RPC style, is a dynamic query service using a SOAP binding.

```
<SOAP-ENV:Envelope
   1 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <SOAP-ENV:Body>
     2 <ns0:executeQuery
      3 xmlns:ns0="http://schemas.ibm.com/db2/dqs">
      4 <queryInputParameter>
           select * from employee
      </queryInputParameter>
     <extendedInputParameter>
          <properties/>
     </extendedInputParameter>
   </ns0:executeQuery>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP request has key information:

1. The SOAP envelope.
2. The operation name.
3. The actual request document. In the WORF environment, this is an XML document that is now an actual Web service SOAP request.
4. The parameters.

The SOAP response is:

```
<?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:executeQueryResponse
      xmlns:ns1="http://schemas.ibm.com/db2/dqs"
          SOAP-ENV:encodingStyle="http://xml.apache.org/xml-soap/literalxml">
    <queryOutputParameter>
       ...
    </queryOutputParameter>
   </ns1:executeQueryResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The information that is contained within the `SOAP-ENV:Body` tag is the actual response document. In the WORF environment, the response document is an XML document. This is an actual Web service SOAP response.

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

Related concepts:

- "Accessing the Web service with GET, POST, and SOAP bindings" on page 111
- "The Web service consumer functions" on page 145
- "Overview of the Web services process" on page 32

Related tasks:
- "Testing the Web service" on page 109

Related reference:
- "Dynamic query service operations in the Web services provider" on page 99

Web services description language

Web service providers are described by Web services description language (WSDL) documents. The key to the Web service is the Web services description language document.

The WSDL is an XML document that describes Web services as a collection of endpoints, or ports. An endpoint is an addressable location at which a Web service can be accessed according to the associated binding of a specified interface. One Web service can have multiple endpoints. The endpoints in a WSDL operate on messages. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a document-oriented or procedure-oriented (RPC) style binding. A SOAP binding can also have an encoded use or a literal use.

As Figure 37 shows, the Web service provider implements a service and publishes the interface to some service broker, such as UDDI. The service requester can then use the service broker to find a Web service. When the requester finds a service, the requester binds to the service provider so that the requester can use the Web service. The requester invokes the service by exchanging SOAP (Simple object access protocol) messages between the requester and provider.

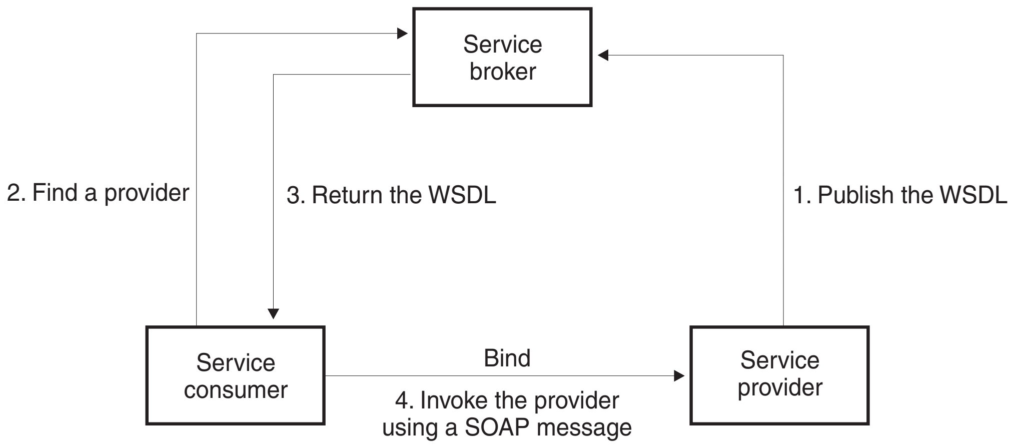


Figure 37. Web services as a service oriented architecture

The SOAP specification defines the layout of an XML-based message. A SOAP message is contained in a SOAP envelope. The envelope consists of an optional SOAP header and a mandatory SOAP body. The SOAP header can contain information about the actual message, such as encryption information or authentication information. The SOAP body contains the actual message. The SOAP specification also contains a default encoding for programming language bindings, which is called the SOAP encoding.

A WSDL document can contain one or more Web services. A service consists of one or more ports with a binding. The WSDL document can have one or more port types. A port type has one or more operations with abstract input and output messages. A binding refers to the process of associating protocol or data format information with an abstract entity like a message, operation, or a portType. A binding creates a concrete protocol and data format specification for a particular port type. A port is an endpoint that is a binding and a Web address.

The example in Figure 38 on page 117 shows the WSDL definition of a simple service providing stock quotes. The Web service supports a single operation that is named `GetLastTradePrice`. The service is deployed using the SOAP 1.1 protocol over HTTP. The request reads a ticker symbol as input, which is a string data type, and returns the price, which is a float data type. The type shown in this example is an XML schema definition. You can use XSD files to associate tables and columns in a DB2® Universal Database table to your Web service.

The WSDL style that is specified in the <soap:binding> element is the document style. The <soap:operation> element provides information for the operation as a whole. The style attribute in the <soap:operation> element indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (messages containing documents). The value of this attribute also affects the way in which the body of the SOAP message is constructed. If the attribute is not specified, it defaults to the value specified in the <soap:binding> element. The IBM® DB2 Information Integrator Web services provider contains samples that are set to use RPC style. For new applications, you should use document style for maximum interoperability. In Version 8.2, the Web services provider uses an RPC style with literal usage and type nodes instead of an RPC style with literal usage and element nodes. However, there is no change to the SOAP messages from earlier releases of Web services provider.

The complete example and the WSDL specification is at the W3C site (http://www.w3.org/TR/2001/NOTE-wsdl-20010315).

```
<?xml version='1.0'?>
<definitions name='StockQuote'
...


<types>
       <schema targetNamespace='http://example.com/stockquote.xsd'
              xmlns='http://www.w3.org/2000/10/XMLSchema'>
           <element name='TradePriceRequest'>
              <complexType>
                  <all>
                      <element name='tickerSymbol' type='string'/>
                  </all>
              </complexType>
           </element>
           <element name='TradePrice'>
              <complexType>
                  <all>
                      <element name='price' type='float'/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

<message name='GetLastTradePriceInput'>
...
</message>

    <portType name='StockQuotePortType'>
        <operation name='GetLastTradePrice'>
           <input message='tns:GetLastTradePriceInput'/>
           <output message='tns:GetLastTradePriceOutput'/>
        </operation>
    </portType>


    <binding
       name='StockQuoteSoapBinding'
       type='tns:StockQuotePortType'>
        <soap:binding
          style='document'
            transport='http://schemas.xmlsoap.org/soap/http'/>
        <operation name='GetLastTradePrice'>
           <soap:operation
             soapAction='http://example.com/GetLastTradePrice'/>
           <input>
               <soap:body use='literal'/>
           </input>
           <output>
               <soap:body use='literal'/>
           </output>
        </operation>
    </binding>

   <service name='StockQuoteService'>
        <documentation>My first service</documentation>
        <port name='StockQuotePort'
           binding='tns:StockQuoteBinding'>
           <soap:address
             location='http://example.com/stockquote'/>
        </port>
    </service>

</definitions>
```

Figure 38. Example of a WSDL

Since WSDL documents have a certain structure, Web services developers might need to use types from external schemas either at the definitions level of the WSDL document, or the types level of the WSDL document. To use external schemas, you can use imported schema definitions in your WSDL. WORF supports two types of imports during the WSDL generation:

An import at the /definitions scope of a WSDL
http://schemas.xmlsoap.org/wsdl/:import

An import at the /definitions/type/schema scope of a WSDL
http://www.w3.org/2001/XMLSchema:import

You can add import definitions by using a group.imports file. If a group.imports file exists in the resources of the Web service group directory, then WORF includes the group.imports information in the generated WSDL. The following example is a group.imports file:

```
<?xml version='1.0' encoding='UTF-8'?>
<imports xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
         xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
   <wsdl:import namespace='http://some/namespace/1'
           location='schema1.xsd'/>
   <wsdl:import namespace='http://some/namespace/2'
           location='schema2.xsd'/>
   <xsd:import namespace='http://some/namespace/3'
           schemaLocation='schema3.xsd'/>
   <xsd:import namespace='http://some/namespace/4'
           schemaLocation='schema4.xsd'/>
</imports>
```

This example defines two imports in the /definitions scope that will be added to the WSDL (schema1.xsd and schema2.xsd). The example defines two imports in the /definitions/types/schema scope that will be added to the WSDL (schema3.xsd and schema4.xsd). The WSDL that WORF generates that includes the import definitions from the above file has the following structure:

```
<?xml version='1.0' encoding='UTF-8'?>
<definitions>
  <wsdl:documentation xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
       xmlns='http://schemas.xmlsoap.org/wsdl/'>
     Documentation Text Node
  </wsdl:documentation>
  <import location='schema2.xsd'
            namespace='http://some/namespace/2'/>
  <import location='schema1.xsd'
            namespace='http://some/namespace/1'/>
  <types>
    <schema>
      <import namespace='http://some/namespace/4'
            schemaLocation='schema4.xsd'/>
      <import namespace='http://some/namespace/3'
            schemaLocation='schema3.xsd'/>
      <element name='executeQueryResponse'> .... </element>
      <element name='executeQuery'> .... </element>
    </schema>
  </types>
   ...
</definitions>
```

If you installed the WORF examples, and have an application called `services`, then you can request a Web services description language (WSDL) document for the service, *HelloSample.dadx*. Use the following uniform resource locator (URL) to request the WSDL. The *localhost* port number, designated here by <*yourWebAppServer*> depends on your own current machine:

```
http://<yourWebAppServer>/services/db2sample/HelloSample.dadx/WSDL
```

WORF automatically generates the WSDL document, from DADX.

Related concepts:
- "WSDL from a DADX file" on page 87

Related tasks:
- "Customizing the group.properties file" on page 64
- "Testing the Web service" on page 109

UDDI business registries

Register your 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 UDDI Best Practices.

The service instance document contains the address from which you deploy the service and it imports the binding document. Many service instances might refer to a common binding document. You register the binding document 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 uniform resource locator (URL). The *localhost* port number, designated here by <*yourWebAppServer*> depends on your own current machine:

```
http://<yourWebAppServer>/services/db2sample/HelloSample.dadx/WSDLservice
```

Request the WSDL binding document with the URL:

```
http://<yourWebAppServer>/services/db2sample/HelloSample.dadx/WSDLbinding
```

Related concepts:
- "Overview of the Web services process" on page 32

Related tasks:
- "Testing the Web service" on page 109

XML schema definitions

An XML schema defines the data types used in the Web service interface. Request the XML schema definitions for the service by the uniform resource locator (URL). The *localhost* port number, designated here by <*yourWebAppServer*> depends on your own current machine:

```
http://<yourWebAppServer>/services/db2sample/HelloSample.dadx/XSD
```

WORF generates an XML schema file similar to the example in Figure 39 on page 120.

```
<?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 39. The XML schema definition file

The DB2® XML Extender can use document type definitions (DTDs) to define the schema of XML documents, so the WORF run-time automatically translates the DTD into an XML Schema. For example, if the DTD *order.dtd* defines an XML document, then you can use the following URL to request the translation into XML Schema:

```
http://<yourWebAppServer>/services/db2sample/order.dtd/XSD
```

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66
- "Testing Web services applications – a scenario" on page 109
- "Web services that exist from Web services provider" on page 120

Related reference:

- Appendix C, "XML schema for the DADX file," on page 247

Web services that exist from Web services provider

Within a directory of Web applications, or a group of Web services, there is potentially a large number of Web services that you can use on your network. But before you can use these Web services, you must find them and get information about them. Web Services Inspection Language (WSIL) makes this search process easier.

Web services inspection language document

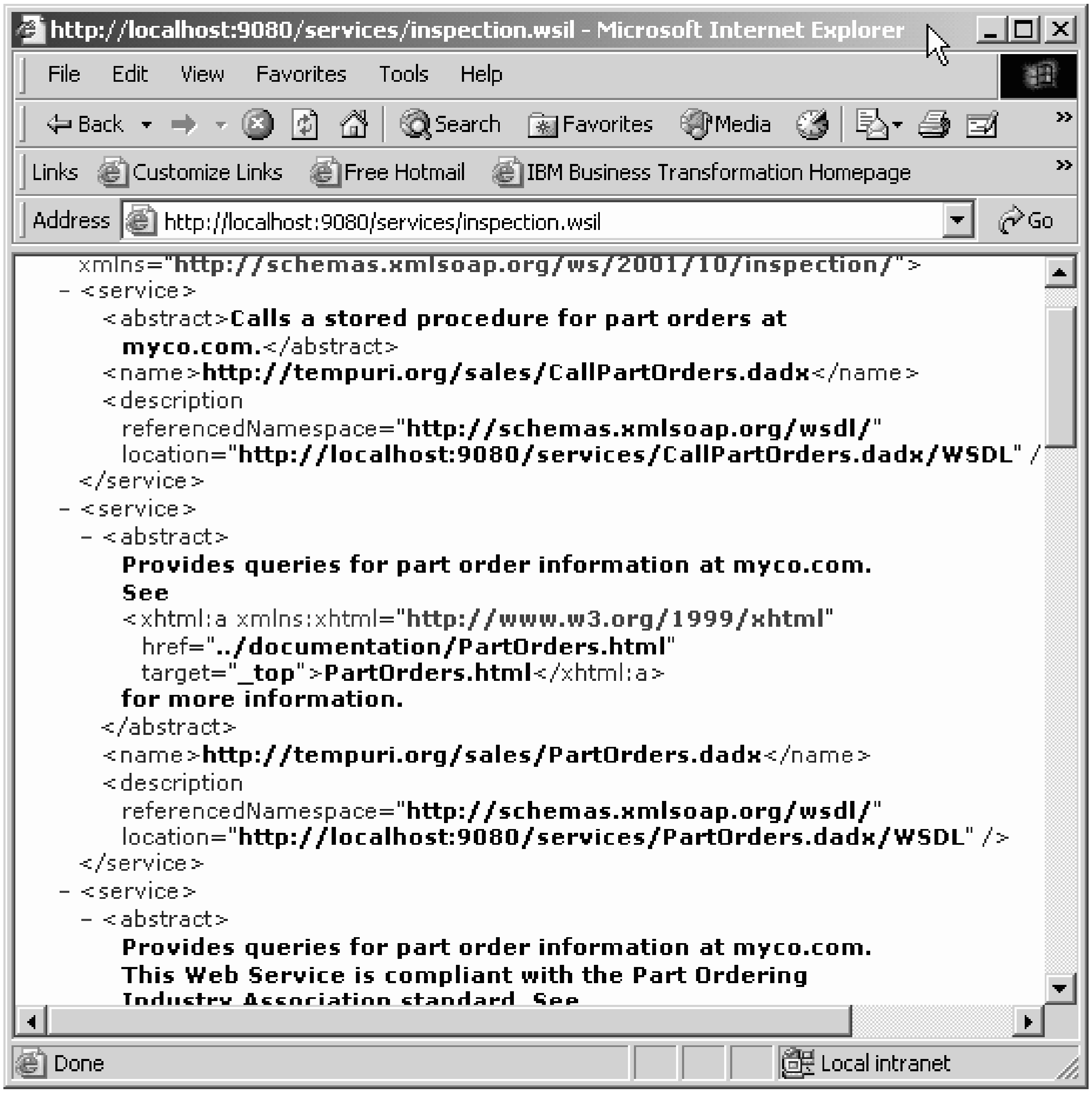
DB2® Web services provides a way to find the Web services operations that you need. By using WORF, you inspect all of the Web services available within an application, or within a group. The inspection generator produces an XML document that is a list of the Web services available to you. The list is a report of the Web services at the group directory level, if you run the generator from the group directory. The list is a report of the Web services at the application directory level if you run the generator from the application directory. You run the inspection generator from your browser by typing *<your-Web-server>:9080/<context_root_name>/inspection.wsil* in your browser.

The examples in this topic are based on the WORF samples and a WORF sample application named *services*

The following example creates a list of Web services that are available from the Web application named *services*:

```
http://localhost:9080/services/inspection.wsil
```

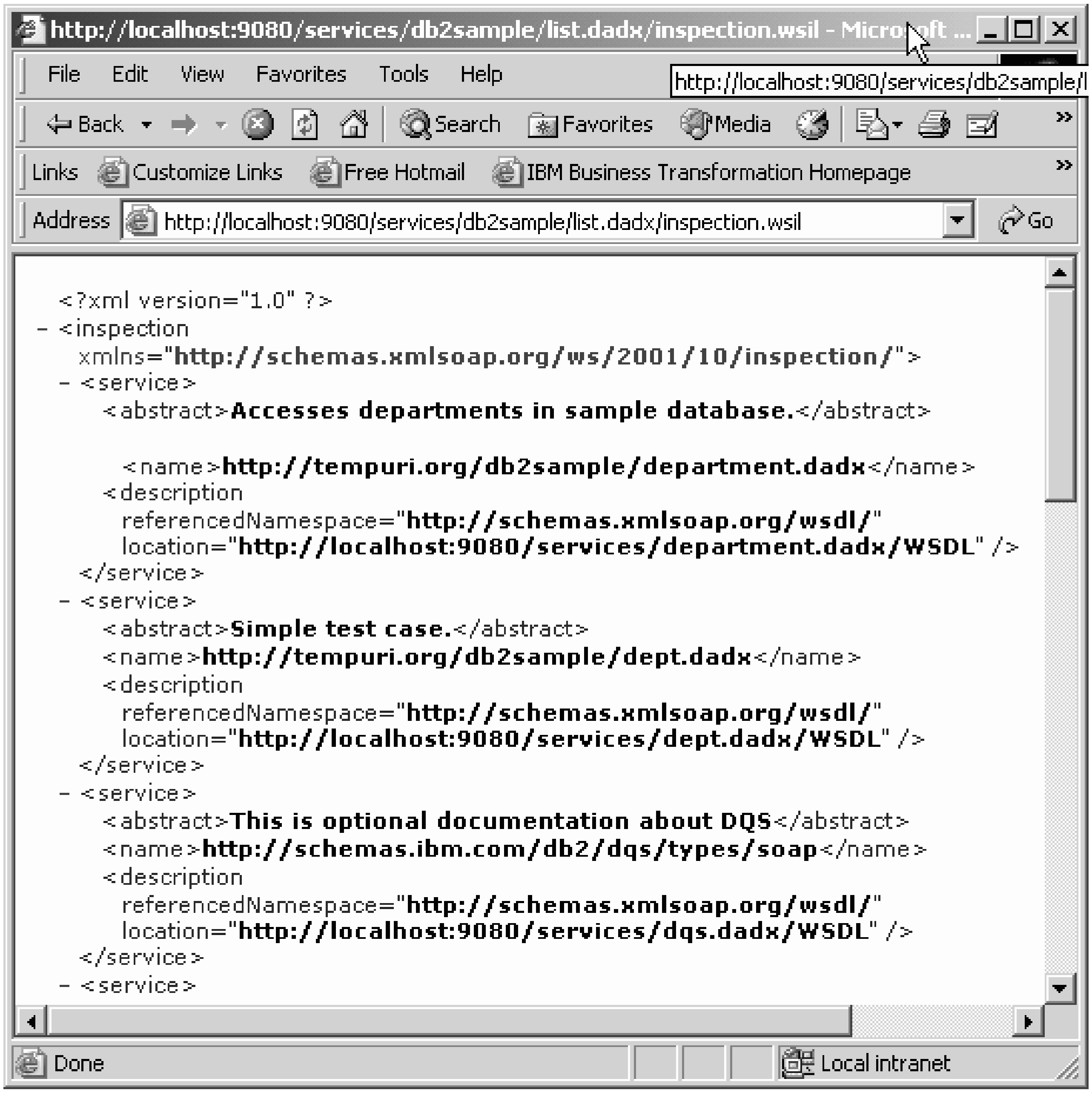
An inspection.wsil at the application level looks like this:


http://localhost:9080/services/inspection.wsil - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail IBM Business Transformation Homepage
Address http://localhost:9080/services/inspection.wsil Go
Done
Local intranet

```
   xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
 - <service>
     <abstract>Calls a stored procedure for part orders at
       myco.com.</abstract>
     <name>http://tempuri.org/sales/CallPartOrders.dadx</name>
     <description
       referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
       location="http://localhost:9080/services/CallPartOrders.dadx/WSDL" /
   </service>
 - <service>
   - <abstract>
       Provides queries for part order information at myco.com.
       See
       <xhtml:a xmlns:xhtml="http://www.w3.org/1999/xhtml"
         href="../documentation/PartOrders.html"
         target="_top">PartOrders.html</xhtml:a>
       for more information.
     </abstract>
     <name>http://tempuri.org/sales/PartOrders.dadx</name>
     <description
       referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
       location="http://localhost:9080/services/PartOrders.dadx/WSDL" />
   </service>
 - <service>
   - <abstract>
       Provides queries for part order information at myco.com.
       This Web Service is compliant with the Part Ordering
       Industry Association standard. See
```

Figure 40. WSIL at the application level

An inspection.wsil at the group level looks like this:


http://localhost:9080/services/db2sample/list.dadx/inspection.wsil - Microsoft ...
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail IBM Business Transformation Homepage
Address http://localhost:9080/services/db2sample/list.dadx/inspection.wsil
Go
Done
Local intranet

```
  <?xml version="1.0" ?>
- <inspection
   xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
  - <service>
      <abstract>Accesses departments in sample database.</abstract>

        <name>http://tempuri.org/db2sample/department.dadx</name>
      <description
        referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
        location="http://localhost:9080/services/department.dadx/WSDL" />
    </service>
  - <service>
      <abstract>Simple test case.</abstract>
      <name>http://tempuri.org/db2sample/dept.dadx</name>
      <description
        referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
        location="http://localhost:9080/services/dept.dadx/WSDL" />
    </service>
  - <service>
      <abstract>This is optional documentation about DQS</abstract>
      <name>http://schemas.ibm.com/db2/dqs/types/soap</name>
      <description
        referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
        location="http://localhost:9080/services/dqs.dadx/WSDL" />
    </service>
  - <service>
```

Figure 41. WSIL at the group level

To ensure that the WSIL that you generate includes a Web service, the Web service must conform to the following criteria:

- The DADX file that describes the Web service must be valid.
- You must define a group.properties file that belongs to the Web service.
- The web.xml file must contain appropriate servlet mappings that identify the group.

The servlet mapping in the web.xml file for the WSIL should look like the following example:

```
<servlet>
    <servlet-name>wsil</servlet-name>
    <display-name>wsil</display-name>
```

```
        <servlet-class>
           com.ibm.etools.webservice.rt.wsil.servlet.WSILInvoker
        </servlet-class>
        <init-param>
          <param-name>soap-engine</param-name>
          <param-value>apache-axis</param-value>
        </init-param>
        <load-on-startup>-1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>wsil</servlet-name>
        <url-pattern>/inspection.wsil</url-pattern>
    </servlet-mapping>
```

When a Web service is invoked, WORF reads the web.xml file for information about how to run the service, such as servlet information, and group information. The web.xml file includes the servlet definition so that the WSIL specification associates the correct Web operations with the WORF samples. WORF dynamically generates a WSIL document that contains all of the available Web services in the groups directory of the Web application server. If you add a Web service to the group.properties file after the Web application server is started, you do not need to restart the server before you start the WSIL generator.

Web services list page

Another kind of inspection document that you can produce is the Web services list page. The Web services list page is an HTML document that contains a list of all of the available Web services in an application directory or in the groups directory of the Web application server. The list also contains links to the WORF samples and the WSDL of the services. You can access this page by typing the following URL in your browser:

Application level

```
<your-Web-server>:9080/<context_root_name>/LIST
```

Group level

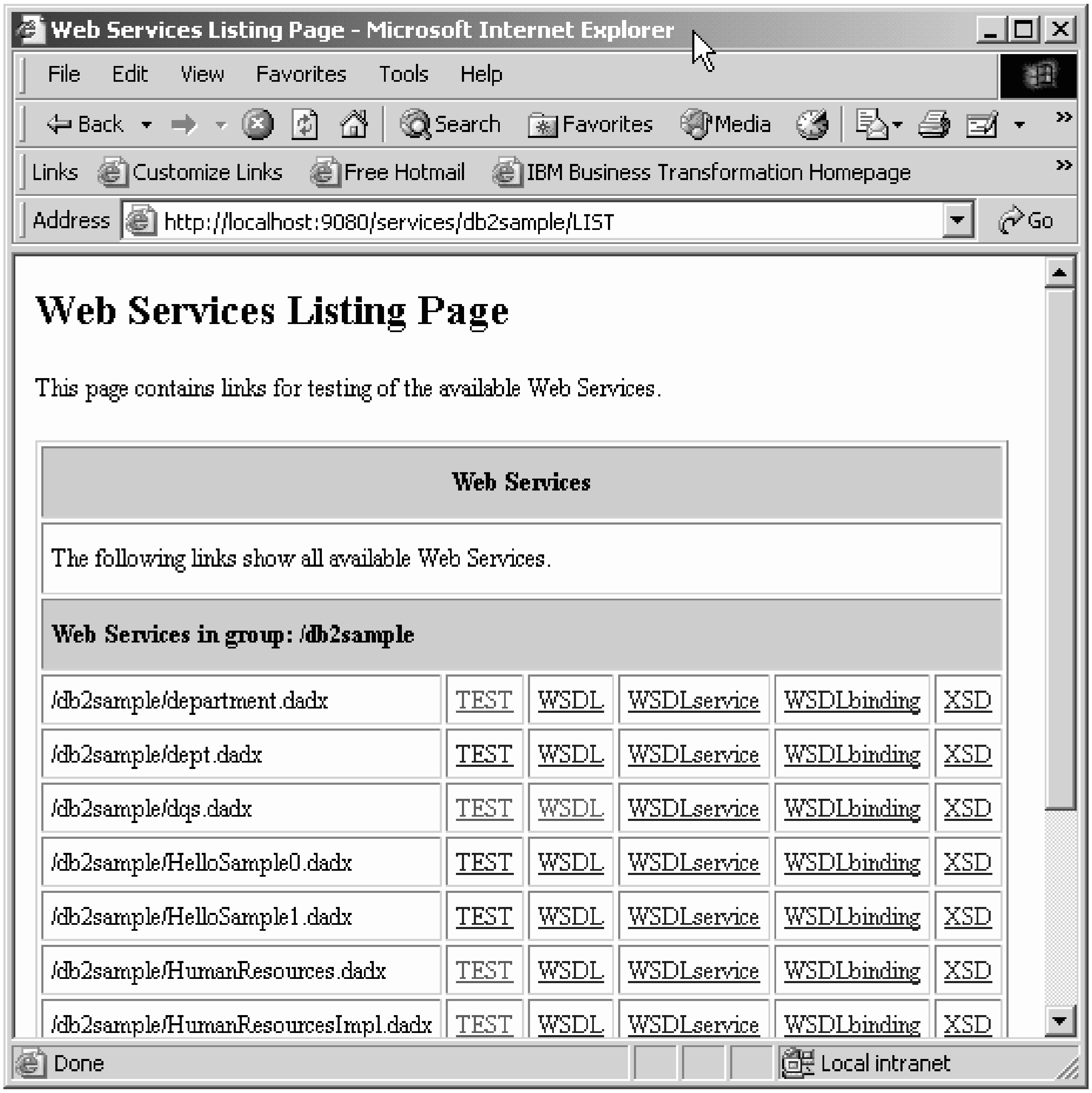
```
<your-Web-server>:9080/<context_root_name>/<group name>/LIST
```

The servlet mapping and URL pattern in the web.xml file for the list page should look like the following example:

```
<servlet>
        <servlet-name>list</servlet-name>
        <display-name>list</display-name>
        <servlet-class>
          com.ibm.etools.webservice.rt.list.servlet.ListInvoker
        </servlet-class>
        <init-param>
          <param-name>soap-engine</param-name>
          <param-value>apache-axis</param-value>
        </init-param>
        <load-on-startup>-1</load-on-startup>
    </servlet>

   <servlet-mapping>
        <servlet-name>list</servlet-name>
        <url-pattern>/LIST</url-pattern>
    </servlet-mapping>
```

A list page at the group level looks like this:


Web Services Listing Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail IBM Business Transformation Homepage
Address http://localhost:9080/services/db2sample/LIST
Go
Done
Local intranet

Web Services Listing Page

This page contains links for testing of the available Web Services.

Web Services					
The following links show all available Web Services.					
Web Services in group: /db2sample					
/db2sample/department.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/dept.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/dqs.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/HelloSample0.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/HelloSample1.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/HumanResources.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
/db2sample/HumanResourcesImpl.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD

Figure 42. Web services list page

Related concepts:

- “XML schema definitions” on page 119

Related tasks:

- “Testing the Web service” on page 109
- “Defining the web.xml and group.properties files” on page 59
- “Installing or migrating WORF to work with WebSphere Application Server Version 5 or later for Windows and UNIX” on page 39
- “Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX” on page 41

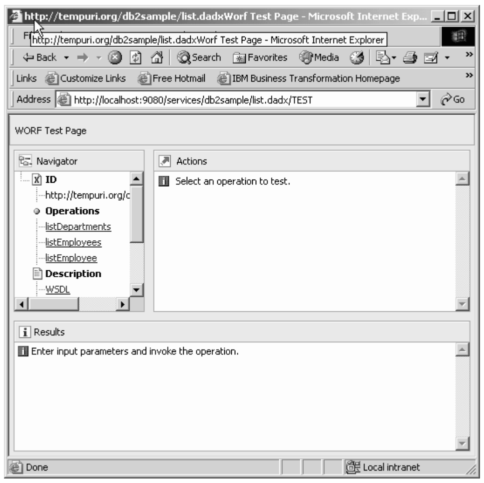
Web services documentation

You can include documentation in the DADX for the service as a whole and for each operation. Figure 43 illustrates how to add documentation:

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

Figure 43. HelloSample.dadx

The documentation can contain any valid Extensible Markup Language (XML). For proper display in a browser, you should use XHTML. If you use XHTML, then define the XHTML namespace for the documentation. When you request the test page, it also includes the documentation:


http://tempuri.org/db2sample/list.dadxWorf Test Page - Microsoft Internet Exp...

http://tempuri.org/db2sample/list.dadxWorf Test Page - Microsoft Internet Explorer

Address http://localhost:9080/services/db2sample/list.dadx/TEST

WORF Test Page

Navigator

ID

http://tempuri.org/c

Operations

listDepartments

listEmployees

listEmployee

Description

WSDL

Actions

Select an operation to test.

Results

Enter input parameters and invoke the operation.

Done

Local intranet

Figure 44. WORF test page with documentation

Related concepts:

- “Testing Web services applications – a scenario” on page 109

Related reference:

- “A simple DADX file” on page 75
- “Syntax of the DADX file” on page 67

Web services 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 the developing of Java™ Server Pages. You can turn off automatic reloading when you deploy your DADX Web services to a production server.

Related concepts:
- "Web services provider features" on page 30
- "Overview of the Web services process" on page 32

Related tasks:
- "Customizing the group.properties file" on page 64
- "Testing the Web service" on page 109

Web services samples – PartOrders.dadx

The example in this topic uses a database sample called dxx_sales_db. This is the sample database used in the documentation and samples shipped with DB2 XML Extender and with WORF. The dxx_sales_db database stores information about part orders.

Suppose that you must provide a Web Service that retrieves orders that are 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. This is 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:
- Hypertext Transfer Protocol (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 that you deploy the Web services 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 uniform resource locator (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, you send the query string 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 Transmission Control Protocol (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, you invoke the services 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`. The following example is a SOAP request that is traced with a TCP monitor. It is 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/2001/XMLSchema"
        xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <documentation>
      Provides queries for part order information at myco.com.
      See <xhtml:a href="../documentation/PartOrders.html" target="_top">
       PartOrders.html</xhtml:a> for more information.
    </documentation>

    <operation name="findAll">
      <documentation>

        Returns all the orders with their complete details.
      </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 45. The PortOrder.DADX file (Part 1 of 3)

```
<operation name="findByColor">
      <documentation>
Returns all  the orders that include one or
      more parts that have the specified
      color, and only shows the details for those parts.
</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 45. The PortOrder.DADX file (Part 2 of 3)

```
 <operation name="findByMinPrice">
      <:documentation>
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.
</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 45. The PortOrder.DADX file (Part 3 of 3)

Related concepts:

- "Accessing the Web service with GET, POST, and SOAP bindings" on page 111
- "Introduction to using DB2 as a Web services provider – WORF" on page 25

- “Testing Web services applications – a scenario” on page 109
- “SOAP binding” on page 113

Related tasks:
- “Testing the Web service” on page 109

Deploying and testing your Web application

You can now work from your design and compose queries and write applications to achieve your business goals. Then you deploy the programs on the Web. You can deploy the same set of files in many ways.

Installing Web applications

You can use Web archives files (WAR) to package, distribute, and install Web applications. You generally package the files that comprise your Web application in a single WAR file for deployment. A WAR file might contain the *web.xml* server configuration files, the group.properties configuration files, and the DAD and DADX files. The WORF samples that you deployed in Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX contain examples of two WAR files: *apache-services.war* and *axis-services.war*. Some development environments, like WebSphere® Studio, provide automatic deployment capabilities. But, you can disable this automatic deployment if you want to provide your own deployment descriptor file for your Web service.

Related tasks:
- “Preparing and creating the Web archive file” on page 136
- “Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX” on page 41

Java 2 Enterprise Edition applications

When you deploy a Java™ 2 Enterprise Edition (J2EE) application, the following components must be built for an e-business application:

Web archive (WAR)
The Web-related components (HTML, JavaScript™, JavaServer Pages)

Java archive (JAR)
The Java classes that make up the business logic components

Enterprise archive (EAR)
The Java archive files plus Web archive files that make up an enterprise solution

The minimum deployable unit in WebSphere® Application Server 5.0 is a Web archive file. If the application is developing Enterprise JavaBeans™, then a Java archive file and an Enterprise archive file are required.

Related concepts:
- “Advantages of designing queries in IBM DB2 Information Integrator” on page 164
- “Defining a group of Web services” on page 58

Installing the application server for DB2 in DB2 Information Integrator

Application servers enable enterprises to develop, deploy, and integrate next-generation e-business applications. You can use application servers as tools to administer your Web applications.

Starting in Version 8.1.2, DB2® Universal Database, provides an embedded application server, referred to as the application server for DB2. If you use the application server for DB2 UDB, you do not need to install a separate application server to run your DB2 UDB Web applications on Windows®, Linux, AIX, and Solaris.

Prerequisites:

- DB2 Universal Database ESE Version 8.1.2 or later
- At least one DB2 UDB instance must exist
- Issue the following command for your environment:

  ```
  <db2instance path>/sqllib/db2profile (for Windows)
  . <db2instance path>/sqllib/db2profile (for UNIX systems)
  ```

Restrictions:

You can have only one DB2 application server in a system that has one or multiple DB2 UDB instances.

Procedure:

Install the application server for DB2 from the *Java application development and Web administration tools supplement for DB2* CD. DB2 Universal Database™ provides this CD with the DB2 Universal Database™ installation package. To install the application server for DB2:

```
db2appserverinstall
     -asroot path
     -hostname name
```

-asroot
: The absolute path for the application server installation.

-hostname
: The name of the host system.

Related tasks:

- "Installing the application server for DB2" in the *Installation and Configuration Supplement*
- "Uninstalling the application server for DB2" in the *Installation and Configuration Supplement*

Starting and stopping the application server for DB2 in Information Integrator

You can start and stop the application server for DB2 UDB from the *bin* subdirectory of the application server for DB2 directory. You can also use a stored procedure named DB2EAS.SERVER to start and stop the application server.

Procedure:

To start and stop the application server for DB2 UDB use the following commands:

```
startServer <serverName>
stopServer <serverName>
```

The start and stop commands require the following parameter:

serverName
: The name of the application server that you want to start.

Refer to *WebSphere Application Server System Administration* for information on deploying and managing applications so that you can deploy the WORF samples with the application server for DB2. After you deploy the WORF samples with the application server for DB2, you can access the WORF test page from your browser.

Related tasks:

- "Starting the application server for DB2 locally" in the *Installation and Configuration Supplement*
- "Stopping the application server for DB2 locally" in the *Installation and Configuration Supplement*
- "Starting the application server for DB2 remotely" in the *Installation and Configuration Supplement*
- "Stopping the application server for DB2 remotely" in the *Installation and Configuration Supplement*
- "Deploying WORF examples on WebSphere Application Server Version 5.1 or later for Windows and UNIX" on page 41
- "Installing the application server for DB2 in DB2 Information Integrator" on page 132

Generating deployment descriptors

WebSphere Application Server 5.1 when using Apache SOAP, 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 starts. For OS/390 and z/OS platforms, automatic deployment is always enabled, so you do not have to generate your own deployment descriptors.

Procedure:

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 Extensible Markup Language (XML) file named dds.xml, which the WebSphere ConfigManager reads when the Web application starts. The Apache SOAP configuration managers deploy the Web services 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 class path.
2. Generate the deployment descriptor from a DADX file with the DADX2DD command. A deployment descriptor file is not required for Apache Axis. Use 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. The deployment descriptor requires this parameter.

-p The group path. The deployment descriptor requires this parameter.

-n The group name. The deployment descriptor requires this parameter.

-i The name of the DADX file or the directory that contains one or more DADX files. The directory can include one or more subdirectories that contain one or more DADX files. If you use a directory name, use the root directory of the Web application, such as the directory that contains files dds.xml and WEB-INF. The *–i* parameter is optional. If present, a DADX file must exist and be readable. If absent then the name of the DADX file comes 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. It is overwritten if it already exists. If absent then the deployment descriptor file writes to standard output. To indicate standard output, use a dash (–).

-s The target SOAP engine. Valid values include *apache-soap* and *apache-axis.*

When a Web service is invoked, WORF reads the web.xml file to determine which SOAP engine classes to load. If the SOAP engine cannot be loaded, the default SOAP engine is used. If the soap-engine parameter is not specified in the web.xml file, the default soap engine is Apache Axis. For Apache SOAP, the output of the `Dadx2Dd` command can be inserted into the ddx.xml file. For Apache Axis, the deployment descriptor is always generated dynamically at runtime. The output is for information only.

You can switch SOAP engines at the time you execute your Web service. You must make sure that the deployment descriptor is in the dds.xml file before you switch the SOAP engines. The dds.xml file is only for Apache SOAP. A deployment descriptor file is not needed for Apache Axis. Prior to switching to Apache Axis, make sure that the deploy.wsdd file is available to your application.

You can run the deployment descriptor generator (`Dadx2Dd`) without being connected to a SOAP engine.

For example, if the current directory is *WEB-INF*, the following command reads the *ZipCity.dadx* file from the dxx_travel group. It then 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 Apache SOAP ISD file and add it to the *dds.xml* file in your Web application directory.

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

<dds>

<isd:service xmlns:isd='http://xml.apache.org/xml-soap/deployment'
  id='http://tempuri.org/travel/ZipCity.dadx'>
   <isd:provider
    type='com.ibm.etools.webservice.rt.framework.ServiceProvider'
    scope='Request'
    methods='findCityByZipCode insertZipCodeAndCity
       updateCityForZipCode deleteZipCode'>
    <isd:java class='com.ibm.etools.webservice.rt.dxx.DxxService'/>
    <isd:option key='group.name' value='/dxx_travel'/>
    <isd:option key='group.path' value='/travel'/>
    <isd:option key='group.class.name'
           value='com.ibm.etools.webservice.rt.dxx.DxxGroup'/>
   </isd:provider>
 <isd:faultListener>org.apache.soap.server.DOMFaultListener
 </id:faultListener>
 <isd:mappings
    defaultRegistryClass=
    'com.ibm.etools.webservice.rt.dxx.DxxMappingRegistry'/>
</isd:service>
...
...
</dds>
```

Figure 46. Example of part of a dds.xml with an isd file

4. Restart the Web application.

Related concepts:
- "Apache SOAP configurations" on page 135
- "Installing Web applications" on page 131
- "Defining a group of Web services" on page 58

Related tasks:
- "Testing the Web service" on page 109

Apache SOAP configurations

If you use the Apache SOAP engine, you can configure your Web application to use either the Apache configuration manager, which is the default, or the IBM® configuration manager (XMLDrivenConfigManager). The Apache configuration manager stores the deployed services in a serialized Java™ file named DeployedServices.ds. XMLDrivenConfigManager uses an XML format instead and disables the automatic deployment of Web services when adding new DADX files, so that you must deploy them manually (see Generating deployment descriptors).

To help you deploy your sample applications, WORF provides a configuration file (*dds-example.xml*) in the *services.war* file that is shipped with the WORF engine. The *dds-example.xml* is a deployment descriptor file that you can use with all of the DADX files in the examples. The sample *dds.xml* file deploys a few services. If you invoke a service that is not deployed, the server will report that the service is unknown.

If you want to use the IBM configuration manager, rename *soap-ibm.xml* to *soap.xml* and *dds-example.xml* to *dds.xml*. Restart the application server to use the XMLDrivenConfigManager in the installedApps\servicesApp.ear\services.war subdirectory.

Related concepts:

- "Overview of the Web services process" on page 32

Related tasks:

- "Generating deployment descriptors" on page 133

Preparing and creating the Web archive file

To create a Web archive (WAR) file, do the following:

Procedure:

1. Create the basic directory structure for the WAR file as in the following example:

   ```
   WEB-INF\lib\worf-servlets.jar
   WEB-INF\web.xml

   files from the worf\ that you downloaded
   ```

   You can find this WORF directory hierarchy in the apache-services.war file or the axis-services.war file. You use these files when you run the TEST page. The files in the *worf* subdirectory are not necessary if you do not plan to use the built-in test facility of WORF. The worf-servlets.jar file is in the *lib* subdirectory where WORF is installed. The web.xml is the standard J2EE web.xml. An empty web.xml would look like the example in Figure 47.

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

Figure 47. 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*).
   c. Optional: Generate the deployment descriptor (you can skip this step for OS/390 and z/OS platforms). This step is needed only for the IBM configuration manager. In the following example, the first instruction changes the current directory to the WEB-INF subdirectory on a Windows platform. Then, you run the application. When the application completes, you can 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.dadx
cd ..
```

Note: ivt.dadx is a specific sample that is shipped; you might not have this file in your new WAR file

d. Optional: Create or modify file *dds.xml* to add the content of the generated descriptor (skip this step for the OS/390 and z/OS platforms). This step is needed only for the IBM configuration manager. An empty dds.xml file looks like the following:

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

3. Create the WAR file with either of the following methods:
   - 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, the URL for your TEST might look similar to the following:
   ```
   http://<your WebAppServer>/myContext/myURLPath/ivt.dadx/TEST
   ```

   Note: ivt.dadx is a specific sample that is shipped; you might not have this file in your new WAR file

Related concepts:
- "Defining a group of Web services" on page 58

Related tasks:
- "Customizing the group.properties file" on page 64
- "Defining the web.xml and group.properties files in the iSeries platform" on page 62

| ## Web services provider tracing

| After you deploy your Web service, you might need to get information about
| run-time events and diagnostics from the Web service provider. To debug and
| troubleshoot your Web services application, DB2® Web services uses the trace
| facility of the Web application server on which your application runs. The trace
| information that you receive from the Web application server includes messages
| and event activity.

| The DB2 Web services provider supports two tracing systems:

| **log4j** Jakarta-log4j-1.2.8 and commons-logging-1.0.3 on Apache Jakarta Tomcat
| 4.0.6 and later

| **JRas** The tracing and logging system that is used by WebSphere® Application
| Server, Version 5 and later

With these tracing systems you can incorporate message logging and trace facilities into your Java™ applications.

The output that is generated from a trace that you enable within your application appears in the root directory of the Web application server that you use. Table 17 shows the location of the output log file:

Table 17. Trace output location

Server	Output log file location
WebSphere Application Server	${SERVER_LOG_ROOT}/trace.log
Apache Jakarta Tomcat	<tomcat>/logs/worf_log4j.log

All trace messages and events are identified by the operation name of the Web service, the name of the servlet, or the name of the DADX file.

DB2 Web services provider can trace the following types of events:

Informational messages
Messages that indicate when a Web service request event or a Web service response event completes successfully, such as when a DADX file is parsed successfully.

Warning messages
Messages that indicate when a warning condition is detected during processing of the Web service request or the Web service response, such as a warning message from the XML parser for a DADX file.

Error messages
Messages that indicate when an error is detected during the processing of the Web service request or the Web service response, such as when the application produces an exception.

Trace events
Events that indicate when the application enters or exits a method, an exception, a call stack, or value of a variable.

Related concepts:
- “Web services provider features” on page 30

Related tasks:
- “Enabling tracing for the DB2 Web services provider-Apache Tomcat Version 4.0 or later Web application server” on page 139
- “Enabling tracing for the DB2 Web services provider–WebSphere application server” on page 139
- “Enabling tracing for the DB2 Web services provider-WebSphere Studio Application Developer” on page 142

Related reference:
- “Troubleshooting Web services” on page 57

Enabling tracing for the DB2 Web services provider-Apache Tomcat Version 4.0 or later Web application server

You can configure the Apache Tomcat server to trace your DB2 Web services.

Prerequisites:

You need authorization to modify the configuration of the server that you use.

Procedure:

To modify the default `log4j` tracing for the DB2 Web services provider:

1. Create a configuration file with the name `log4j.configuration`, with entries such as in the following example:

   ```
   log4j.rootCategory=DEBUG, console, rollingFile
   log4j.logger.com.ibm.etools.rt.webservice.*=INFO
   log4j.appender.console=org.apache.log4j.ConsoleAppender
   log4j.appender.console.layout=org.apache.log4j.PatternLayout
   log4j.appender.console.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
   log4j.appender.rollingFile=org.apache.log4j.RollingFileAppender");
   log4j.appender.rollingFile.File=<servletContext>\..\..\logs\worf_log4j.log
   log4j.appender.rollingFile.MaxFileSize=100KB
   log4j.appender.rollingFile.layout=org.apache.log4j.TTCCLayout
   log4j.appender.rollingFile.layout.layout.ConversionPattern=%p %t %c - %m%n
   ```
2. Modify the settings in the configuration file to display only certain types of messages:

Table 18. Message settings for log4j configuration file

Message type	**Configuration setting**
log4j warning messages or higher	log4j.logger.com.ibm.etools.webservice.*
log4j informational messages	log4j.logger.com.ibm.etools.webservice.*=INFO
log4j error messages	log4j.logger.com.ibm.etools.webservice.*=ERROR

3. Place the configuration file, `log4j.configuration`, in the WEB-INF/classes directory of the Web application.

You can see a log of your trace events at <installed Web server location>\AppServer\logs\<local server name>.

Related concepts:
- “Web services provider features” on page 30
- “Web services provider tracing” on page 137

Related tasks:
- “Installing and deploying the WORF examples on Apache Jakarta Tomcat” on page 52

Enabling tracing for the DB2 Web services provider–WebSphere application server

You can configure the WebSphere application server to trace the DB2 Web services from the administrative console.

Prerequisites:

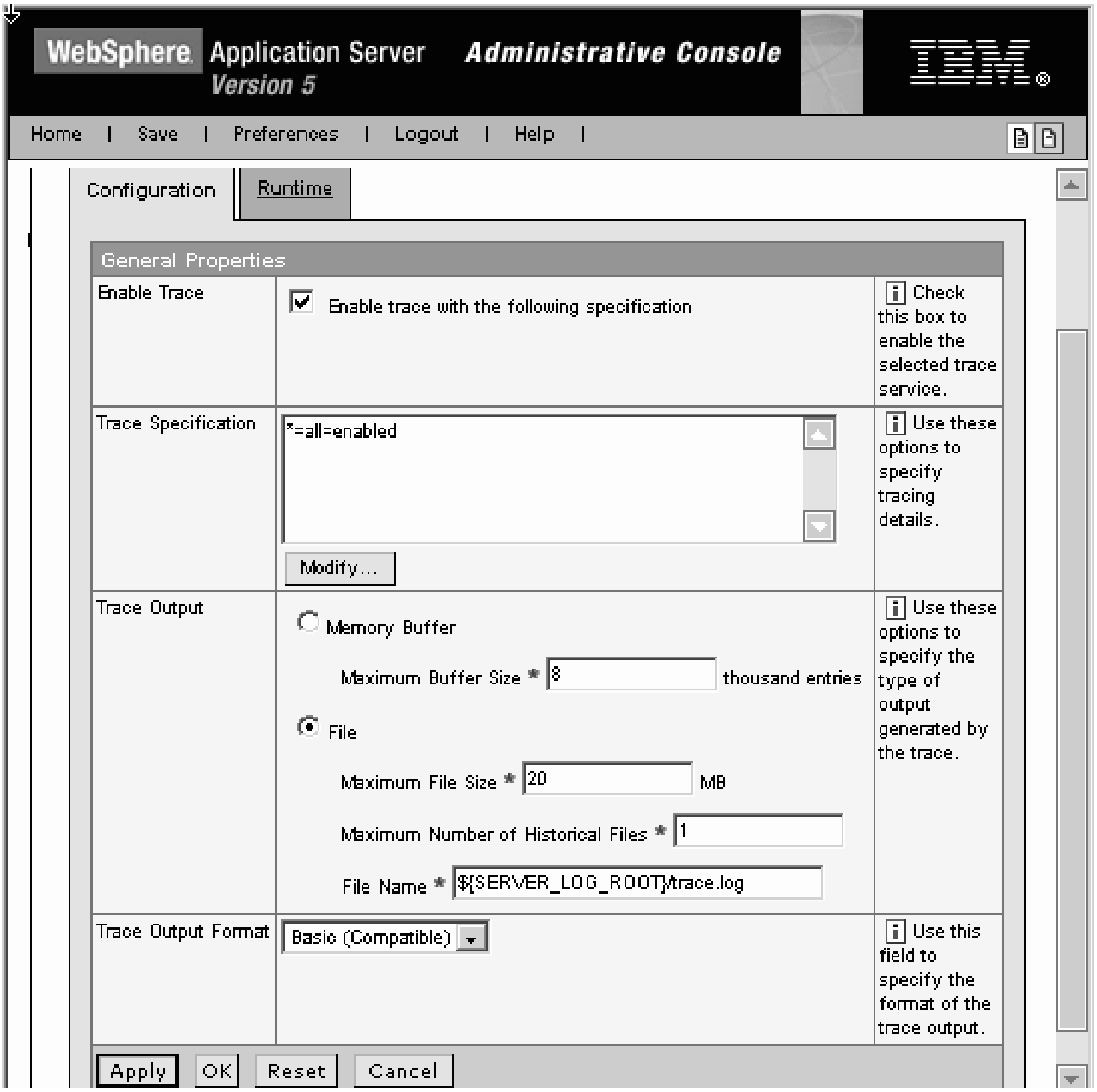
You need authorization to modify the configuration of the server that you use.

Procedure:

To modify the default jRAS tracing for the DB2 Web services provider:

1. Start the WebSphere application server administrative console.
2. In the Navigation tree, click **Troubleshooting —> Log and Trace**. The Logging and Tracing window opens.
3. In the Logging and Tracing window, click the server name and then select **Diagnostic Trace**.
4. In the **Trace Specification** field, type the trace string:
   ```
   com.ibm.etools.webservice.*=all=enabled
   ```
5. If the server is stopped, go to the Configuration page. If the server is running, go to the Runtime page.
   - Optional: From the Runtime page, select the **Save trace** check box to write your changes to the server configuration. If the **Save trace** check box is cleared, the changes that you make apply only for the life of the server process that is currently running.
   - Optional: From the Configuration page, select the **Enable Trace** check box.

   An example of enabling the trace when the server is not running is in Figure 48 on page 141.


WebSphere Application Server Version 5
Administrative Console
IBM
Home | Save | Preferences | Logout | Help |
Configuration
Runtime
General Properties

Enable Trace	✓ Enable trace with the following specification	Check this box to enable the selected trace service.
Trace Specification	*=all=enabled Modify...	Use these options to specify tracing details.
Trace Output	○ Memory Buffer Maximum Buffer Size * 8 thousand entries ● File Maximum File Size * 20 MB Maximum Number of Historical Files * 1 File Name * ${SERVER_LOG_ROOT}/trace.log	Use these options to specify the type of output generated by the trace.
Trace Output Format	Basic (Compatible)	Use this field to specify the format of the trace output.

Figure 48. Enabling the Web services provider trace

6. Save your changes and restart the server.

You can see a log of your trace events at <installed Web server location>\AppServer\logs\<local server name>.

Related concepts:

- “Web services provider features” on page 30
- “Web services provider tracing” on page 137

Related tasks:

- “Enabling tracing for the DB2 Web services provider-WebSphere Studio Application Developer” on page 142

Enabling tracing for the DB2 Web services provider-WebSphere Studio Application Developer

You can configure the WebSphere Studio to trace the DB2 Web services from the administrative console.

Prerequisites:

You need authorization to modify the configuration of the server that you use.

Procedure:

To modify the default jRAS tracing for the DB2 Web services provider:
1. Start the WebSphere Studio administrative console.
2. From the main menu, click **Window** —> **Show View** —> **Server Configuration** to open the Server Configuration view
3. On the **Servers** menu, double-click **WebSphere v5.0 Test Environment** to open the server editor.
4. Go to the Trace page.
5. In the **Trace Specification** field, type the following trace string:
   ```
   com.ibm.etools.webservice.*=all=enabled
   ```
6. Select the **Enable Trace** check box.
7. Save your changes and restart the server.

You can see a log of your trace events at <installed Web server location>\AppServer\logs\<local server name>.

Related concepts:
- "Web services provider features" on page 30
- "Web services provider tracing" on page 137

Related tasks:
- "Enabling tracing for the DB2 Web services provider–WebSphere application server" on page 139

Publishing your Web services

Web service providers publish their Web services so that clients can access them using simple object access protocol (SOAP) over Hypertext Transfer Protocol (HTTP). This contrasts with Enterprise Java™ Bean (EJB) clients, who access beans by using remote method invocation (RMI) over Internet Inter-Orb Protocol (IIOP). Web services process requests from Web clients, invoking the appropriate business function and typically returning a response. The Web service description language (WSDL) document describes the Web service. You store the WSDL in a repository (such as a UDDI registry) or on the server of the Web service provider. Storing the Web service description in an appropriate repository offers the potential for interested customers to discover its existence, potentially generating new business for the Web service provider.

Related concepts:
- "Introduction to using DB2 as a Web services provider – WORF" on page 25

Related tasks:

- "Testing the Web service" on page 109

Installing and using the Web services consumer

DB2 Universal Database can optimize access to Web service providers as a consumer. By using SQL statements, you can consume and integrate Web services data. By using SQL to access Web services data, you save effort because you can manipulate the data within the context of an SQL statement. Then you can return the statement to the client application. The Web services consumer set of tools helps access Web services data from SQL. The Web services consumer converts existing WSDL interfaces into DB2 table or scalar functions. This section describes the Web services consumer standalone tool and the WebSphere Studio plug-in that IBM provides to convert WSDL to DB2 SQL functions.

Installation of the Web services consumer user-defined functions

Prerequisites:

The Web services consumer user-defined functions (UDFs) are available on the following platforms (all platforms are 32 bit):

- Windows 2000
- Linux
- AIX
- Solaris Operating Environment (with DB2 for Universal Database Version 8, Fix Pack 2)

You should install the following software before executing the SOAP UDFs:

- DB2 Universal Database
  - Version 8 which includes Xerces parser and XML Extender
- Optional: WebSphere Studio Application Developer (WSAD) Version 5.1.1

  The plug-in requires WebSphere Studio to generate Web service UDFs from WSDL. You can invoke the Web service consumer directly, and you can develop your own SQL functions.

You must also enable DB2 XML Extender with the **dxxadm enable_db sample** command. See the*DB2 XML Extender Administration and Programming* for more options on the DB2 XML Extender commands.

Procedure:

To enable, or install, and disable the Web service consumer:

1. Execute the following utility to register five user-defined functions:

   ```
   db2enable_soap_udf  -n dbName [-u uID] [-p password] [-force]
   ```

   The parameters have the following definitions:

   dbName
   A database name

   uID Optional: A user ID

   password
   Optional: The password associated with the user ID

–force Attempts to drop any existing functions.

The enable command enables your database to use the SOAP requester functions.

2. When you disable the Web service consumer, you drop the functions. Execute the following utility:

```
db2disable_soap_udf  -n dbName [-u uID] [-p password]
```

The meanings of the parameters are the same as for the enable utility described above.

3. You can also create and delete the user-defined functions by using the DB2 CLP (command line processor).

```
CREATE SCHEMA db2xml;

   CREATE FUNCTION db2xml.soaphttpv (
               endpoint_url VARCHAR(256),
               soap_action VARCHAR(256),
               soap_body VARCHAR(3072))
         RETURNS VARCHAR(3072)
      LANGUAGE C PARAMETER STYLE DB2SQL
      SPECIFIC soaphttpvivo EXTERNAL NAME 'db2soapudf!soaphttpvivo'
      SCRATCHPAD FINAL CALL FENCED
      NOT DETERMINISTIC CALLED ON NULL INPUT
      NO SQL EXTERNAL ACTION DBINFO;

   CREATE FUNCTION db2xml.soaphttpv (
               endpoint_url VARCHAR(256),
               soapaction VARCHAR(256),
               input_message CLOB(1M))
         RETURNS VARCHAR(3072)
      LANGUAGE C PARAMETER STYLE DB2SQL
      SPECIFIC soaphttpcivo EXTERNAL NAME 'db2soapudf!soaphttpcivo'
      SCRATCHPAD FINAL CALL FENCED
      NOT DETERMINISTIC CALLED ON NULL INPUT
      NO SQL EXTERNAL ACTION DBINFO;

   CREATE FUNCTION db2xml.soaphttpc (
               endpoint_url VARCHAR(256),
               soapaction VARCHAR(256),
               input_message CLOB(1M))
         RETURNS clob(1M)
      LANGUAGE C PARAMETER STYLE DB2SQL
      SPECIFIC soaphttpcico EXTERNAL NAME 'db2soapudf!soaphttpcico'
      SCRATCHPAD FINAL CALL FENCED
      NOT DETERMINISTIC CALLED ON NULL INPUT
      NO SQL EXTERNAL ACTION DBINFO;

   CREATE FUNCTION db2xml.soaphttpc (
               endpoint_url VARCHAR(256),
               soapaction VARCHAR(256),
               soap_body varchar(3072))
         RETURNS clob(1M)
      LANGUAGE C PARAMETER STYLE DB2SQL
      SPECIFIC soaphttpvico EXTERNAL NAME 'db2soapudf!soaphttpvico'
      SCRATCHPAD FINAL CALL FENCED
      NOT DETERMINISTIC CALLED ON NULL INPUT
      NO SQL EXTERNAL ACTION DBINFO;

   CREATE FUNCTION db2xml.soaphttpcl (
               endpoint_url VARCHAR(256),
               soapaction VARCHAR(256),
               soap_body varchar(3072))
         RETURNS CLOB(1M) as locator
      LANGUAGE C PARAMETER STYLE DB2SQL
```

```
SPECIFIC soaphttpviclo EXTERNAL NAME 'db2soapudf!soaphttpviclo'
SCRATCHPAD FINAL CALL NOT FENCED
NOT DETERMINISTIC CALLED ON NULL INPUT
NO SQL EXTERNAL ACTION DBINFO;
```

4. The Web service consumer WebSphere Studio plug-in is a component of WebSphere Studio Application Developer (WSAD) Version 5.1.1.

Related concepts:
- "Web services consumer user-defined functions" on page 146

Related reference:
- "Using the Web services consumer UDFs" on page 159

The Web service consumer functions

IBM® DB2® Information Integrator and DB2 Universal Database™ extend the functions of DB2 Universal Database and Web services with the ability to invoke Web services from within Structured Query Language (SQL) statements. You do this by invoking a set of user-defined functions (UDFs) that provide a high-speed client simple object access protocol (SOAP) over Hypertext Transfer Protocol (HTTP) interface to accessible Web services. You can call these functions directly from SQL statements.

You can construct the SOAP body according to the Web services description language (WSDL) of a Web service. You can also use the Web service User-Defined Function (UDF) tool in WebSphere® Studio Application Developer (WSAD) to automatically generate specific UDFs. These UDFs can invoke operations that are defined by a user-specified Web services description language file. The generated UDFs are DB2 UDB functions that do the following:
- Provide the parameters for the Web service request.
- Invoke the SOAP client functions.
- Map the result of the Web service invocation to the return types specified by the user.

On some networks, access to the internet must go through a firewall. The traffic might be restricted to certain machines and certain ports that are allowed to send network traffic. Some systems allow applications to tunnel through the firewall. The SOAP UDFs support tunneling with SOCKS clients and HTTP proxies. To use a SOCKS server to tunnel through the firewall you must install SOCKS client software on your system. To use HTTP proxies you must set two environment variables for configuring to DB2 Universal Database. Set `DB2SOAP_PROXY` to include the host name of the machine with the HTTP proxy. Set `DB2SOAP_PORT` to the port of the HTTP proxy, such as 8080. In both cases the SOAP traffic goes through the system that tunnels the firewall.

You can test the sample applications that are shipped with DB2 Information Integrator with the following steps:
1. Start the database manager (with the db2start command).
2. Create the "sample" database (with the db2sampl command).
3. Establish a connection with the "sample" database.

4. Invoke the example files (in a Windows® environment, these example files are in <DB2 UDB installed path>\samples\soap) with the following command: *db2 -vf filename -t*.

Related tasks:

- "Declaring registry and environment variables" in the *Administration Guide: Implementation*
- "Installation of the Web services consumer user-defined functions" on page 143

Related reference:

- "Using the Web services consumer UDFs" on page 159

Web services consumer user-defined functions

Simple Object Access Protocol (SOAP) is an Extensible Markup Language (XML) protocol consisting of the following characteristics:

- An envelope that defines a framework for describing the contents of a message and how to process the message
- A set of encoding rules for expressing instances of application-defined data types
- A convention for representing SOAP requests and responses

DB2® Universal Database needs the following information to build a SOAP request and receive a SOAP response.

- A service endpoint, for example, *http://services.xmethods.net/soap/servlet/rpcrouter*
- Some Extensible Markup Language (XML) content of the SOAP body, which includes the name of an operation with requested namespace URI, an encoding style, and input arguments.
- Optional: A SOAP action URI reference. The reference can be empty, as shown in the following example, *http://tempuri.org/* or just ''.

The DB2 UDB function db2xml.soaphttp() does the following actions:

1. It composes a SOAP request
2. It posts the request to the service endpoint
3. It receives the SOAP response
4. It returns the content of the SOAP body

This is an overloaded function that is used for VARCHAR() or CLOB(), depending on the SOAP body.

```
db2xml.soaphttpv returns VARCHAR():
    db2xml.soaphttpv (endpoint_url VARCHAR(256),
                      soap_action VARCHAR(256),
                      soap_body VARCHAR(3072))
                  RETURNS VARCHAR(3072)
db2xml.soaphttpv returns VARCHAR():
    db2xml.soaphttpv (endpoint_url VARCHAR(256),
                      soap_action VARCHAR(256),
                      soap_body CLOB(1M))
                  RETURNS VARCHAR(3072)


db2xml.soaphttpc returns CLOB():
       db2xml.soaphttpc (endpoint_url VARCHAR(256),
                        soapaction VARCHAR(256),
                        soap_body VARCHAR(3072))
                      RETURNS CLOB(1M)
db2xml.soaphttpc returns CLOB():
       db2xml.soaphttpc (endpoint_url VARCHAR(256),
                      soapaction VARCHAR(256),
                      soap_body CLOB(1M))
                    RETURNS CLOB(1M)

db2xml.soaphttpcl returns CLOB() as locator:
       db2xml.soaphttpcl(endpoint_url VARCHAR(256),
                       soapaction VARCHAR(256),
                       soap_body varchar(3072))
                    RETURNS CLOB(1M) as locator
```

Example of a DB2 UDB constructed SOAP request envelope

The example in Figure 49 on page 147 shows an Hypertext Transfer Protocol (HTTP) post header to post a SOAP request envelope to a host. The bold areas show the web service endpoint (post path and host) and the content of the SOAP body. The SOAP body shows a temperature request for zip code 95120.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: services.xmethods.net
Connection: Keep-Alive User-Agent: DB2SOAP/1.0
Content-Type: text/xml; charset="UTF-8"
SOAPAction: ""
Content-Length: 410

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
                   xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
                   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
                   xmlns:xsd=http://www.w3.org/2001/XMLSchema >
   <SOAP-ENV:Body>
       <ns:getTemp xmlns:ns="urn:xmethods-Temperature">
         <zipcode>95120</zipcode>

       </ns:getTemp>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 49. A DB2 UDB constructed SOAP request envelope

Example of using DB2 UDB to extract the content of the SOAP response envelope

The example in Figure 50 on page 148 shows the HTTP response header with the SOAP response envelope. The **bold** content of the SOAP body shows the result of the temperature request. The namespace definitions from the SOAP envelope are not shown here, but they would also be included.

```
HTTP/1.1 200 OK
Date: Wed, 31 Jul 2002 22:06:41 GMT
Server: Enhydra-MultiServer/3.5.2
Status: 200
Content-Type: text/xml; charset=utf-8
Servlet-Engine: Lutris Enhydra Application Server/3.5.2
  (JSP 1.1; Servlet 2.2; Java™ 1.3.1_04;
   Linux 2.4.7-10smp i386; java.vendor=Sun Microsystems Inc.)
Content-Length: 467
Set-Cookie:JSESSIONID=JLEcR34rBc2GTIkn-0F51ZDk;Path=/soap
X-Cache: MISS from www.xmethods.net
Keep-Alive: timeout=15, max=10
Connection: Keep-Alive
<?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:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
           SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/ >
              <return xsi:type="xsd:float">85<return>
           </ns1:getTempResponse>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 50. Using DB2 UDB to extract the content of the SOAP response envelope

Related tasks:

- "Installation of the Web services consumer user-defined functions" on page 143

Related reference:

- "Using the Web services consumer UDFs" on page 159

Tracing Web services consumer events

The Web services consumer user-defined function can be traced by using the DB2 Universal Database trace utility. In addition, when you use a Windows platform, you can trace Hypertext Transfer Protocol (HTTP) SOAP requests and responses into a file.

Procedure:

To trace the SOAP component in DB2 Universal Database, use the following trace mask:

```
db2trc on -m *.*.147.*.*
```

Related concepts:

- "The Web service consumer functions" on page 145

Related reference:

- "Using the Web services consumer UDFs" on page 159

Web services consumer—using the WebSphere Studio User-Defined Function tool

The Web services consumer User-Defined Function wizard generates and tests user-defined functions in WebSphere® Studio Version 5.

The wizard used with WebSphere Studio reads the Web Services Definition Language (WSDL) file. It then generates the user-defined functions (UDFs) that provide easy access to Web services from database applications. You can use the generated UDFs in SQL statements to combine relational data with dynamic data that are retrieved from a Web service. You can invoke the Web service consumer functions directly in SQL (see The Web service consumer functions). However, the task can require some advanced programming skills, and it can be time-consuming. After you generate and deploy the UDFs, you can use the functions in SQL to combine relational data with dynamic data that you retrieve from Web services. The generated UDFs are structured as follows:

1. Construct the SOAP body
2. Invoke the SOAP consumer (submit the SOAP request envelope)
3. Extract values from the SOAP response

Related concepts:

- "The Web service consumer functions" on page 145
- "Web services consumer user-defined functions" on page 146

Related tasks:

- "Installation of the Web services consumer user-defined functions" on page 143
- "Tracing Web services consumer events" on page 148

Related reference:

- "Using the Web services consumer UDFs" on page 159

How to generate the user-defined functions from WebSphere Studio

Prerequisites:

1. Enable the DB2 XML Extender database
2. Enable the Web services consumer UDFs for the database
3. Create a project that you want to use with the Web service UDF
4. Create a connection to the database that you just enabled
5. Import the database to your WebSphere Studio, Version 5 project. See *WebSphere Studio Application Developer Programming Guide* for more information.

Procedure:

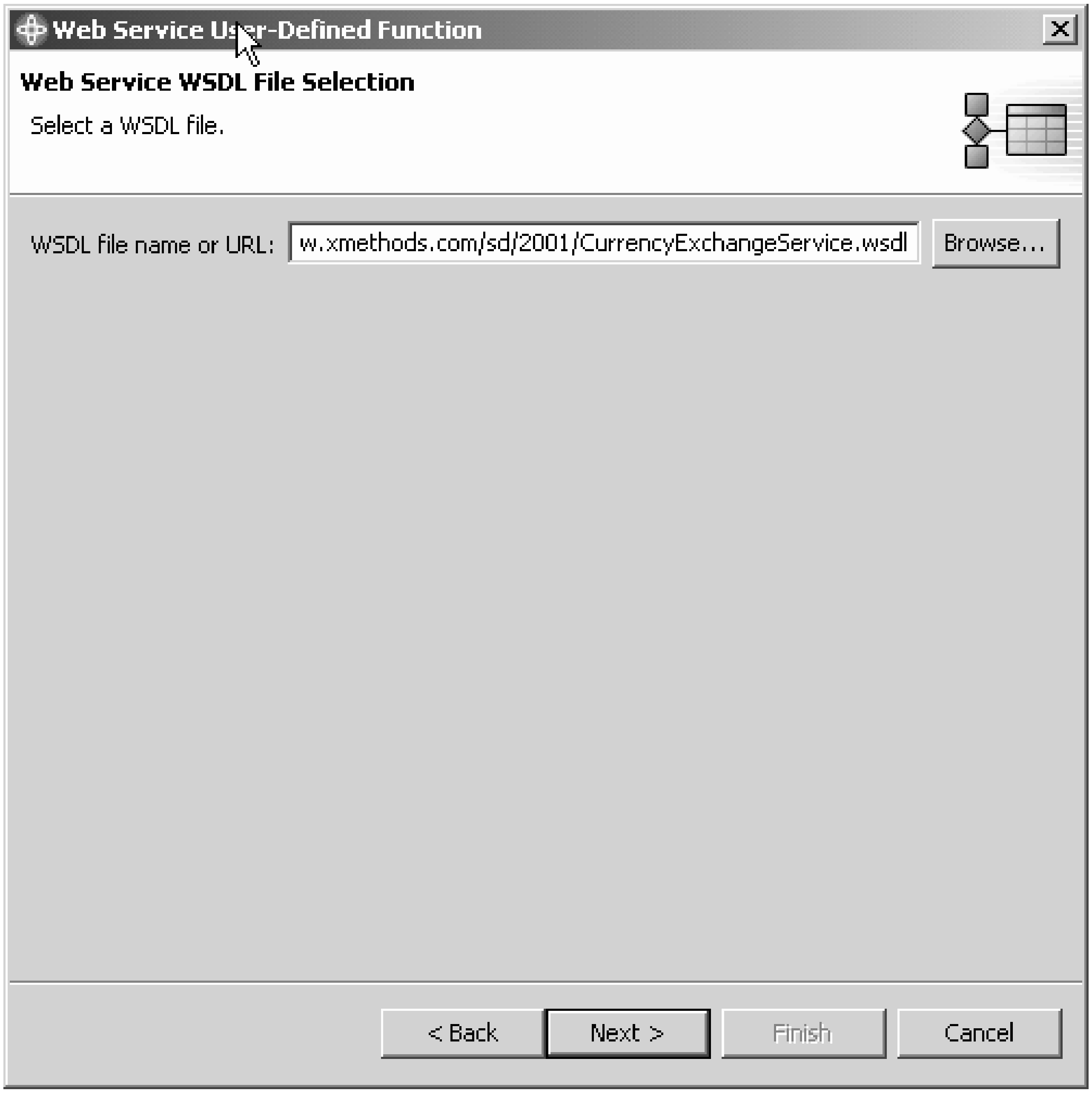
Within the WebSphere Studio, you can launch the wizard that generates the user-defined functions (UDFs) in three different ways.

- You can invoke it from the *File* > *New* > *Other*> menu. Then select **Data**. The folder expands and you select **Web Service User-Defined Function** from the menu. Click the **Next** push button to proceed.
- You can start it in the Web service client wizard where it appears as an option in addition to generating a Java proxy.

- It is an option in the Web service wizard when generating a test client.

Generate the UDFs with the following steps:

1. Specify the WSDL file from the first page of the wizard (see Figure 51 on page 150). You use this WSDL file to generate the UDF.



Web Service User-Defined Function

Web Service WSDL File Selection

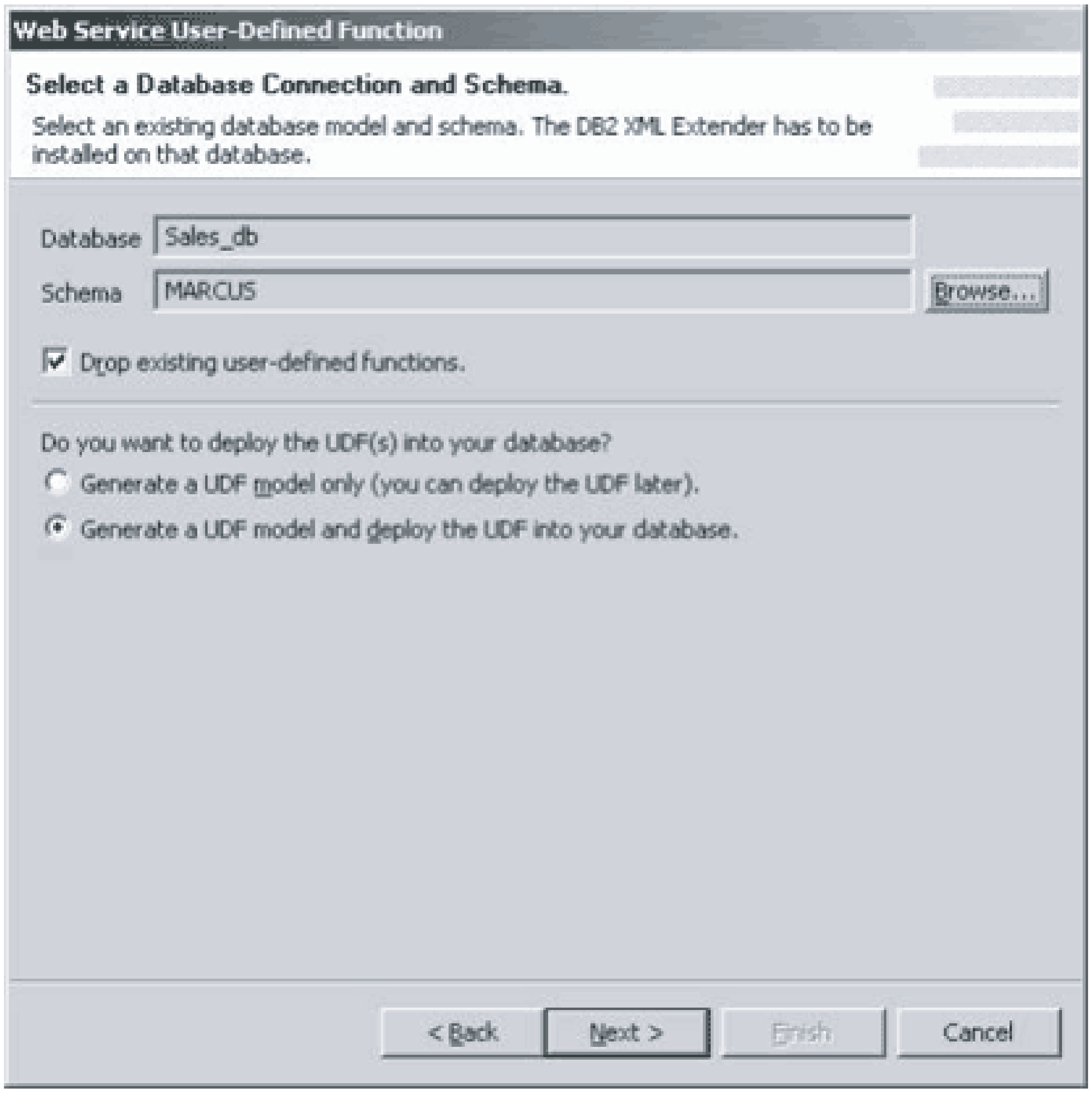
Select a WSDL file.

WSDL file name or URL: w.xmethods.com/sd/2001/CurrencyExchangeService.wsdl

Figure 51. Select the WSDL file

Select a WSDL file from the work space or specify an appropriate uniform resource locator (URL). For example, the currency exchange rate Web service takes two countries as input parameters and returns the currency exchange rate between them. The WSDL file is at *www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl*.

2. Select the database. In Figure 52 on page 151, you see the database connection and a schema for which the UDF is generated. Click the **Browse** push button to select a database schema from the WebSphere Studio work space. The wizard requires that the database is enabled for the Web service consumer UDFs, and the DB2 XML Extender. If there is currently no connection to the specified database available, an additional message window asks for connection information. You can choose to immediately deploy the generated UDF into the database or generate a UDF in the WebSphere Studio work space only. You can deploy the UDF later.


Web Service User-Defined Function

Select a Database Connection and Schema.

Select an existing database model and schema. The DB2 XML Extender has to be installed on that database.

Database: Sales_db

Schema: MARCUS

[✓] Drop existing user-defined functions.

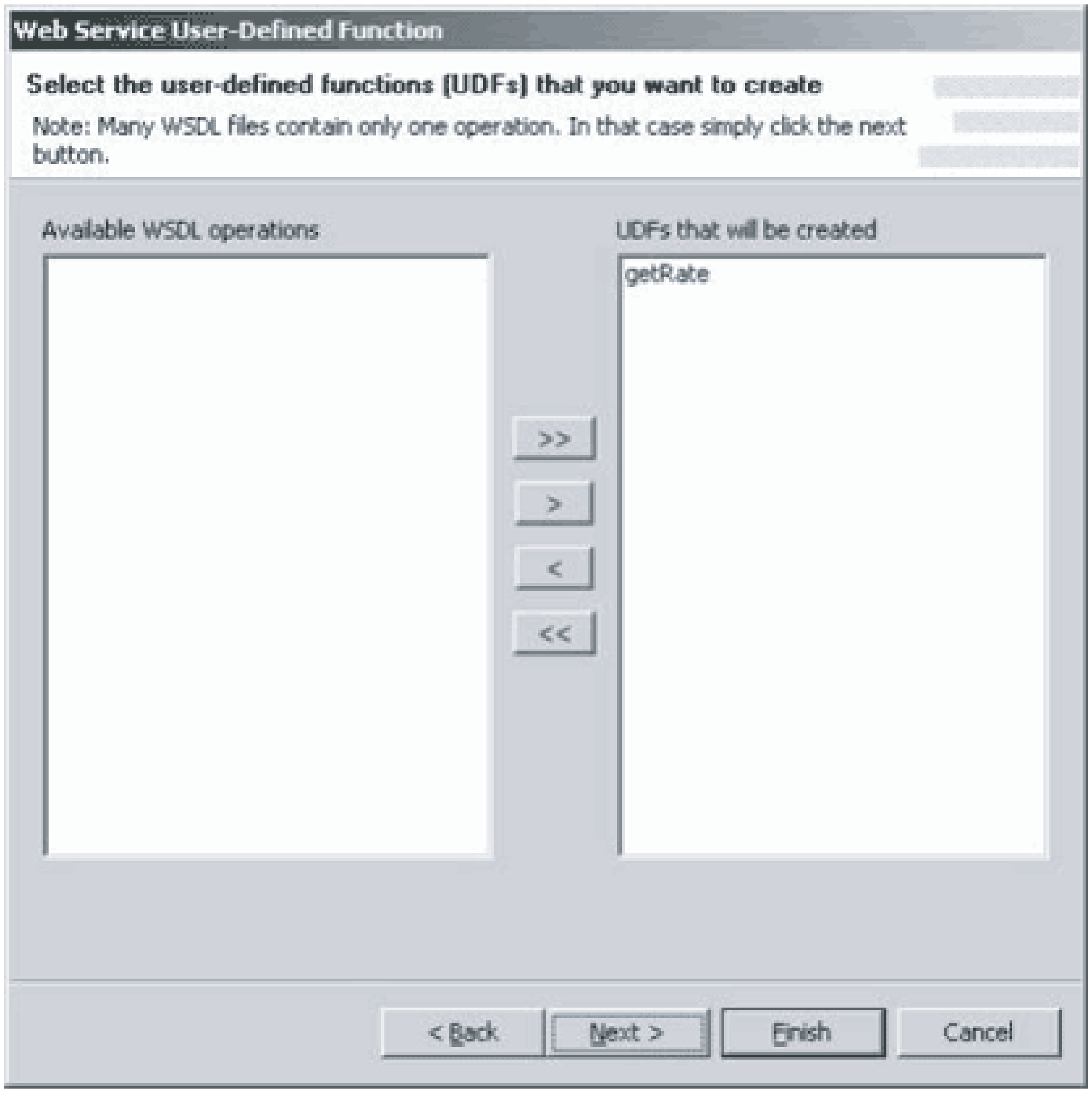
Do you want to deploy the UDF(s) into your database?

( ) Generate a UDF model only (you can deploy the UDF later).

(●) Generate a UDF model and deploy the UDF into your database.

Figure 52. WSDL page 2

3. Select the UDFs that you want to create. From the list of operations that are described in Figure 53 on page 152, select the one that you want to create. The wizard generates one UDF for every operation that is selected. Since the Web service that is used for this example provides only one operation, the wizard selects it automatically. Proceed to the next page.


Web Service User-Defined Function
Available WSDL operations
UDFs that will be created
>>
>
<
<<
< Back
Next >
Finish
Cancel

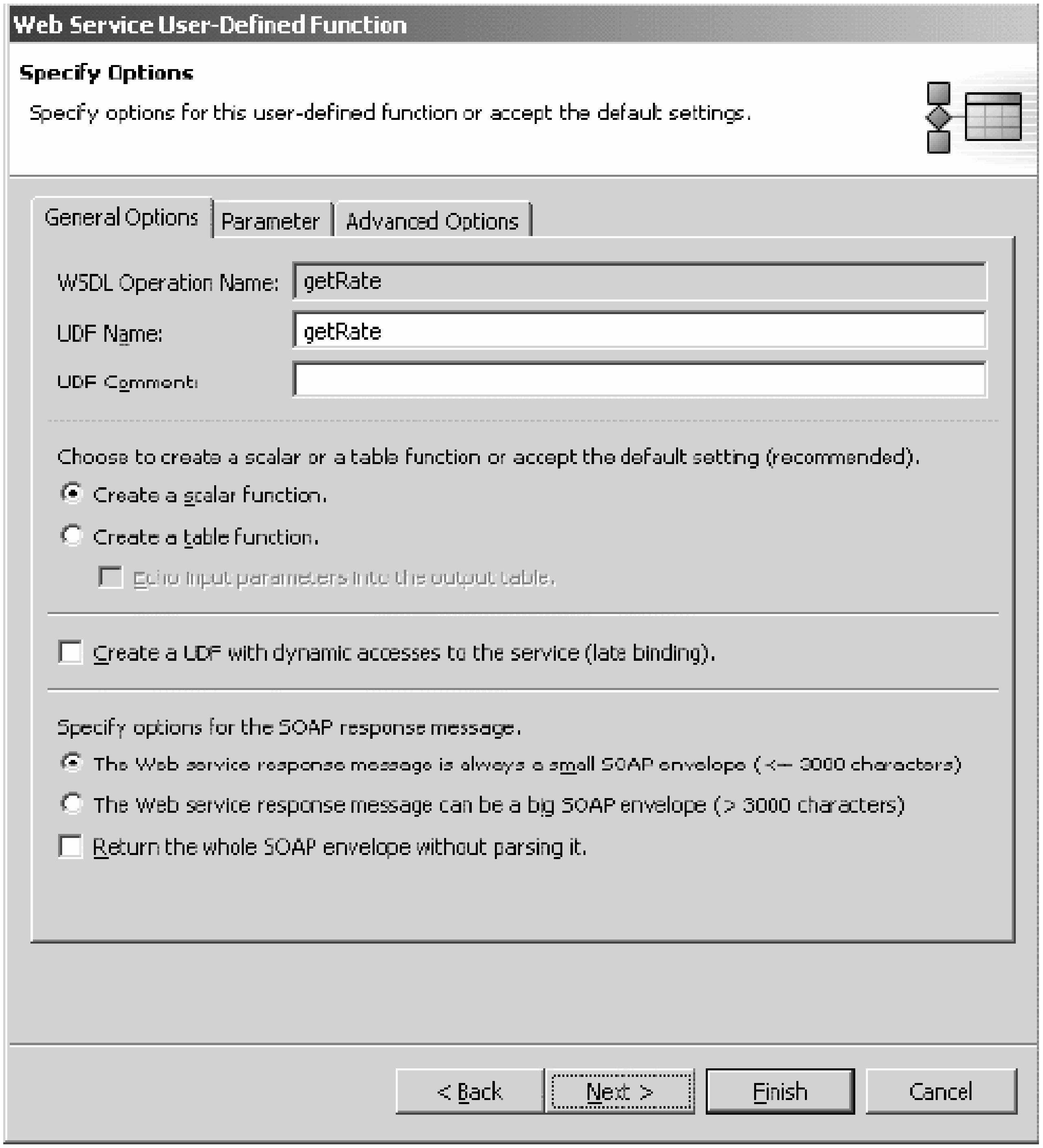
Select the user-defined functions (UDFs) that you want to create

Note: Many WSDL files contain only one operation. In that case simply click the next button.

getRate

Figure 53. Wizard page 3

4. Select options for the UDF. For each operation selected in the previous step, you can define options for those UDFs, such as changing the function name, or providing comments on the function. See Figure 54 on page 153 and Figure 55 on page 154.


Web Service User-Defined Function

Specify Options

Specify options for this user-defined function or accept the default settings.

General Options | Parameter | Advanced Options

WSDL Operation Name: getRate

UDF Name: getRate

UDF Comment:

Choose to create a scalar or a table function or accept the default setting (recommended).

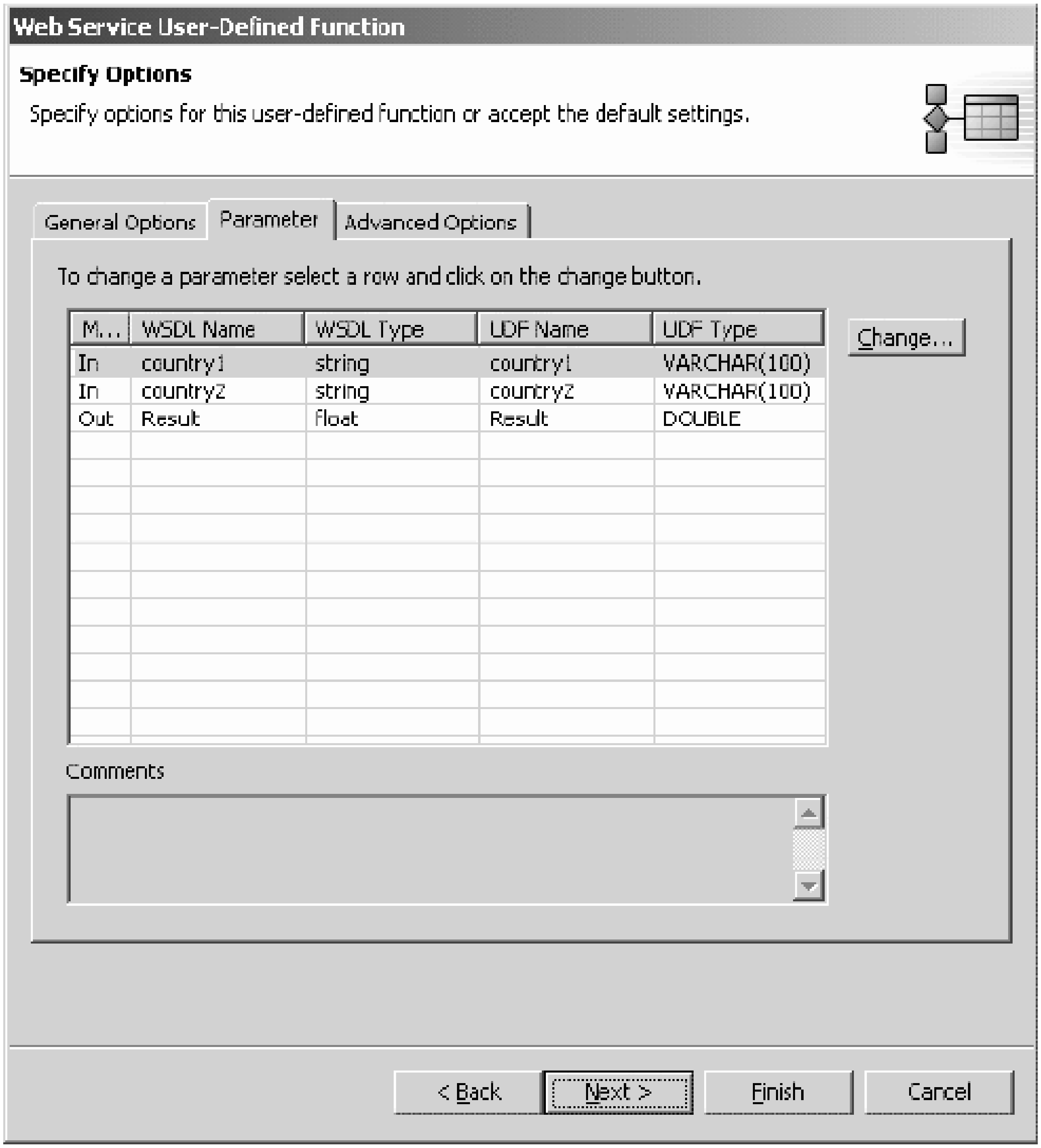
- [✓] Create a scalar function.
- [ ] Create a table function.
  - [ ] Echo input parameters into the output table.

- [ ] Create a UDF with dynamic accesses to the service (late binding).

Specify options for the SOAP response message.

- [✓] The Web service response message is always a small SOAP envelope (<= 3000 characters)
- [ ] The Web service response message can be a big SOAP envelope (> 3000 characters)
- [ ] Return the whole SOAP envelope without parsing it.

Figure 54. Select options page 1


Web Service User-Defined Function

Specify Options

Specify options for this user-defined function or accept the default settings.

General Options | Parameter | Advanced Options

To change a parameter select a row and click on the change button.

M...	WSDL Name	WSDL Type	UDF Name	UDF Type
In	country1	string	country1	VARCHAR(100)
In	country2	string	country2	VARCHAR(100)
Out	Result	float	Result	DOUBLE

Figure 55. Select options page 2

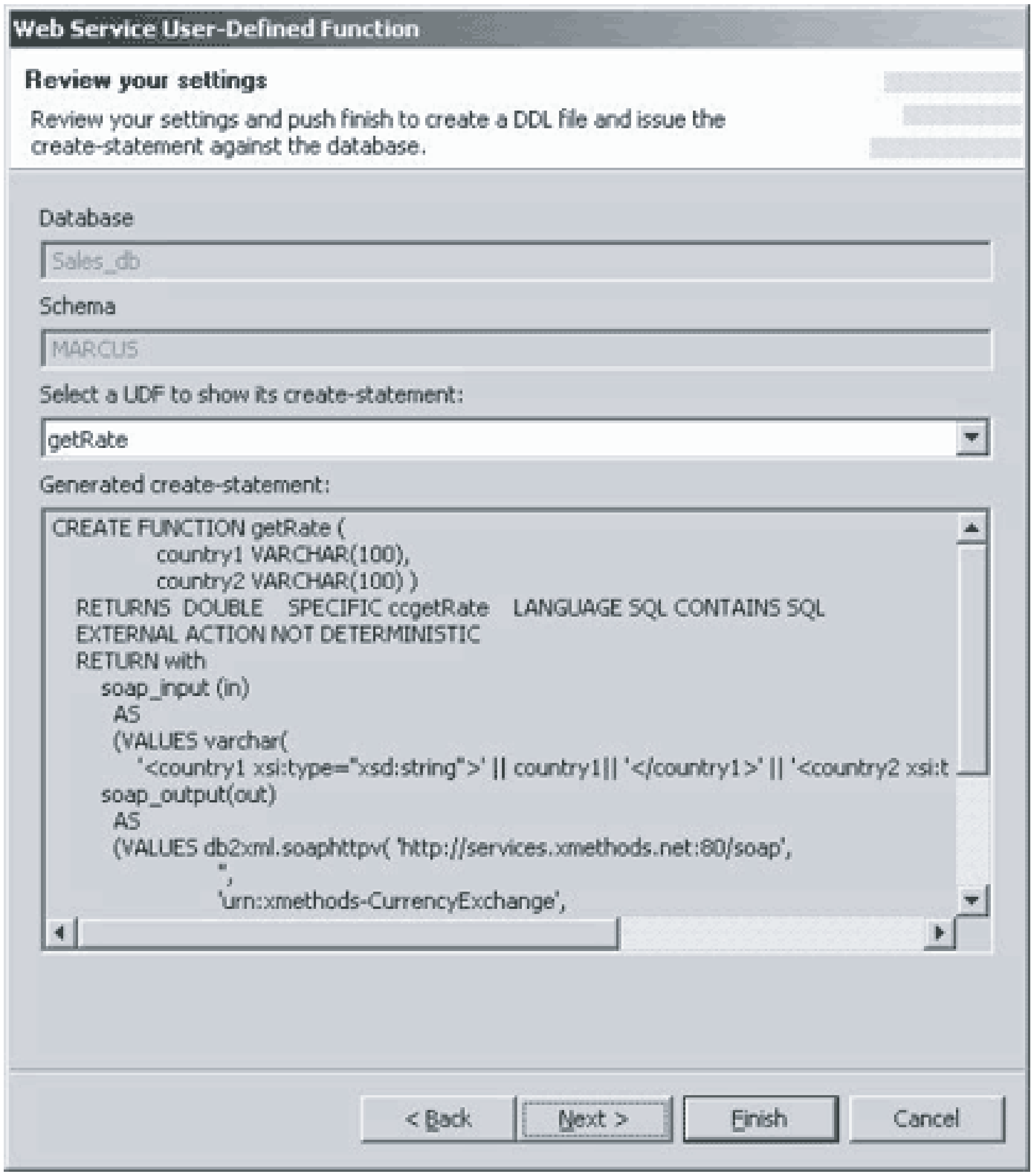
- You can choose to build a scalar or a table function.
  - The wizard generates a scalar function when the Web service returns a simple XML type.
  - The wizard generates a table function when it returns a complex XML type. The table function automatically maps the complex XML type into multiple columns.

  Switching from a table function to a scalar function makes sense when the wizard should not automatically map the returned types. In this case, the wizard should return them as an XML fragment. Being able to switch from a scalar to a table function allows you to use the UDF in a FROM clause.

- You can include the input parameters as columns in the output table by selecting the **Echo the input parameters into the output table** check box.
- You can choose to generate a UDF with dynamic access to the Web service.

  When you do not specify the service location (the location attribute of the soap:address element) in the WSDL document, you generate a dynamic function. You can select the **Create a UDF with dynamic accesses to the service** check box even when the service location is specified to make use of late binding. When you generate a dynamic function, specify the service location at runtime as a parameter of the UDF.
- When using a Web service that can return responses of more than 3000 characters, specify **The Web service response message can be a big SOAP envelope** radio button. By default, **The Web service response message is always a small SOAP envelope** radio button is specified because this results in better performance for most Web services. If you specify the small SOAP response option and the wizard returns a SOAP envelope with more than 3000 characters, the generated Web Service UDF returns a descriptive error message.
- Select the **Return the whole SOAP envelope without parsing it** check box to help in debugging the Web services consumer UDFs.

5. From the Parameter page of the options window, you can review, and change the parameter mappings from WSDL types to SQL types (see Figure 55 on page 154).
6. From the Advanced Options page, you can specify the name for the UDF. If you do not specify a name, then a unique name is automatically generated by the database when you deploy the UDF.
7. Review the settings for generating the UDFs. Examine the database and schema, and the CREATE statement that will be issued on the database (see Figure 56 on page 156).
8. Click the **Next** or **Finish** push button. This generates the UDF and deploys it into the database, because of the earlier selections to generate and deploy.



Review your settings

Review your settings and push finish to create a DDL file and issue the create-statement against the database.

Database: Sales_db

Schema: MARCUS

Select a UDF to show its create-statement: getRate

Generated create-statement:

```
CREATE FUNCTION getRate (
            country1 VARCHAR(100),
            country2 VARCHAR(100) )
   RETURNS  DOUBLE    SPECIFIC ccgetRate    LANGUAGE SQL CONTAINS SQL
   EXTERNAL ACTION NOT DETERMINISTIC
   RETURN with
      soap_input (in)
       AS
       (VALUES varchar(
          '<country1 xsi:type="xsd:string">' || country1|| '</country1>' || '<country2 xsi:t
      soap_output(out)
       AS
       (VALUES db2xml.soaphttpv( 'http://services.xmethods.net:80/soap',
                   '',
                   'urn:xmethods-CurrencyExchange',
```

Figure 56. Review

9. You can run the Web service consumer UDF directly from the work space. To run the deployed UDF:
   a. Right-click on the UDF.
   b. Select **Run** (see Figure 57 on page 157). The Run Settings window opens (Figure 58 on page 158).
   c. From the Run Settings window, you can specify the parameter values.
   d. Click the **OK** push button to see the results of your test (Figure 59 on page 158).

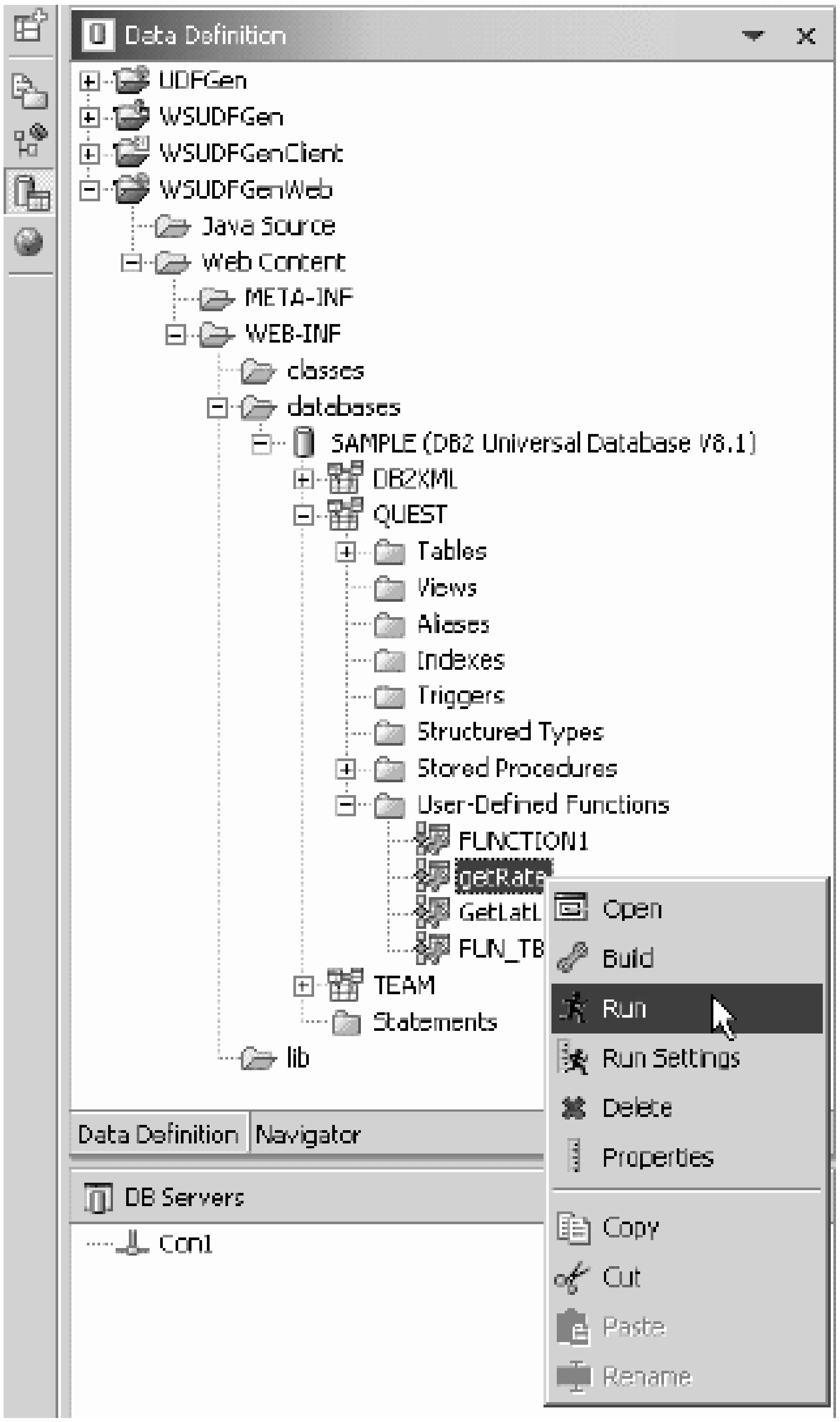
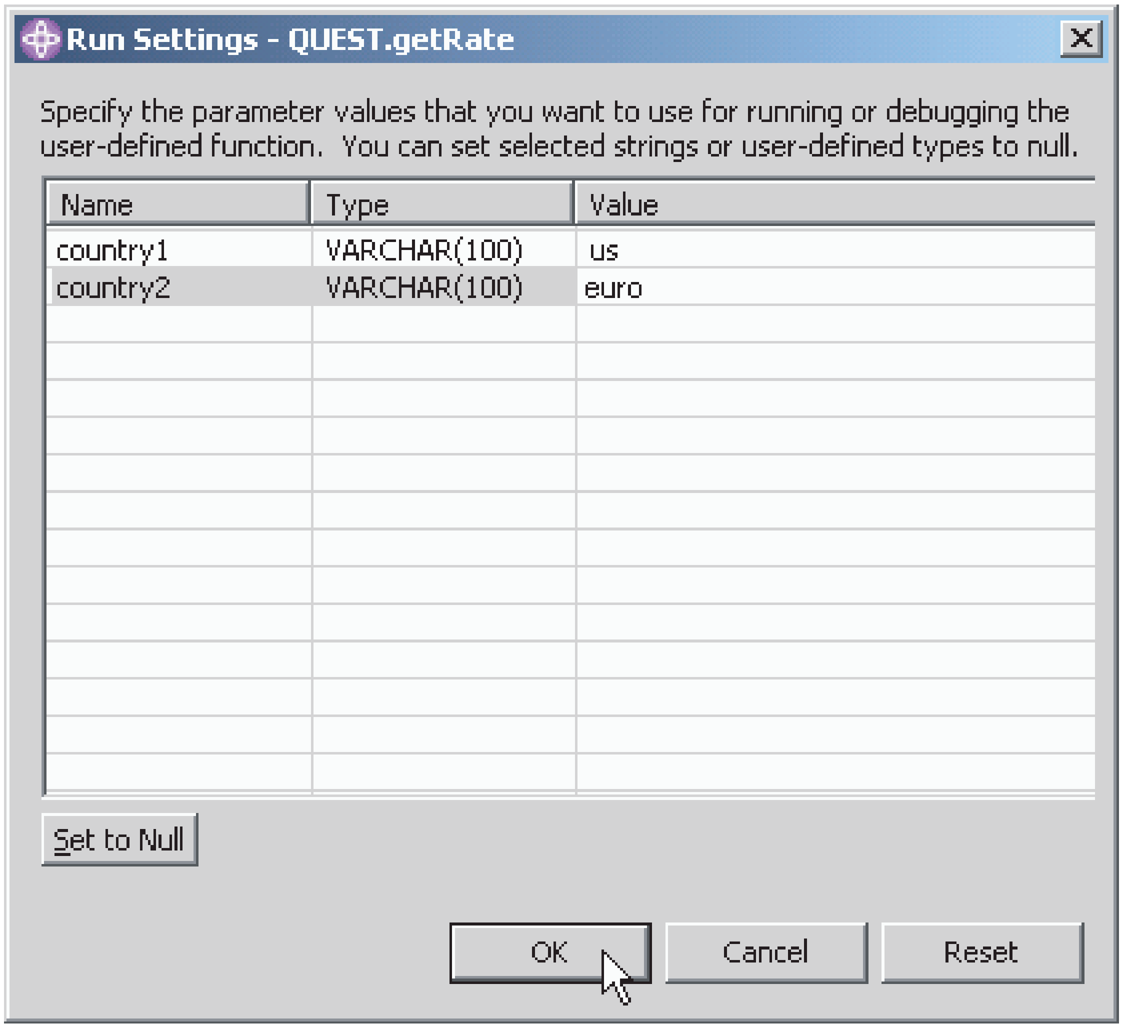


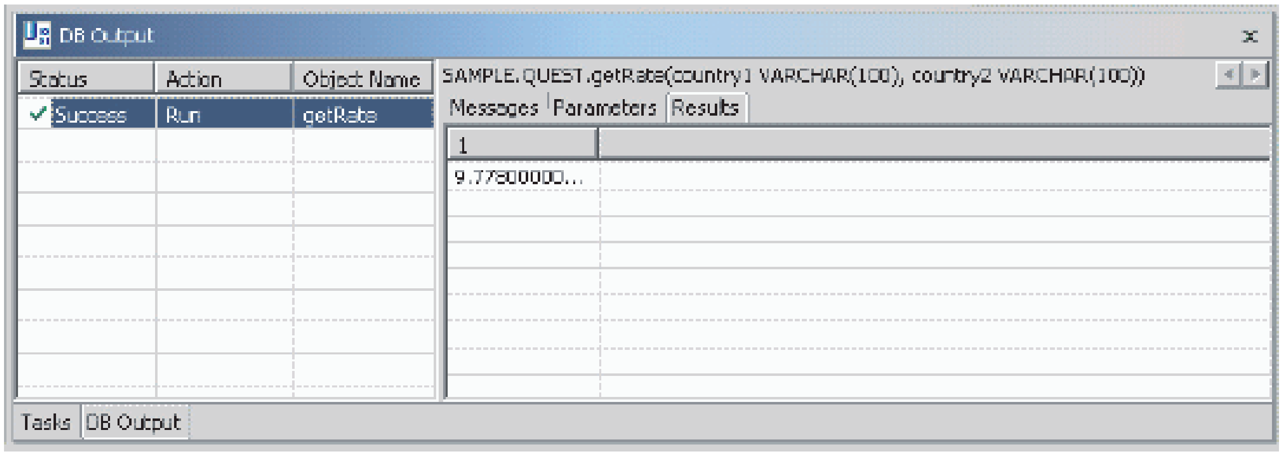
Figure 57. Test



Specify the parameter values that you want to use for running or debugging the user-defined function. You can set selected strings or user-defined types to null.

Name	Type	Value
country1	VARCHAR(100)	us
country2	VARCHAR(100)	euro

Figure 58. Run Settings



Status	Action	Object Name
✓ Success	Run	getRate

SAMPLE.QUEST.getRate(country1 VARCHAR(100), country2 VARCHAR(100))

1
9.77800000...

Figure 59. Results

Related concepts:

- "The Web service consumer functions" on page 145

- “Web services consumer user-defined functions” on page 146
- “Web services consumer—using the WebSphere Studio User-Defined Function tool” on page 149

Related tasks:

- “Installation of the Web services consumer user-defined functions” on page 143

Related reference:

- “Using the Web services consumer UDFs” on page 159
- “dxxEnableDB() stored procedure” in the *DB2 XML Extender Administration and Programming*

Using the Web services consumer UDFs

Use the UDFs to share information between your relational tables and your Web services. Assume that there is a table in a relational database with the following data:

Table 19. Products table

Product	Price
Gear	950.00
Nut	25.00
Bolt	35.00

And assume that there is information about currency types in a remote table.

Table 20. Currency table

Area
US
EURO
UK

Use the following SQL statement on page 159 to determine how you can use the currency exchange rate function to display price information in Euros instead of US dollars. This accesses real-time exchange rates. Note that the statement uses a built-in decimal function to cast the price information.

```
SELECT product, decimal(getRate('us', 'euro') * price, 10, 2)
  as 'EUR_Price'
  FROM products
```

The result of the statement on page 159 is:

Table 21. Using real-time exchange rates

Product	EUR_Price
Gear	1019.82
Nut	26.84
Bolt	37.57

Use the following SQL statement to show how you can use relational data as input to the Web service. The example on page 160 shows how the currency exchange rate function can display price information in different currencies.

```
SELECT p.product, c.area,
    decimal(getRate('us', c.area) * price, 10, 2)
  as Price
  FROM products, areas
```

The result is:

Table 22. Displaying price information in different currencies

Product	Area	Price
Gear	us	950.00
Nut	us	25.00
Bolt	us	35.00
Gear	euro	1019.82
Nut	euro	26.84
Bolt	euro	37.57
Gear	uk	650.84
Nut	uk	17.12
Bolt	uk	23.97

If you use this query often, you might want to define a view to provide a simpler interface. An example of this view would be:

```
CREATE VIEW prices AS
  SELECT p.product, c.area,
      decimal(getRate('us', c.area) * price, 10, 2)
      as Price
  FROM p.products, c.areas
```

By using the view, you can code the following simpler query:

```
SELECT * FROM prices
```

Use the following SQL statement to show how you can use a UDF that is generated as a table function in a FROM clause. This example regenerates the getRate-UDF as a table function. The input parameters are echoed into the output table.

```
SELECT t.*
FROM countries c,
table( getRate('us', c.countries) ) t
```

The result is:

Table 23. Using getRate as a table function

AREA1	AREA2	RESULT
us	us	+1.00000000000000E+000
us	euro	+1.07280000000000E+000
us	uk	+6.84800000000000E-001

Related concepts:

- "The Web service consumer functions" on page 145
- "Web services consumer user-defined functions" on page 146

Related tasks:

- “Tracing Web services consumer events” on page 148

Web services consumer examples

The examples that are referred to here work with DB2 Universal Database Version 8. The file *<DB2_installed path>/samples/soapsample.sql* describes how to run the samples. The file *soapsample.sql* contains the following list of examples and sample queries:

- getTemp - Retrieves a temperature in Fahrenheit
- getRate - Returns the exchange rate between any two currencies

Related concepts:

- “The Web service consumer functions” on page 145
- “Web services consumer user-defined functions” on page 146
- “Web services consumer—using the WebSphere Studio User-Defined Function tool” on page 149

Related tasks:

- “Installation of the Web services consumer user-defined functions” on page 143

Related reference:

- “Using the Web services consumer UDFs” on page 159

Chapter 3. Developing federated, warehouse, and message queue applications

This section explains how to develop federated and warehouse applications and how to use message queue (MQ) functions.

Developing applications that use a federated server

The DB2 Universal Database server in a federated system is referred to as a federated server. To enable transparent data access, you can create nicknames for the remote data that you want to access. You can create functions or use already defined functions that compensate for differences between different data sources and simulate capabilities that are not natively supported. Multisite joins and unions promote integration of data from multiple sources.

Advantages of a federated system

A federated system is one type of distributed database management system that makes it possible for you to send distributed requests to multiple data sources within a single SQL statement. IBM® DB2® Information Integrator's federated systems complements built-in database support that is provided by Web application servers.

Without federated systems, accessing disparate data sources requires multiple steps:

1. You must connect to each data source individually.
2. You must extract the necessary data by using different native application programming interfaces.
3. You must filter, sort, and consolidate the data manually.

With a federated system, you simply query the nicknames using SELECT, INSERT, UPDATE, and DELETE statements.

In a federated system, you have transparent access to data that spans multiple heterogeneous sources. With federated systems you enhance the uses and power of a Web application server to support remote data (physically stored or dynamically generated). You can use federated systems with application server components such as enterprise beans and Web services. By using enterprise beans, and federated system objects, programmers can perform some database operations or transactional work, access multiple data sources, and create applications that integrate disparate data.

Related concepts:

- "Federated systems" in the *Federated Systems Guide*
- "Enterprise beans in a federated system" on page 165
- "DB2 Information Integrator—nonrelational federated technologies" on page 9
- "Performance and tuning planning— materialized query tables in a federated system" on page 22
- "DB2 Information Integrator—relational federated technologies" on page 7

© Copyright IBM Corp. 2003, 2004

Advantages of designing queries in IBM DB2 Information Integrator

DB2® Universal Database enables database administrators to create views of data that span multiple tables from different data sources. In a federated system, the views can encompass data that is stored in difference formats on multiple servers. To build such a view, you create nicknames for the remote data objects. Then you use SQL to create a view that joins or unions these nicknames. The views that join or union multiple data sources are read-only views. You can build container-managed persistence entity beans that are mapped to these views, which are read-only beans.

In addition to federated views, a federated system includes SQL data definition language (DDL) transparency for relational database managers. DDL transparency means that you can create a DB2 UDB table with an OPTIONS clause and perform two separate tasks with one statement. As the example in Figure 60 shows, you create a table at a remote data source and create a corresponding DB2 UDB nickname for this table.

```
     CREATE TABLE orarest (
 id int primary key not null,
 name varchar(20),
 cuisine varchar(20),
 budget int)
 OPTIONS (remote_server 'ORACLE8',
          remote_schema 'ORACLEUSER1')

           CREATE TABLE msrest (
 id int primary key not null,
 name varchar(20),
 cuisine varchar(20),
 budget int)
 OPTIONS (remote_server 'MSSQL',
          remote_schema 'MSUSER1')
```

Figure 60. Example of DDL transparency

The create statements with the OPTIONS clauses create the following database objects:

- A table named ORAREST in a remote Oracle database.
- A nickname named ORAREST in a DB2 UDB federated database.
- A table named MSREST in a remote Microsoft® SQL Server database.
- A nickname named MSREST in a DB2 UDB federated database.

You could then use the generated tables and nicknames in a view to union or integrate the information from two disparate data sources.

```
CREATE VIEW multirest
   (id, name, cuisine, budget)
  AS
  SELECT id, name, cuisine, budget
  FROM orarest UNION
   SELECT id, name, cuisine, budget FROM msrest;
```

When you use DDL transparency, the database manager performs most of the SQL translation to properly construct the remote table. You do not need to learn the specific SQL syntax of a data source to create a valid table.

You can also incorporate materialized query tables into your application to improve performance. Materialized query tables allow you to precompute whole or parts of each query and then use the computed results to answer future queries. Materialized query tables provide the means to save the results of prior queries and then reuse the common query results in subsequent queries. Materialized query tables help to avoid redundant scanning, aggregating and joining.

- Materialized query tables permit query processing, even when the remote data source is not available (such as when the network is not available). A materialized query table on a remote table can be perceived as a cache for that table, which can seem to enhance system availability.
- Materialized query tables increase the scalability of the overall system. The materialized query table can offload work from the primary database. You can have many local databases each with a copy of a subset of frequently used primary data. The primary database becomes less of a constraint on the enterprise.
- By using materialized query tables, you can avoid some connections to remote systems for some queries. The overall system throughput can potentially increase, and your total response time can decrease.
- Materialized query tables use the full functionality of DB2 UDB when accessing remote data sources. You can get the benefits of materialized query tables even though the remote data source does not support materialized query tables.

Web-enabled applications that are developed in a database environment can use several components of the Java™ 2 Enterprise Edition (J2EE) server environment. Some of the J2EE components include support for enterprise beans, servlets, JavaServer Pages code, and the Java Naming and Directory Interface extension. J2EE also offers support for connecting to the database manager and accessing the database manager, which includes support for Java Database Connectivity code and Java transaction application programming interfaces.

Related concepts:

- "What is transparent DDL?" in the *Federated Systems Guide*
- "Materialized query tables and federated systems – overview" in the *Federated Systems Guide*
- "Tuning query processing" in the *Federated Systems Guide*
- "Enterprise beans in a federated system" on page 165
- "Performance and tuning planning— materialized query tables in a federated system" on page 22

Related tasks:

- "Creating a federated materialized query table" in the *Federated Systems Guide*

Enterprise beans in a federated system

Enterprise beans are Java™ components that run on a Web server. You can create container-managed persistence entity beans to map to nicknames that you created with IBM® DB2® Information Integrator federated systems. The container-managed persistence entity bean can access data that is located in relational databases. The read-only container-managed persistence entity bean can access data that is located in nonrelational databases. You can use entity beans to integrate disparate data through Enterprise JavaBean architecture.

Enterprise JavaBean architecture

Enterprise bean components are part of the Enterprise JavaBean architecture and execute within an Enterprise JavaBean container. The container runs inside of an Enterprise JavaBean server. An enterprise bean implements business logic. The Enterprise JavaBean container provides services such as transaction and resource management, persistence, and security to the enterprise bean components. The details of database manipulation can be left to the Enterprise JavaBean container.

Entity beans and session beans

There are two types of enterprise beans that are significant for a federated system - session beans and entity beans.

Session beans
: A session bean is usually associated with a single client and is usually not persistent. The purpose of the session bean is not to represent or update existing database contents. Instead, the purpose of the session bean is to act as a single client that performs some actions on the server.

Entity beans
: An entity bean represents information that is stored persistently in a database. Entity beans are associated with database transactions. Entity beans can provide data access to multiple users. An entity bean might represent an underlying database row, or the result of a SELECT statement in a single row.

Federated systems support automated development and deployment of a single container-managed persistence entity bean whose attributes map to data from multiple resources. When you map a container-managed persistence entity bean to a federated database object, the federated server transparently translates the database access to data access requests that are appropriate to the data sources. When you deploy an enterprise bean, the bean resides in containers that provide services such as support for persistence. The entity bean automatically generates the code that implements persistence when you deploy the enterprise bean. By contrast, when you build session enterprise beans that access persistent data, you must write your own Java database connectivity statements to establish database connections and issue SQL statements.

A container-managed persistence entity bean defers all interaction with the database to the enterprise JavaBean container. Typically, the enterprise bean reads the data from the database and places the data into the fields in the container-managed persistence entity bean. You can reference or update the data in the entity bean. When a transaction ends, the Enterprise JavaBean container accesses the data in the entity bean and updates the underlying row in the table.

Related concepts:

- "Enterprise Java Beans" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Creating and deploying a container-managed persistence bean" on page 174

Employee skills scenario – solution design

YBar, Incorporated stores employee information in DB2® Universal Database tables, and XML documents. YBar, Incorporated can manipulate the data in the DB2 UDB tables, and in the XML documents. The logical solution is to create applications that store the employee resumes in the employee database as Extensible Markup Language (XML) documents. The company can then use XML search capability to find the resumes that match the skills that are needed for a specific project.

YBar, Incorporated, solves their business problem by using the federated systems of IBM® DB2 Information Integrator to access the resume files that are in an XML format as relational data.

1. To improve the performance of accessing the data in the flat file data sources, YBar, Incorporated creates indexes on the side tables that were defined in the DAD file. DB2 XML Extender creates side tables based on the schema definitions in the DAD file.

   ```
   CREATE INDEX KEY_Skill ON
         resume_skills_sidetable(skill)
   ```

2. The YBar, Inc, database administrator inserts three rows into the employees data in the relational table. The column named `Resume` is an XML column. The XMLVARCHARFROMFILE is a DB2 XML Extender function that reads an XML document from a server file and returns the document as an XMLVARCHAR type. The values for column `Resume` are populated from the resume file that is at 'Some_Path'\<the resume file contents>.

   ```
   INSERT INTO Employee
       (Emp_ID, Lastname, Firstname,
       Dept_ID, Current_job_ID, Resume)
       VALUES(12,'Douglas','Laurie',
         123, 1,
         db2xml.XMLVarcharFromFile
         ('Some_Path'\resume_ld.xml'))
   ```

   ```
   INSERT INTO Employee
       (Emp_ID, Lastname, Firstname,
       Dept_ID, Current_job_ID, Resume)
       VALUES(13,'Smith','John',
         123, 2,
         db2xml.XMLVarcharFromFile
         ('Some_Path'\resume_js.xml'))
   ```

   ```
   INSERT INTO Employee
       (Emp_ID, Lastname, Firstname,
       Dept_ID, Current_job_ID, Resume)
       VALUES(14,'Jackson','George',
         123, 3,
         db2xml.XMLVarcharFromFile
         ('Some_Path'\resume_gl.xml'))
   ```

3. Then YBar, Incorporated developer queries the base table by using the XML column and the side table. This SQL statement looks for all employees with Java™ skills. It takes advantage of the side table so that an index access can be used.

   ```
   SELECT e.firstname, e.lastname,e.resume
     FROM employee
      AS e, resume_skills_sidetable AS r
       WHERE e.emp_id=r.emp_id
         AND r.skill LIKE '%Java%'
   ```

The YBar, Inc., developers can query the XML content in several ways:

- The following example uses a scalar function to return the experience of a single employee:

  ```
  SELECT db2xml.EXTRACTVARCHAR
     (resume,'/resume/experience')
     FROM employee
     WHERE emp_id=13
  ```

The user-defined function extractVarchar uses the column `resume` as the input and the location path /resume/experience as the select identifier. The scalar function returns the experience of one employee. With the WHERE clause, the extracting function evaluates only the resume with an identifier of ″13″..

- The following example uses a table function to return several rows:

```
SELECT e.firstname, e.lastname, e.resume
   FROM employee
   AS e,
    TABLE(db2xml.EXTRACTVARCHARS(e.resume,'/resume/skill'))
    AS t
   WHERE t.returnedvarchar LIKE '%Java%'
```

The user-defined function extractVarchars uses the column `resume` as the input and the location path /resume/skill as the select identifier. The table function returns the skills of the employees that include the string ″Java″ in the skills node of the resume file. Here is an example of part of the original resume.xml file:

```
<resume emp_id="12" email_address="some_person@email_address.com">
   <experience start_date="12/01/1999" end_date="06/30/2002">
     Develop partner and customer demos.
    </experience>
   <experience start_date="12/01/1999" end_date="06/30/2002">
     IBM DB2 Information Integrator development
   </experience>
   <skill>
     Databases
   </skill>
   <skill>
     Java
   </skill>
   <skill>
     C++
   </skill>
   <skill>
     FORTRAN
   </skill>
</resume>
```

Solving the external education problem

The human resources department needs to send a list of employees and their current job descriptions to an external education provider. The education provider can then offer customized classes to the employees of YBar, Incorporated. The YBar, Inc., developers join the employee and the job databases and publish the list in a message queue by using WebSphere® MQ. The following steps demonstrate the example solution:

1. The job table is created as a federated systems nickname of an external flat file. The nickname is created as part of the flat file wrapper.

```
CREATE NICKNAME job
(Job_ID INTEGER,
Job_description VARCHAR(255),
Title VARCHAR(50),
responsibilities VARCHAR(100))
FOR SERVER local_flat_files OPTIONS
(FILE_PATH 'C:\SPC\job_table.txt',
COLUMN_DELIMITER ',',
KEY_COLUMN 'job_id',
VALIDATE_DATA_FILE 'y')"
```

The FILE_PATH option defines the location of the external file, which is named job_table.txt.

2. The developers run the query that selects the employee information and the job information.

   ```
   SELECT 1 as x, emp_id , firstname, lastname,
   Job_description
   FROM employee
   as e, job as j
   WHERE e.Current_Job_ID = j.Job_ID
   and j.Job_ID != 1000
   ORDER BY x, emp_id
   ```
3. The YBar developers generate a new XML document that contains the joined information from the DB2 UDB table, employee, and the external flat file information, job. They name the new generated file, All_employee.xml. They use the employee.dad file to map the xml information to the DB2 UDB relational table information.
4. Create the DB2 UDB temporary table to hold the XML documentation with the following statement:

   ```
   create table tmpTable
       (x_doc DB2XML.XMLCLOB not logged)
   ```
5. Decompose the XML information that is in the employee.dad and populated tmpTable. Then
6. Write XML data and store it in the file employee.xml by using the following example:

   ```
   select DB2XML.Content
       (x_doc, 'c:\YourName\employee.XML')
       from tmpTable
   ```
7. The information is now prepared to be sent as a message to the external education provider. You put the joined information in a WebSphere MQ message queue. :
8. The following examples put the messages on the queue as an XML document, displays the contents of the message queue, and leaves the messages undisturbed.

   Use the MQSENDXML function to send an XML message to the queue. The message is the resume information for employee number 12.

   ```
   db2 "SELECT db2xml.MQSENDXML('DB2.DEFAULT.SERVICE',
       'DB2.DEFAULT.POLICY', resume)
       FROM employee
       WHERE emp_id=12"
   ```

   Create a DB2 UDB table that has one column to hold the entire XML document.

   ```
   db2 "CREATE TABLE temporaryXML
       (XML_doc DB2XML.XMLVARCHAR)"
   ```

   Populate the table with the XML information that is a join of the DB2 UDB employee table and the external job file:

   ```
   db2 "INSERT INTO temporaryXML
       SELECT CONCAT('<FullName>'||firstname
       ||' '||lastname||'</FullName>          ','<Job_Desc>'||
       job_description||
       '</Job_Desk>')
       FROM employee
       AS e, job AS j
       WHERE e.current_job_id=j.job_id
       AND  j.Job_ID = 1"
   ```

   Use the MQSENDXML function to send an XML message to the queue. The message is the employee and job information in an XML column.

```
db2 "SELECT db2xml.MQSENDXML('DB2.DEFAULT.SERVICE',
    'DB2.DEFAULT.POLICY', XML_doc)
    FROM temporaryXML"
```

The complete set of components that YBar, Incorporated uses can be seen in Figure 61.

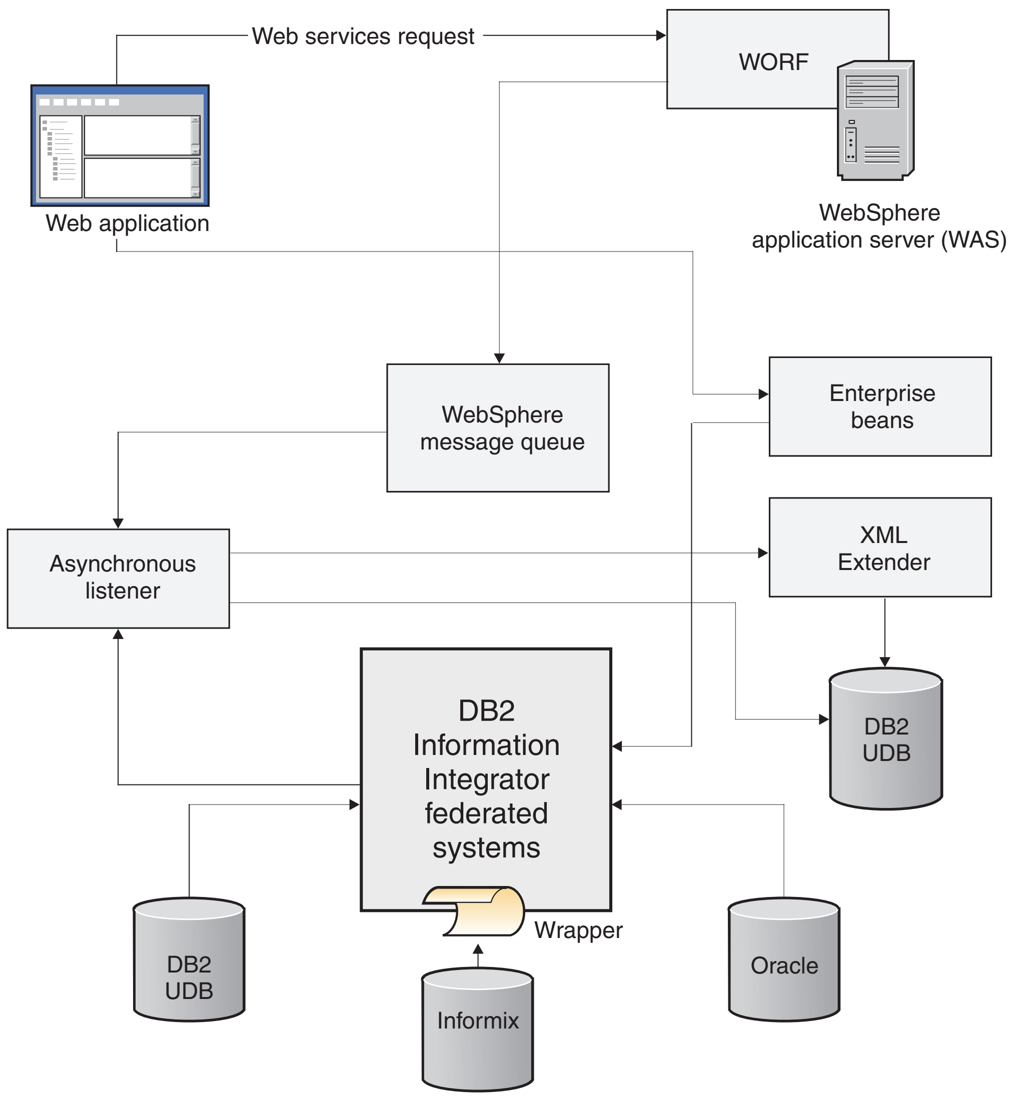


Figure 61. Components of a federated system that use Web applications

Related concepts:

- “Planning side tables” in the *DB2 XML Extender Administration and Programming*
- “Using indexes for XML column data” in the *DB2 XML Extender Administration and Programming*
- “Employee database scenario - solution design” on page 170
- “Discovering the data—the employee skills scenario” on page 12

Employee database scenario - solution design

| YBar, Incorporated, a company that specializes in human resources, is trying to
| solve two problems:

- They must identify the best candidates to fill some internal job openings. They base their search on the personnel resume files that are in XML format.
- They provide an employee list to an external education provider. The education provider customizes classes according to employee needs.

YBar, Inc, solves their business problem by using the federated systems of IBM® DB2® Information Integrator to store and retrieve the data. The YBar, Inc. design solution is in Figure 62 on page 171. This design uses several functions of the DB2 XML Extenders, and takes advantage of the use of side tables that match the schema in the data type definition files. Side tables are DB2 UDB tables that are used to extract the content of an XML document that will be searched frequently. The XML column is associated with side tables that hold the contents of the XML document. When the XML document is updated in the application table, the values in the side tables are automatically updated.

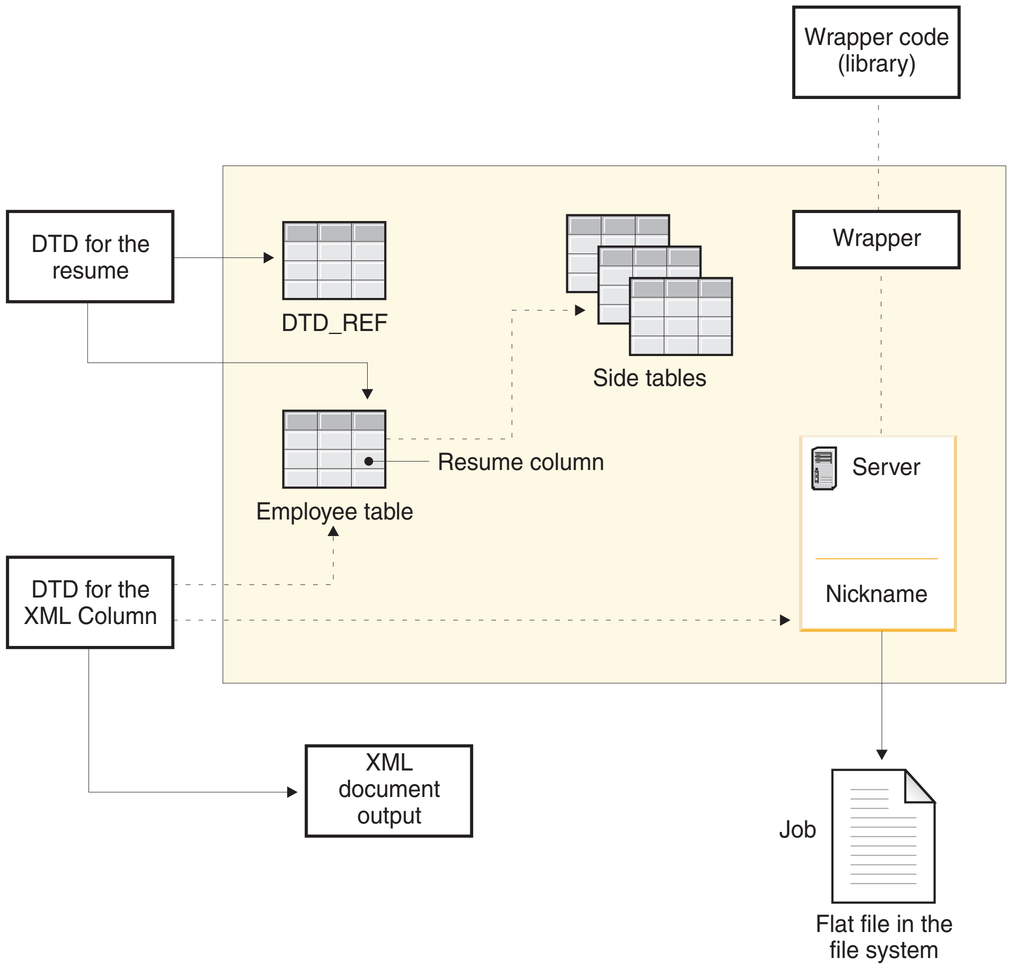


Figure 62. YBar, Inc design flow

YBar, Inc, can treat the XML data that is stored in the DB2 UDB tables as relational data that can be used with the other columns of data. They use the functions of DB2 XML Extender to store the resumes that are in XML format as complete XML documents.

1. You must enable the database for federated systems so that you can access that data that is stored in the external flat files.

   ```
   DB2 UPDATE DBM CFG USING FEDERATED YES
   ```

2. You must enable the database for DB2 XML Extenders, to use the XML functions.

   ```
   DXXADM ENABLE_DB emp_db
   ```

3. The database management administrator adds a column named "Resume" to the Employee table. The XMLVARCHAR data type is a DB2 XML Extender type that allows an XML document to be included in the DB2 UDB table.

   ```
   ALTER TABLE employee ADD COLUMN Resume XMLVARCHAR
   ```

   The new column contains the resumes in the employee database as XML documents. The Resume column provides an XML search capability to find which resumes match the skills that are needed for a specific project.

4. You must create the federated data sources such as wrappers, servers, and nicknames. You can use the DB2 UDB Control Center to create the federated data objects.

   The following example registers the table-structured file wrapper in AIX®:

   ```
   CREATE WRAPPER flat_files LIBRARY 'libdb2lsfile.a'
   ```

   You must register a server definition for each wrapper that you want to use.

   ```
   CREATE SERVER local_flat_files WRAPPER flat_files
   ```

   You register a nickname to use the nonrelational data in SQL statements. The FILE_PATH option defines the path to the resume data that is stored in a flat file in another data source.

   ```

   CREATE NICKNAME job
   (Job_ID INTEGER,
   Job_description VARCHAR(255),
   Title VARCHAR(50),
   responsibilities VARCHAR(100))
   FOR SERVER local_flat_files OPTIONS
   (FILE_PATH 'C:\SPC\job_table.txt',
   COLUMN_DELIMITER ',',
   KEY_COLUMN 'job_id',
   VALIDATE_DATA_FILE 'y')"
   ```

5. Verify that the nickname options are valid by using the following statement:

   ```
   SELECT tabschema,
   tabname, option, setting
   FROM SYSCAT.TABOPTIONS
   ```

   The result of the SELECT statement is in Table 24 on page 172. This table shows the schema name and the table options that were defined in the CREATE NICKNAME statement.

Table 24. Result of select from the TABOPTIONS table

TABSCHEMA	TABNAME	OPTION	SETTING
USERNAME	JOB	FILE_PATH	c:\SPC\job_table.txt
USERNAME	JOB	COLUMN_DELIMITER	,
USERNAME	JOB	KEY_COLUMN	JOB_ID
USERNAME	JOB	VALIDATE_DATA_FILE	Y
USERNAME	JOB	REMOTE_TABLE	JOB
USERNAME	JOB	REMOTE_SCHEMA	USERNAME
USERNAME	JOB	SERVER	LOCAL_FLAT_FILES

6. You must register the document type definition (DTD) to map the XML schema to a DB2 UDB table, and then enable the XML column, `resume`.

   ```
   DXXADM enable_column Employee_db
     Employee resume resume.dad -v resume_view -r Emp_ID
   ```

The `enable_column` option of the DB2 XML Extender command DXXADM connects to a database and enables an XML column so that it can contain the XML Extender user-defined types. When enabling a column, the XML Extender completes several tasks, including the following tasks:

- Creates side tables that are specified in the DAD file with a column containing a unique identifier for each row in the XML table. The column in the side table is `Emp_ID` in this example.
- Creates a default view for the XML table and its side tables, using the name `resume_view` in this example.

The following definitions are used for the terms in the `enable_column` statement:

- `Employee_db`: the name of the database.
- `Employee`: the name of the table.
- `resume`: the name of the XML column
- `resume.dad`: the name of the document access definition file that contains the schema definition.
- `resume_view`: the name of the default view that joins the XML column and the side tables.
- `Emp_ID`: the name of the primary key in the XML column table that will be used as the root_id for the side tables.

7. You can use a DTD to validate XML data in an XML column or in an XML collection. You can store a DTD in the DTD repository table, which is a DB2 UDB table named DTD_REF. The DTD_REF table has a schema name of DB2XML. Each DTD in the DTD_REF table has a unique ID, named the DTDID. The DTDID can be an identifier or it can be the path that specifies the location of the DTD on the local system. The DTDID must match the value that is specified in the DAD file for the <DTDID> element. The XML Extender creates the DTD_REF table when you enable a database for XML. You can insert the DTD from the command line by issuing this statement.
```
INSERT INTO db2xml.dtd_ref VALUES
  ('resume.dtd',db2xml.XMLClobFromFile
      ('%PATH_DEMO%\resume.dtd'),0,
      'user1','user1','user1')
```
8. You can verify that you registered the DTD correctly by issuing this statement:
```
SELECT dtdid, content FROM db2xml.dtd_ref;
```
In this statement, `content` is the content of the DTD.
9. Query the data by using side tables and an extracting user-defined function:
```
SELECT e.firstname, e.lastname, e.resume
  FROM employee AS e,
 resume_skills_sidetable AS r
 TABLE(DB2XML.EXTRACTVARCHARS(e.resume,'/resume/skill')) AS t
  WHERE SUBSTR(t.returnedvarchar,1,8) LIKE '____Java'
```
The XML documents are stored in the column `resume`. The user-defined function extractVarchars() uses the column resume as the input and the location path /resume/skill as the select identifier. The user-defined function returns a table of employee skills that are listed in their resumes.

Related concepts:

- "Planning side tables" in the *DB2 XML Extender Administration and Programming*
- "Advantages of a federated system" on page 163
- "Employee skills scenario – solution design" on page 166

Related tasks:

- "Accessing heterogeneous data through federated views" in the *Federated Systems Guide*
- "Adding DB2 family data sources to a federated server" in the *IBM DB2 Information Integrator Data Source Configuration Guide*

Creating and deploying a container-managed persistence bean

When you define a container-managed persistence entity bean, you use a deployment descriptor file to control the data source that contains the persistent data and any access restrictions. After you develop the entity bean, you set the deployment descriptors that govern the characteristics of the bean, and then package and deploy the bean.

You can create container-managed persistence entity beans that map to federated systems nicknames. Nicknames are federated objects that represent remote data to the database manager. The example that is used throughout this section creates individual entity beans that map to nicknames that are associated with individual Oracle tables, DB2 Universal Database tables, and flat files. A single container-managed persistence entity bean can span multiple data sources. The entity bean provides a way of integrating disparate data through standard enterprise JavaBean technology.

Prerequisites:

Install WebSphere Studio Version 5 or later.

Register and create federated systems objects for Oracle tables and flat files.

Procedure:

To create and deploy a container-manager persistence entity bean using WebSphere Studio:

1. From the Java 2 Enterprise Edition (J2EE) perspective in WebSphere Studio, create an `EJB` project for the entity bean you create.
2. Create the container-managed persistence entity bean.
   - Give the container-manager persistence entity bean the same name as the nickname you defined for the data source.
   - Add attributes that correspond to each of the column names in the nickname. Specify the data type for each column that corresponds to the data type in the column of the nickname. Designate one of the attributes as the key field, which should map to the primary key column of the nickname.
3. Open the enterprise JavaBean data modeling window:
   a. Select the entity bean you created and right-click to open the drop-down menu. Select **Generate —> EJB to RDB mapping** and select **Top-down modeling**
   b. Click **Next**. Ensure that you have set the database name and the schema names properly. The database name must map to the federated database known to your DB2 UDB client. The schema name must map to the authorized federated database user.
   c. Clear **Generate DDL**.
   d. Click **Finish**.

4. Verify that the mapping between the entity bean and the database completed successfully.
   a. Select the entity bean module and right-click to open the drop-down menu.
   b. Select **Open With —> Mapping Editor.**
   c. Correct any errors on the **Tasks** window.
5. Bind the entity bean to the data source that you created for the federated database.
   a. Select the entity bean and right-click to open the drop-down menu.
   b. Select **Open with —> Deployment Descriptor Editor**.
   c. On the Overview page, scroll down to WebSphere Bindings. For JNDI-CMP Factory Bindings, specify a valid JNDI name and container authorization type.
   d. Save your modifications and close the window.
6. Generate the code to deploy the entity bean.
   a. Select the entity bean and right-click to open the drop-down menu.
   b. Select **Generate Deploy Code**.

Related concepts:
- "Java 2 Enterprise Edition applications" on page 131
- "Enterprise beans in a federated system" on page 165

Designing applications for a federated solution—Cottonwood Distributors, Incorporated

The database programmers at Cottonwood Distributors, Incorporated (CDI), create the federated objects that they need to access all of the newly acquired data sources. The programmers need to create federated objects on all three of the data sources that access the remote systems. The programmers create the access by using the statements shown in Figure 63 on page 176. This example shows the SQL data definition language (DDL) statements for one of the data sources, DB2® Universal Database.

```
...
catalog tcpip node DB2_TPCH remote x.xx.xx.xx server 50000;
...
catalog database tpcd at node db2_tpch;
...
create wrapper drda;

create server db2_tpch type db2/udb version  8.1
  wrapper drda authorization "demo" password "xxxxx"
  options (dbname 'TPCD');
create user mapping
  for user SERVER db2_tpch
  OPTIONS ( REMOTE_AUTHID 'demo', REMOTE_PASSWORD 'xxxxx' );

create nickname db2_part for db2_tpch.tpcd.part;
create nickname db2_supplier for db2_tpch.tpcd.supplier;
create nickname db2_partsupp for db2_tpch.tpcd.partsupp;
create nickname db2_nation for db2_tpch.tpcd.nation;
create nickname db2_region for db2_tpch.tpcd.region;
create nickname db2_customer for db2_tpch.tpcd.customer;
create nickname db2_orders for db2_tpch.tpcd.orders;
```

Figure 63. Some federated objects for the DB2 data source

The programmers at Cottonwood need to create views of the main tables (part, partsupp, supplier) for the three different data sources. They create a view by using a UNION ALL statement across three data sources. The example in Figure 64 on page 176 shows one of the views.

```
CREATE  VIEW part_fed (
  p_partkey, p_mfgr, p_type, p_size, p_retailprice) AS
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM db2_part
UNION ALL
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM inf_part
UNION ALL
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM ora_part;
```

Figure 64. Unioning the three data sources

The programs that the Cottonwood database programmers write, need to access an XML file that comes from an outsourced vendor. This XML file contains human resource information such as employee data. To access this file, the programmers at Cottonwood create a wrapper, a server, and a nickname for XML files, as shown in Figure 65 on page 177.

```
create wrapper XML_files library 'db2lsxml.dll';
create server LOCAL_XML_FILES wrapper XML_FILES;
create nickname Employees_From_XML (
  doc Varchar(100) OPTIONS(DOCUMENT 'FILE'),
  Employee_Number Varchar(5) OPTIONS(XPATH './@SerialNum'),
  First_Name Varchar(50) OPTIONS(XPATH './/Firstname'),
  Middle_Initial Varchar(50) OPTIONS(XPATH './/Initial'),
  Last_Name Varchar(50) OPTIONS(XPATH './/Lastname'),
  Department_Number Varchar(50) OPTIONS(XPATH './/Department'),
  Phone_Number Varchar(50) OPTIONS(XPATH './/PhoneNumber'),
  Job Varchar(50) OPTIONS(XPATH './/Job'),
  Education_Level Varchar(50) OPTIONS(XPATH './/EDLevel'),
  Gender Varchar(50) OPTIONS(XPATH './/Sex'),
  Hire_Date Varchar(50) OPTIONS(XPATH './/HireDate'),
  Birth_Date Varchar(50) OPTIONS(XPATH './/BirthDate'),
  Annual_Salary Varchar(50) OPTIONS(XPATH './/Salary'),
  Annual_Bonus Varchar(50) OPTIONS(XPATH './/Bonus'),
  Commission Varchar(50) OPTIONS(XPATH './/Comm'),
  cid Varchar(16) OPTIONS(PRIMARY_KEY 'YES'))
 FOR SERVER LOCAL_XML_FILES
 OPTIONS (XPATH '//Employee');
```

Figure 65. Creating the federated systems objects

Because the Web applications route customer orders and supplier price updates to a queue, the programmers create table-read functions and views (see Figure 66 on page 177) of the Message Queues .

```
CREATE FUNCTION NEW_SUPPLIERS_READ()
    RETURNS TABLE  ( SUPPLIER_NAME VARCHAR(80),
                        SUPPLIER_PHONE VARCHAR(12),
                        PART_KEY DOUBLE,
                        PART_PRICE DOUBLE,
                        MAN_DAYS DOUBLE,
                        MAX_QUANTITY DOUBLE,
                    CORRELID VARCHAR(80))
    LANGUAGE SQL
    NOT DETERMINISTIC
    EXTERNAL ACTION
    READS SQL DATA
    RETURN
 SELECT
         VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),80),
         VARCHAR(DB2MQ.GETCOL(T.MSG,',',2),12),
         DOUBLE(DB2MQ.GETCOL(T.MSG,',',3)),
         DEC(DB2MQ.GETCOL(T.MSG,',',4),8,4),
         BIGINT(DB2MQ.GETCOL(T.MSG,',',5)),
         BIGINT(DB2MQ.GETCOL(T.MSG,',',6)),
  CORRELID
 FROM TABLE (DB2MQ.MQREADALL('DB2.DEFAULT.SERVICE',
       'DB2.DEFAULT.POLICY')) AS T;

CREATE VIEW READ_NEW_SUPPLIERS_FROM_QUEUE
  AS SELECT * FROM TABLE(NEW_SUPPLIERS_READ()) t
  WHERE CORRELID = 'CDI_NEW_SUPPLIER';
```

Figure 66. Cottonwood's table-read functions

Finally, the programmers at Cottonwood need to create temporary tables that will be used in processing:

- For running the XML composition:
  ```
  create table Ucustomers (x_doc DB2XML.XMLCLOB not logged);
  ```

- For storing the Web request:

```
create table request_bid
  (reqkey integer not null,
   partkey integer not null,
   bid double not null);

create table request_status
  (reqkey integer not null,
   partkey integer not null,
   suppkey integer not null,
   newquote double not null,
   currentquote double not null,
   status varchar(15) not null);
```

Related tasks:

- "Developing the application for a federated solution—Cottonwood Distributors, Inc." on page 178

Related reference:

- Appendix A, "Script examples for Cottonwood Distributors, Inc. and YBar, Inc. scenarios," on page 217

Developing the application for a federated solution—Cottonwood Distributors, Inc.

Cottonwood Distributors, Incorporated (CDI) can generate an XML message for any new quote or bid by invoking a Web service with a servlet. Here are some steps that you can follow to create a similar environment.

Procedure:

To develop the application to create the Cottonwood Web service:

1. Configure the DB2 Universal Database server to act as a federated database and set the properties for the SVCENAME server and the FEDERATED option:
   ```
   db2 update dbm cfg using SVCENAME myID authentication server
   db2 update dbm cfg using FEDERATED yes
   db2 connect reset
   db2stop
   db2start
   ```
2. Connect to the local DB2 Universal Database:
   ```
   db2 connect to myLocalDB user user1 using myPW
   ```
3. Create the federated objects for each data source.
4. Configure the DB2 Universal Database client to identify the remote nodes on which the DB2 UDB server resides.
5. Create nicknames for the remote data that you want to access:
   ```
   create nickname ora_part for oraserver.CDI.part;
   ```
6. Configure the WebSphere Application Server:
   - | Make sure that the Java build path that is associated with your Web services projects includes the location of the *db2java.zip* or *jcc.jar* file. The `dbDriver` parameter in the `group.properties` file determines what database driver package that you use.
   - Create a data source object in your WebSphere environment that maps to the DB2 Universal Database that you configured for federated support. Data

source objects represent pooled database connections, and can be established using JNDI services to search a data source, and invoke methods that are associated with the data source.

With all of the connections made, you can now select, insert, update, or delete data from the federated data objects. Because you created a nickname of ora_part, the SQL statements that you create reference only the ora_part nickname. In the Cottonwood examples, the statements include joined data from multiple sources with a single SQL statement that references multiple nicknames.

The code shown in Figure 67 uses a Web service to write a message to the queue.

```
public void doGet(HttpServletRequest req,
 HttpServletResponse res)
  throws javax.servlet.ServletException, java.io.IOException {
  try {
   PrintWriter pr = res.getWriter();
   pr.print(htmlHeader1);
   pr.print(message1);
   pr.print(message2);
   String method = "";

...
```

Figure 67. New quote or bid writes a message to the queue (MessageFormatter.java) (Part 1 of 5)

```
public class MessageFormatter
  extends javax.servlet.http.HttpServlet
  {

final static String htmlHeader1 = "<HTML><TITLE>MessageFormatter";
  final static String message1 =
 "<p align=\"center\"><font color=\"navy\" face=\"verdana\"
 size=\"+2\"><b>Thank You<p><font color=\"black\"
 face=\"veranda\" size=\"+1\">";
 final static String message2 = "";

 final static String htmlHeader2 = "";
 static String quote = "";

 static Connection con = null;
 final static String url = "jdbc:db2:demo";

...
```

Figure 67. New quote or bid writes a message to the queue (MessageFormatter.java) (Part 2 of 5)

```
WSProxy WSid = new WSProxy();
   boolean fCustomer = true;
if (req.getParameter("method")!=null) {
    //Get the common parms to the servlet
    //from the REQ object
    key = req.getParameter("name");
    part = req.getParameter("part");
    method = req.getParameter("method");

    // If customer order
    if (method.equals("orderNewParts")) {
     fCustomer = true;
     // Get the quantity the customer is ordering
     quantity = req.getParameter("quantity");

// set the table and column names for SELECT
     col_name1 = "c_name";
     col_name2 = "c_custkey";
     tab_name  = "db2_customer";

    ...
```

Figure 67. New quote or bid writes a message to the queue (MessageFormatter.java) (Part 3 of 5)

```
   // Get the real customer name from federated data source
    try {
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
     url = "jdbc:db2:demo";
     con = DriverManager.getConnection(url,"demo","xxxxx");
     stmt = con.createStatement();
     rs = stmt.executeQuery
        ("SELECT " + col_name1 + " from " + tab_name +
          " where " + col_name2 + " = " + key);
     while (rs.next()) {
      cust_name = rs.getString(1);
     }
...
```

Figure 67. New quote or bid writes a message to the queue (MessageFormatter.java) (Part 4 of 5)

```
   // Write request status back to user
   try{
    // If this is a supplier update request
    if (!fCustomer) {
     String message1id=  "2," + key + "," + part + ","
          + quantity + "," + price ;
     java.lang.String message1idTemp  = message1id;
     System.out.println
           ("message type 2 being written: " + message1idTemp);
     org.tempuri.worftestweb.
        demo.newdadx.dadx.xsd.Stmt1ResultElement mtemp =
     WSid.stmt1(message1idTemp);
```

Figure 67. New quote or bid writes a message to the queue (MessageFormatter.java) (Part 5 of 5)

The *CustomerRoutine* parses the message. The routine takes actions based on the message type (a quote or a bid). The routine puts parsed information into the

REQUEST_BID or REQUEST_STATUS table. If the message type is a quote for a part, it compares the new quote to an existing quote. If this is a quote for a new part, the status is changed to new. The status is added to the local and status tables as NEW. If the quote is for an existing part with a lower price, then the program accepts the price. The program writes to the local table, and updates the status as ACCEPTED. If the quote is for an existing part and the price is higher, then the program writes to the status table as REVIEW.

Related concepts:

- "Deploying the application—Cottonwood Distributors, Inc. solution" on page 189
- "Discovering the data—Cottonwood Distributors, Inc." on page 185

Deploying a federated application

To deploy a federated application, the database programmers at Cottonwood Distributors, Incorporated invoke *newdadx.dad* for Web service actions. Here are some examples to help you deploy your application.

Prerequisites:

The minimum deployable unit in WebSphere Application Server 5.0 is a Web archive (WAR) file. If the application is developing Enterprise JavaBeans (EJB), then a Java archive file (JAR) and an enterprise archive file (EAR) are necessary.

Before you can deploy an application, you must build the following components for your enterprise federated application:

WAR file
The Web-related components (Hypertext Markup Language (HTML), JavaScript, JSP)

JAR file
The Java classes that make up the business logic components

EAR file
The JAR files plus the WAR files that make up an enterprise solution

Procedure:

To deploy the federated application:

1. Import the EAR file into the WebSphere Application Development environment. In the case of the Cottonwood Distributors, Incorporated, the EAR file (CDI.ear) contains the following files:
   - .project
   - META-INF/.modulemaps
   - META-INF/application.xml
   - META-INF/ibm-application-ext.xml
   - META-INF/MANIFEST.MF
   - WorfTestWeb.war
2. Deploy the Web services (the document access definition extension (DADX) files) by selecting each DADX file in the WebSphere Studio window and clicking the **Deploy as Web service** push button.

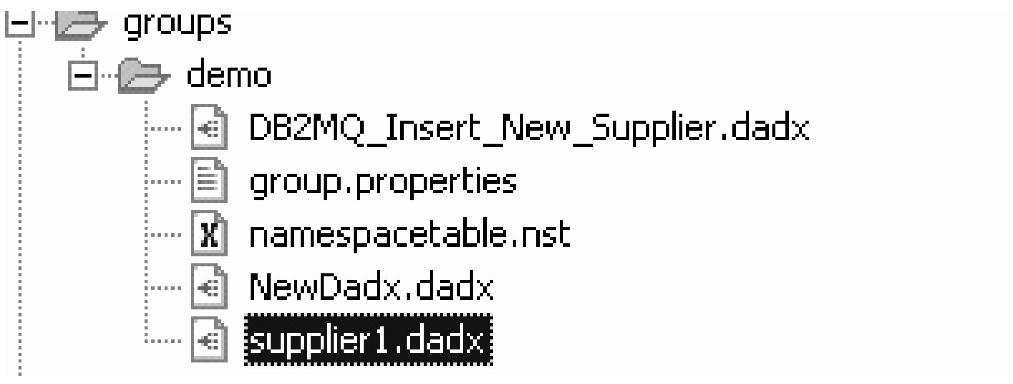


Figure 68. Select the DADX files

3. Restart the WebSphere Application Development server.

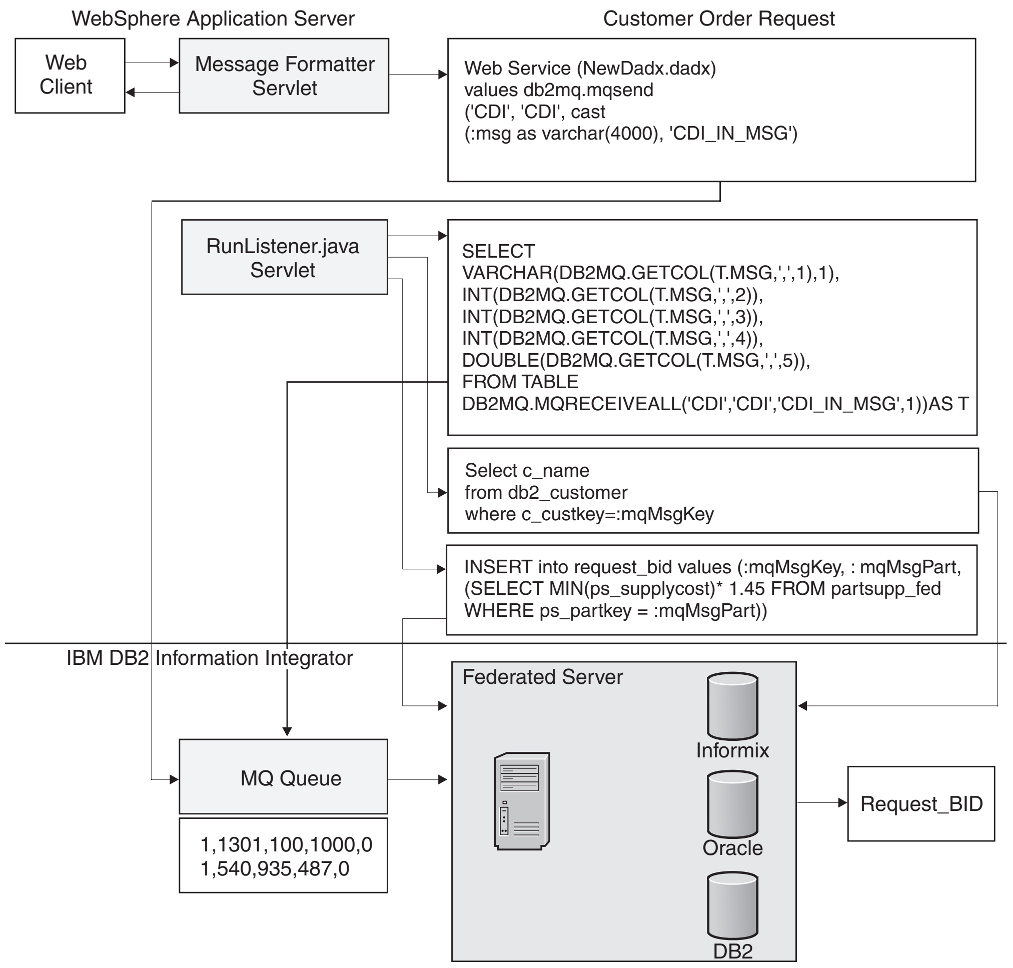
The application that is deployed provides the customer bid requests and status of bids that is required of Cottonwood's new merged enterprise. The customer bid requests are sent through the Web services. The actual request is a message that is routed to a WebSphere message queue. Cottonwood has an application on that message queue that listens for activity so that the programs can retrieve database requests from the queue as the requests enter the queue. This listener application invokes a set of actions to process the requests. The listener used by Cottonwood balances the load that is introduced by the number of bid requests that enter the queue, and processes the customer bid requests with the following steps:

1. The customer bid request writes a message to the queue.
2. The listener application invokes a DB2 Universal Database function to extract the order information from the request.
3. The database programmers at Cottonwood run a read-only query to obtain a quote for the requested part.
4. The database programmers at Cottonwood insert a record into the local table to record the order.

Cottonwood processes the supplier quote updates in a series of steps that are similar to the customer bid requests. However, the supplier quotes query is not read-only because the program needs to update the quotes. The listener application allows Cottonwood to inspect the request before allowing the update to commit. Here are the specific steps for the supplier quote requests:

1. The supplier quote request writes a message to the queue.
2. Based on the format of the message, the listener application invokes a DB2 Universal Database function to extract the request to update a quote for a part.
3. The database programmers at Cottonwood review the previously lowest quote from that supplier for that part. If the new quote is lower, or if the quote is for a part not previously supplied by that supplier, the application updates the database. If the new quote is higher, Cottonwood might not make the update. Instead, the Cottonwood application marks the quotes as something that the Cottonwood marketing team should review later or to negotiate with the supplier.

The complete flow for the customer orders and supplier price updates is in Figure 69 on page 183 and Figure 70 on page 184.



```
Web Service (NewDadx.dadx)
values db2mq.mqsend
('CDI', 'CDI', cast
(:msg as varchar(4000), 'CDI_IN_MSG')
```

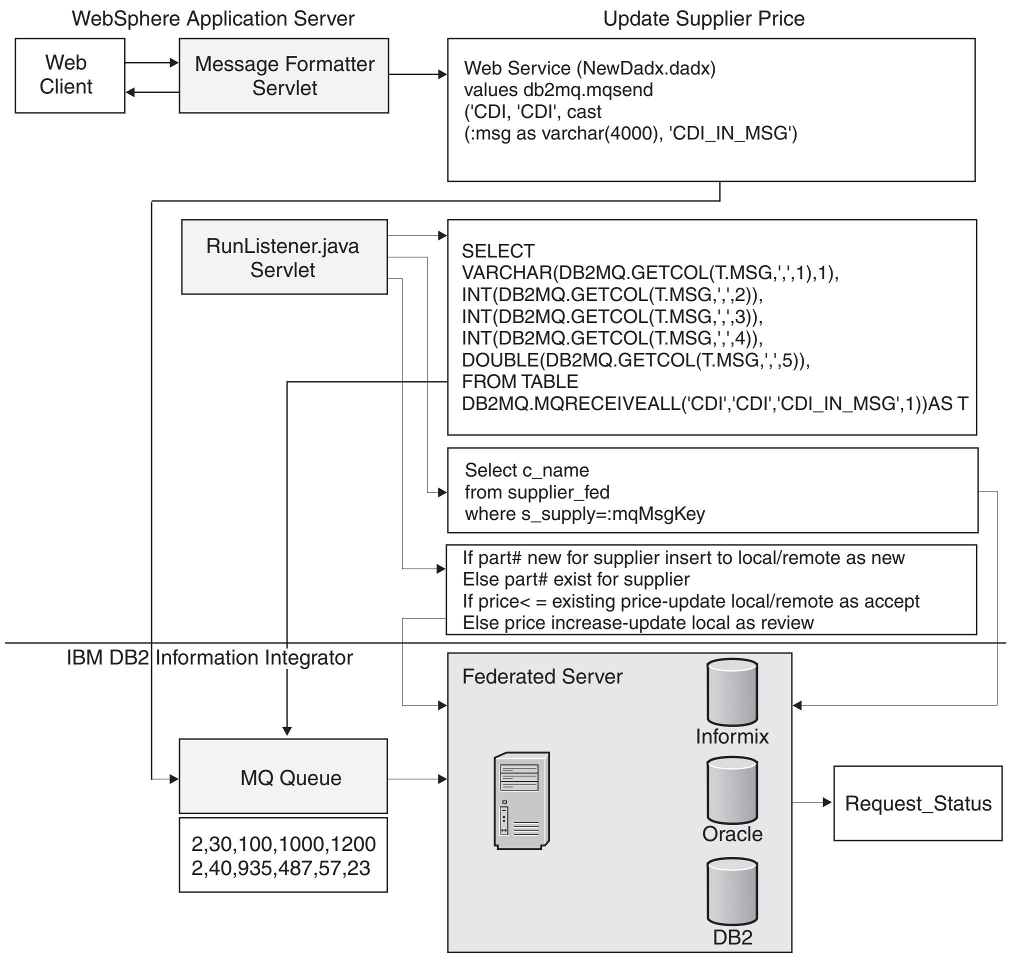
```
SELECT
VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),1),
INT(DB2MQ.GETCOL(T.MSG,',',2)),
INT(DB2MQ.GETCOL(T.MSG,',',3)),
INT(DB2MQ.GETCOL(T.MSG,',',4)),
DOUBLE(DB2MQ.GETCOL(T.MSG,',',5)),
FROM TABLE
DB2MQ.MQRECEIVEALL('CDI','CDI','CDI_IN_MSG',1))AS T
```

```
Select c_name
from db2_customer
where c_custkey=:mqMsgKey
```

```
INSERT into request_bid values (:mqMsgKey, : mqMsgPart,
(SELECT MIN(ps_supplycost)* 1.45 FROM partsupp_fed
WHERE ps_partkey = :mqMsgPart))
```

```
1,1301,100,1000,0
1,540,935,487,0
```

Figure 69. Cottonwood customer order requests



```
Web Service (NewDadx.dadx)
values db2mq.mqsend
('CDI, 'CDI', cast
(:msg as varchar(4000), 'CDI_IN_MSG')
```

```
SELECT
VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),1),
INT(DB2MQ.GETCOL(T.MSG,',',2)),
INT(DB2MQ.GETCOL(T.MSG,',',3)),
INT(DB2MQ.GETCOL(T.MSG,',',4)),
DOUBLE(DB2MQ.GETCOL(T.MSG,',',5)),
FROM TABLE
DB2MQ.MQRECEIVEALL('CDI','CDI','CDI_IN_MSG',1))AS T
```

```
Select c_name
from supplier_fed
where s_supply=:mqMsgKey
```

```
If part# new for supplier insert to local/remote as new
Else part# exist for supplier
If price< = existing price-update local/remote as accept
Else price increase-update local as review
```

```
2,30,100,1000,1200
2,40,935,487,57,23
```

Figure 70. Supplier prices can be updated

Related tasks:

- "Preparing and creating the Web archive file" on page 136

Extending the data warehouse

This section explores a specific example of how DB2 Warehouse Manager and the Data Warehouse Center can be used as part of your information integration solution.

Business solutions: extending the DB2 Warehouse Manager

DB2® Warehouse Manager provides improved integration with DB2 and IBM® DB2 Information Integrator utility functions. You can use SQL UPDATE statements to update target data with the data from the source. In addition, you can define a process that waits for multiple steps to complete. Warehouse transformers, which include stored procedures and user-defined functions, provide commonly used transformations for building data warehouses.

The Data Warehouse Center is a metadata-driven system. Metadata, or information about your data, provides administrators and business users with descriptions of the data that is stored in the data warehouse. You can create information catalogs that describe business metadata in business terms. You can organize the metadata into subject areas, and customize it to your workgroup or the needs of your enterprise.

By using such technologies as the Clean Data transformer, you can perform basic scrubbing, substitution, and mapping operations on source data. You can also define your applications to the Data Warehouse Center so that one or more steps can use the program for processing. After you define a user-defined program to the Data Warehouse Center, the program definition is available for use as a step in the Process Model window. Warehouse sources identify the tables and files that will provide data to your warehouse. The Data Warehouse Center uses the specifications in the warehouse sources to access the data. The sources can be almost any relational or nonrelational source (table, view, file, or predefined nickname).

Use the Information Catalog Manager to provide a graphical representation of data relationships and object definitions for warehouse steps. The Information Catalog Manager provides a powerful, business-oriented solution to help users locate, understand, and access enterprise data. The Information Catalog Manager enables business users to view aggregations, histories, data derivations, data sources, and descriptions of data.

Related concepts:
- "Data Warehouse Center configuration" in the *Data Warehouse Center Administration Guide*
- "The Cottonwood Distributors, Inc.—a warehouse example" on page 11

Discovering the data—Cottonwood Distributors, Inc.

As Cottonwood Distributors, Incorporated begins to make decisions on their future with their acquired companies, they determine some specific areas of concern.
- The Cottonwood programmers must handle a larger amount of data than they handled before the merger. The data is in different physical locations and in different databases with different formats.
- Cottonwood needs to stay competitive. The company decides to replace the telephone-based sales representatives with a Web-based brokerage system. The users of the Cottonwood Web-based brokerage system will request bids online for parts and place orders online for parts. Cottonwood also has a set of suppliers that need to submit new or updated quotes for parts.

Cottonwood Distributors, Incorporated must determine what data they need for their new expanded company. They must also decide what purposes the data should be used for. Then they must determine where the data sources exist.

Considering the challenges facing Cottonwood Distributors, Inc., IBM® DB2® Information Integrator federated systems and data warehousing are important technologies for them. The federated infrastructure allows a client application to view data in a diverse set of sources as though it is in a single database. Rather than forcing the application programmers to write code that handles multiple application programming interfaces (API), a single SQL statement can combine data from all three of the data sources of Cottonwood. Warehousing provides a data cache close to the servers of Cottonwood. This cache keeps frequently referenced data close to the user.

The Cottonwood data management team is making decisions about how to access their data, and on how to exchange the information they have. The time when a decision is about to be made, and the decision maker, such as Cottonwood, stumbles across data that is relevant to the decision, is the time when that company can derive knowledge from data. The key to Cottonwood's discovery is the knowledge that certain data exists. Cottonwood also knows that the data can be collected, and that the data is relevant to making decisions.

Cottonwood wants to know if a set of its suppliers is quoting different prices for parts to each of the merged companies. The database programmers at Cottonwood need to know where the data sources are that contain the supplier data. They need to know where and how to connect to the data sources. Without the advantages of a federated system, this task requires connections to three databases, with three SQL statements. Additional discovery involves determining information and relationships about the customer and the supplier and the requests that the customers make to the Web-based brokerage system. Knowledge about the customers who have not had their orders filled is useful information to determine optimization techniques or interface improvements.

Related concepts:

- "Business solutions: extending the DB2 Warehouse Manager" on page 184
- "Designing applications for a federated solution—Cottonwood Distributors, Incorporated" on page 175
- "Designing applications—Cottonwood Distributors, Inc. warehouse scenario" on page 186
- "The Cottonwood Distributors, Inc.—a warehouse example" on page 11
- "Deploying the application—Cottonwood Distributors, Inc. solution" on page 189

Related tasks:

- "Developing the application for a federated solution—Cottonwood Distributors, Inc." on page 178

Related reference:

- Appendix A, "Script examples for Cottonwood Distributors, Inc. and YBar, Inc. scenarios," on page 217

Designing applications—Cottonwood Distributors, Inc. warehouse scenario

Cottonwood Distributors, Incorporated, combines the technologies of federated systems and warehousing to solve the problems of data scalability, data accessibility, and data currency.

Cottonwood has a database that contains a table named PART, a table named SUPPLIER, and a table named PARTSUPP.

Table 25. Cottonwood SQL primary tables and columns

PART	SUPPLIER	PARTSUPP
p_partkey (unique value)	s_suppkey	ps_partkey
p_name	s_name	ps_suppkey
p_mfgr	s_address	ps_availqty

Table 25. Cottonwood SQL primary tables and columns (continued)

PART	SUPPLIER	PARTSUPP
p_brand	s_nationkey	ps_supplycost
p_type	s_phone	ps_comment
p_size	s_acctbal	
p_container	s_comment	
p_retailprice		
p_comment		

Cottonwood also has other tables that form relationships with the primary tables to help create solutions for their data management problems.

Table 26. CDI secondary tables and columns

NATION	REGION	CUSTOMER	ORDERS
n_nationkey	r_regionkey	c_custkey	o_orderkey
n_name	r_name	c_name	o_custkey
n_regionkey	r_comment	c_address	o_orderstatus
n_comment		c_nationkey	o_totalprice
		c_phone	o_orderdate
		c_acctbal	o_orderpriority
		c_mktsegment	o_clerk
		c_comment	o_shippriority
			o_comment

The PART table contains information about the identifying key, the manufacturer, the type, the size, and the retail price of each part in the database. The part identifier is unique across the industry. Part number 1234 always identifies widgetA. The SUPPLIER table stores information about the name and address of the supplier, and includes an identifying key. The PARTSUPP table associates the parts with the suppliers. It contains both of the identifiers from the parts and suppliers tables, and the price the suppliers charge for parts.

The tables contains indexes on the data to facilitate the Cottonwood inventory. The database programmers at Cottonwood use views that consist of a union of the three primary tables (see Table 25 on page 186). To get the best information about the Cottonwood suppliers and the prices they quote for parts to each of the merged companies, the database programmers at Cottonwood can use a single SQL statement even though the data is stored in DB2® Universal Database, Informix® and Oracle.

In addition to the data that is contained in the databases, they still have to maintain a useful reporting system that contains XML content.

Cottonwood has a configuration that includes a DB2 Universal Database™ server and a DB2 Universal Database client, a message queue client, and Web clients. On the DB2 server, Cottonwood has a data warehouse, federated systems, stored procedures, XML, and message queue integration. The Web application server contains the quote and bid applications, some Web services, and Java™ Server

Pages (JSP). WebSphere® MQ is also on the server with a listener. Cottonwood programmers need to access data on remote DB2 servers, Oracle, and Informix servers.

Cottonwood can combine federated systems and a data warehouse to address some of the scalability and system load issues. The warehouse contains data from all of the data management systems. The fact that Cottonwood stores the data in different databases is transparent to the application. The warehouse also provides a means to buffer the finely tuned data management systems from the traffic generated by end-user analysts. For example, the PART table is a union over the Informix, Oracle, and DB2 Universal Database PART tables. Furthermore, the references to the tables are the same as they are on the underlying databases. Applications that previously ran on only the DB2 Universal Database can migrate to the data warehouse with few application changes.

The design for the warehouse scenario (see Figure 71 on page 188) involves setting up the warehouse environment, including the warehouse control database. The Cottonwood scenario uses the default data warehouse security group that is created by the Data Warehouse Center. The database programmers at Cottonwood extract the data from the data stores according to some predefined schedules. Then the programmers apply certain business rules to scrub the data. Then, the data is loaded to the data warehouse. Cottonwood divides the data warehouse into a supplier data store, an order data store, and a parts data store. The database programmers at Cottonwood extract defined sets of data from the warehouse into the data marts. The data marts provide access to data that is specifically collected to support analysis, reporting, and measuring. From these data marts, the users can run data mining functions that find trends, relationships, and patterns in the data.

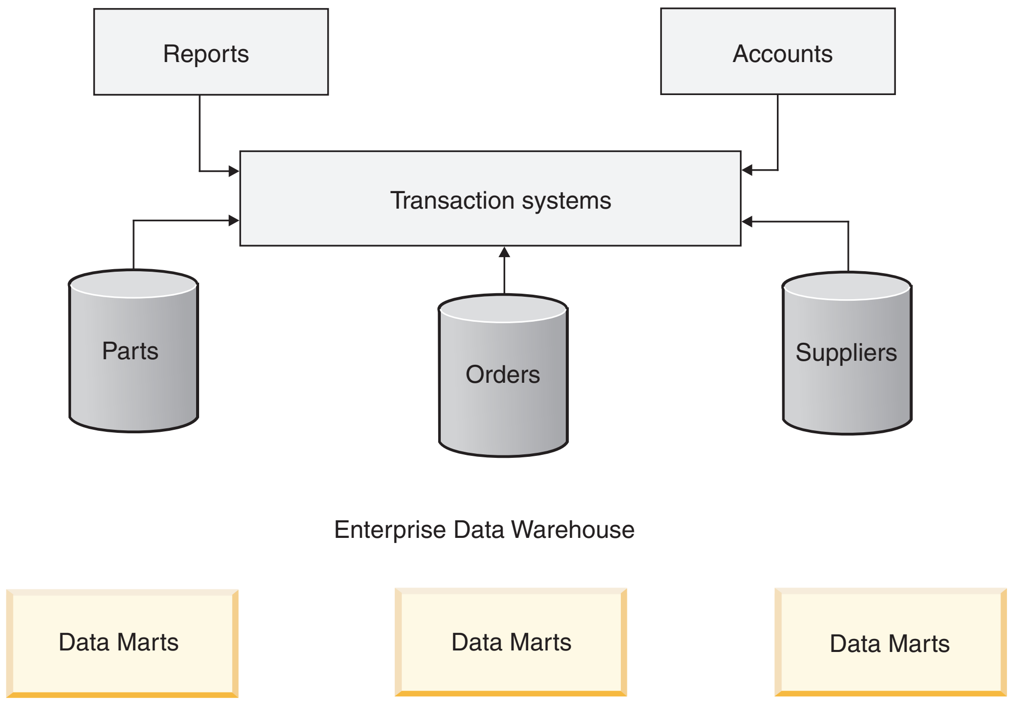


Figure 71. Warehouse configuration for Cottonwood

Related concepts:

- "Designing applications for a federated solution—Cottonwood Distributors, Incorporated" on page 175
- "Deploying the application—Cottonwood Distributors, Inc. solution" on page 189

Deploying the application—Cottonwood Distributors, Inc. solution

After defining the tables, and inserting the data, the database programmers at Cottonwood Distributors, Incorporated can produce information on the available parts and current prices. The database programmers at Cottonwood can use the DB2® Warehouse Manager to help them monitor and administer their integrated systems. DB2 Warehouse Manager allows these programmers to define and run the transformation processes that access the operational federated systems data sources, and then write to the warehouse database. The database systems at Cottonwood can interface with the DB2 Warehouse Manager by using the external trigger capabilities of DB2 Warehouse Manager.

The DB2 Data Warehouse Center graphical interface helps the database programmers at Cottonwood to manage the movement of data from the federated systems operational data to the warehouse. Cottonwood's end-users can use this warehouse. By using the Warehouse Center windows, the database programmers at Cottonwood can manage several warehouse processes. The Cottonwood programmers can schedule these processes to run on a specified schedule.

The Cottonwood programmers can currently run warehouse processes to update the following local tables:
- PART table
- SUPPLIER table
- PARTSUPP table

For the purposes of the Cottonwood Distributors solution, the Cottonwood programmers can improve the access performance by creating federated indexes, or index specifications. See the *Federated Systems Guide* for more information on creating index specifications in the federated database.

The Cottonwood database programmers can use their applications to access both federated and local warehouses. For example, the database programmers at Cottonwood want to build a report that allows them to get the current best price for a customer order. The warehouse currently contains a table with all of the customer orders. The Cottonwood programmers want to give their best customers the best price. They also want to keep track of the changes in prices made by the suppliers. The database programmers at Cottonwood perform the following actions:
1. Create a report that selects data from the warehouse customer order table.
2. Select the prices from the federated sources to compare the prices to see if the price is any lower since the last warehouse update.
3. If the price is lower, the Cottonwood programmers selectively inform their best customers that the costs are reduced.

Related concepts:
- "Designing applications for a federated solution—Cottonwood Distributors, Incorporated" on page 175

- "Designing applications—Cottonwood Distributors, Inc. warehouse scenario" on page 186
- "Federated development scenario—Cottonwood Distributors, Inc." in the *DB2 Information Integrator Solutions Guide*
- "Discovering the data—Cottonwood Distributors, Inc." on page 185
- "Performance and tuning planning— materialized query tables in a federated system" on page 22

Related tasks:

- "Developing the application for a federated solution—Cottonwood Distributors, Inc." on page 178

Developing database applications that use WebSphere Message Queue functions

IBM recognizes that Web services provide not just an architecture for integrating applications, but also an architecture for integrating data. And DB2 provides the capability of managing data and providing intelligent, optimized access to data.

Installing DB2 WebSphere MQ functions

The DB2 WebSphere MQ functions are available in DB2 Universal Database as user-defined functions. These functions, allow users to access the WebSphere MQ queues from DB2 UDB objects.

Prerequisites:

1. Make sure that your DB2 UDB installation added the following libraries:
   - UNIX platform: *libdb2qgmq* and *libdb2mqsw* in directory sqllib/lib
   - Windows platform: *db2qgmq.dll* and *db2mqsw.dll* in directory sqllib\bin
2. Install the Application Messaging Interface (AMI), Version 1.2.4.or later, by using the AMI installation image that is shipped with DB2 Universal Database Version 8 or later.
3. Add the AMT_DATA_PATH environment variable to the list that is used by DB2 UDB to ensure that the message queuing user-defined functions (MQ UDFs) execute correctly. You can edit the file *$INSTHOME/sqllib/profile.env* (UNIX) or *%DB2PATH%\profile.env* (Windows), and add AMT_DATA_PATH to DB2ENVLIST. You can also use the db2set command:

   ```
   db2set DB2ENVLIST="AMT_DATA_PATH"
   ```

   Restart the database instance for the environment variable changes to take effect.
4. Install DB2 Universal Database XML Extender, available as an option in DB2 Universal Database Version 8.
5. Create the system default AMI objects by using the `amtsamp.tst` file if you plan to use publish and subscribe functions.

The basic steps for configuring, and enabling the DB2 WebSphere MQ functions are as follows:

1. Install WebSphere MQ Version 5.3, Corrective Service Diskette 05 (CSD05) or a later release.
2. Install the following SupportPacs from the Support page located at http://www.ibm.com/software/ts/mqseries/:

- WebSphere MQ Application Messaging Interface
- WebSphere MQ SupportPacs that include the publish or subscribe capabilities

3. If you want to use transactional MQ UDFs, make sure that you configure the database for federated operations. Do this with the following command

   ```
   update dbm cfg using federated yes
   ```
4. Enable the DB2 UDB MQ Functions (see Step 1 on page 191 in the **Procedure**

Restrictions:

- The DB2 UDB MQ transactional functions that exist under schema **db2mq1c** do not support CLOB type messages.
- The enable utility of the transactional MQ user-defined functions allows only 40 AMI Policies to exist in an AMI repository file for the queue manager specified with the -q option.
- The transactional MQ user-defined functions support only one Queue Manager within a single transaction. The queue manager that you specify in Service and Policy (through connection in amthost.xml) must match. If you leave the Queue Manager blank in the service point, WebSphere MQ defaults to the manager designated by Policy. There is a default set of MQ queues and a default Queue Manager that is normally created during the MQ installation and the enable_MQFunctions processes.
- You must create the queues and WebSphere MQ objects prior to using them within SQL statements.
- If you use the publish and subscribe functions, you need to create certain WebSphere MQ objects prior to using them with SQL statements. You can do this by issuing the MQSC commands through the *.tst files provided by WebSphere MQ and AMI (amtsamp.tst and amtsdfts.tst). To do this, use the following steps:
  1. Make sure that you have the amtsamp.tst and amtsdfts.tst files
  2. Update the *.tst files for your queue manager, if necessary
  3. Start the queue manager that is used by your AMI service
  4. Issue a command similar to the following command:

     ```
     runmqsc QMName <amtsamp.tst
     ```

Procedure:

You use the commands, **enable_MQFunctions** and **disable_MQFunctions** for transactional and nontransactional MQ user-defined functions. The MQ user-defined functions are defined as a group or set under different schema names. The groups that do not support transactions have schema **db2mq**. The groups that do support transactions have schema **db2mq1c**. The **enable_MQFunctions** command with the options that support transactions, allows you to select a set of MQ user-defined functions to install or uninstall for transactional support. To use the **enable_MQFunctions** see the complete command syntax for enable_MQFunctions and disable_MQFunctions.

1. Configure and enable a database for the WebSphere MQ functions. The enable_MQFunctions utility is a flexible command. It first checks that you have properly set up the WebSphere MQ environment. It then installs and creates a default configuration for the WebSphere MQ functions. Then, it enables the specified database with these functions, and confirms that the configuration works.

The following examples assume that the user is connected to the database SAMPLE.

Example 1: Enable the transactional and nontransactional user-defined functions:

```
enable_MQFunctions -n sample -u user1 -p password1
```

Example 2: Create DB2MQ1C functions under schema DB2MQ1C:

```
enable_MQFunctions -n sample -u user1 -p password1 -v 1pc
```

2. Test the MQ functions by using the Command Line Processor (on a Windows environment). Issue the following commands after you connect to the currently enabled database:

   ```
   values DB2MQ1C.MQSEND('a test')
   values DB2MQ1C.MQRECEIVE()
   ```

   The first statement sends the message `a test` to the DB2MQ_DEFAULT_Q queue. The second statement receives it back. This statement assumes that you have used some default configuration. You can use both of these statements in a DB2 transaction that you can commit or roll back as part of the unit of work.

Related concepts:

- "Asynchronous messaging in DB2 Information Integrator" on page 204
- "How to use WebSphere MQ functions within DB2" on page 200

Related tasks:

- "Configuring WebSphere MQ for MQListener" on page 209
- "Configuring MQListener" on page 210
- "Configuring MQListener to run in the DB2 Universal Database environment" on page 207
- "Configuring and running MQListener" on page 207

Related reference:

- "enable_MQFunctions" in the *Command Reference*
- "disable_MQFunctions" in the *Command Reference*

Overview of WebSphere MQ and DB2 application integration

Use DB2® Universal Database and WebSphere® MQ to create SQL requests, develop stored procedures, extend the database with user-defined functions, and turn database requests into Web services. Messaging, queuing, and publishing and subscribing are common technologies within database application environments. These techniques help link together disparate applications, disseminate real-time information and integrate data and communication within the enterprise.

WebSphere MQ is a message handling system that enables applications to communicate in a distributed environment across different operating systems and networks. WebSphere MQ handles the communication from one program to another by using application programming interfaces (APIs).

The Application Messaging Interface (AMI) is a commonly used API for WebSphere MQ that is available in a number of high-level languages. In addition to the AMI, DB2 UDB provides its own application programming interface to the

WebSphere MQ messaging system through a set of external user-defined functions, called DB2 WebSphere MQ functions. Using these functions in SQL statements allows you to combine DB2 Universal Database™ access with WebSphere MQ message handling.

Introduction to message handling and AMI

The WebSphere MQ message handling system takes a piece of information (the message) and sends it to its destination. WebSphere MQ guarantees delivery despite any network disruptions that might occur.

Applications programmers use the AMI to send messages and to receive messages. The three components in the AMI are:

- The *message*, which defines **what** one program sends to another
- The *service*, which defines **where** the message is going to or coming from
- The *policy*, which defines **how** to handle the message

To send a message that uses the AMI, an application must specify the message data, the service, and the policy. A system administrator defines the WebSphere MQ configuration that is required for a particular installation, including the default service and default policy. DB2 UDB provides the default service, and default policy, DB2.DEFAULT.SERVICE and DB2.DEFAULT.POLICY, which application programmers can use to simplify their programs.

For detailed information about the AMI, see *MQSeries Application Messaging Interface*.

WebSphere MQ messages

WebSphere MQ uses messages to pass information between applications. Messages consist of the following parts:

- The message attributes, which identify the message and its properties. The AMI uses the attributes and the policy to interpret and construct MQSeries® headers and message descriptors.
- The message data, which is the application data that is carried in the message. The AMI does not act on this data.

Attributes are properties of an AMI message. With the AMI, the message can contain the attributes, or a system administrator can define the attributes in a default policy. The application programmer is not concerned with the details of message attributes.

WebSphere MQ services

A service describes a destination to which an application sends messages or from which an application receives messages. WebSphere MQ calls a destination a message queue, and a queue resides in a queue manager.

Applications can put messages on queues or get messages from them by using the AMI. A system administrator sets up the parameters for managing a queue, which the service defines. Therefore, AMI hides the complexity from the application programmer. An application program selects a service by specifying it as a parameter for DB2 MQSeries function calls.

WebSphere MQ policies

A policy controls how the AMI functions handle messages. Policies control such items as:

- The attributes of the message, for example, the priority
- Options for send and receive operations, for example, whether an operation is part of a unit of work

The AMI provides default policies. Alternatively, a system administrator can define customized policies and store them in a repository. An application program can specify a policy as a parameter for DB2 MQSeries function calls.

Capabilities of DB2 MQSeries functions

There are two uses of the MQSeries functions with DB2:

- User-defined functions with no transactional semantics (schema name is DB2MQ)
- User-defined functions that use one-phase commit semantics (schema name is DB2MQ1C)

The DB2 WebSphere MQ functions support the following types of operations:

- Send and forget, where messages need no reply
- Read, where the application can read one or all messages without removing them from the queue
- Receive, where the application can receive and remove one or all messages from the queue
- Request and response, where a sending application needs a response to a request

You can use the DB2 WebSphere MQ functions to send messages to a message queue or to receive messages from the message queue. In addition, you can send a request to a message queue and receive a response.

You should locate the WebSphere MQ server on the same system as the DB2 database server. You register the DB2 WebSphere MQ functions with the DB2 database server and provide access to the WebSphere MQ server by using the AMI. For information about installing the DB2 MQSeries functions, see Installing DB2 WebSphere MQ functions.

The DB2 WebSphere MQ functions include both scalar functions and table functions. Remember that the schema name indicates the type of user-defined function. Refer to Setting up DB2 WebSphere MQ functions for more information on the MQ functions. The following definitions describe the DB2 WebSphere MQ scalar functions.

MQREAD
This returns a message in a VARCHAR variable from the MQSeries location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the message from the head of the queue but instead returns it. If no messages are available to be returned, a null value is returned.

```
>>--MQREAD--(--+--------------------------------------+--)------------><
               '-receive-service--+-----------------+-'
                                  '-,--service-policy-'
```

MQRECEIVE
This returns a message in a VARCHAR variable from the MQSeries

location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the message from the queue. If *correlation-id* is specified, the first message with a matching correlation identifier is returned; if *correlation-id* is not specified, the message at the head of queue is returned. If no messages are available to be returned, a null value is returned.

```
►►──MQRECEIVE──────────────────────────────────────────────────────────────►

►─(─┬────────────────────────────────────────────────────────┬─)────────►◄
    └─receive-service─┬────────────────────────────────────┬─┘
                      └─,─service-policy─┬───────────────┬─┘
                                         └─,─correl-id───┘
```

MQSEND

This sends the data in a VARCHAR variable *msg-data* to the MQSeries location specified by *send-service*, using the policy defined in *service-policy*. An optional user-defined message correlation identifier can be specified by *correlation-id*. The return value is 1 if successful or 0 if not successful.

```
►►──MQSEND──(─┬──────────────────────────────────────────┬─msg-data───────►
              └─send-service─,─┬──────────────────────┬──┘
                               └─service-policy─,─────┘

►─┬────────────────────┬─)────────────────────────────────────────────────►◄
  │                (1) │
  └─,─correl-id────────┘
```

Notes:

1 The *correl-id* cannot be specified unless a *service* and a *policy* are also specified.

You can send or receive messages in VARCHAR variables for schemas DB2MQ and DB2MQ1C. The maximum length for a DB2MQ message in a VARCHAR variable is 4,000 bytes long. DB2MQ1C supports a VARCHAR of up to 32,000 bytes long.

The following definitions describe the DB2 MQSeries table functions.

MQREADALL

This returns a table that contains the messages and message metadata in VARCHAR variables from the MQSeries location specified by *receive-service*, using the policy defined in *service-policy*. This operation does not remove the messages from the queue. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.

```
►►──MQREADALL──(─┬─────────────────────────────────────────┬──────────────►
                 └─receive-service─┬────────────────────┬──┘
                                   └─,─service-policy───┘

►─┬──────────┬─)──────────────────────────────────────────────────────────►◄
  └─num-rows─┘
```

MQRECEIVEALL

This returns a table that contains the messages and message metadata in VARCHAR variables from the MQSeries location specified by *receive-service*, using the policy defined in *service-policy*. This operation removes the messages from the queue. If *correlation-id* is specified, only those messages with a matching correlation identifier are returned; if *correlation-id* is not specified, all available messages are returned. If *num-rows* is specified, a maximum of *num-rows* messages is returned; if *num-rows* is not specified, all available messages are returned.

```
►►──MQRECEIVEALL──(────────────────────────────────────────────────────────►

►─┬──────────────────────────────────────────────────────────┬───────────────►
  └─receive-service─┬──────────────────────────────────────┬─┘
                    └─,──service-policy─┬────────────────┬─┘
                                        └─,──correl-id───┘

►─┬──────────────────┬─)────────────────────────────────────────────────────►◄
  └─┬───┬──num-rows──┘
    └─,─┘
```

You can send or receive messages in VARCHAR variables. The maximum length for a message in a DB2MQ VARCHAR variable is 4000 bytes. The maximum length for a message in a DB2MQ1C variable is a VARCHAR 32,000 bytes. The first column of the result table of a DB2 MQSeries table function contains the message.

By using DB2 scalar and table functions along with views with Information Integrator, you can incorporate message handling operations in SQL queries from any environment. If you have a WebSphere MQ client or server, you can use the messaging operations within SQL statements. For example:

```
SELECT DB2MQ1C.MQSend ('MyAddress'|| firstname ||' '|| lastname)
  FROM employee
```

Publishing and subscribing messages gives you more control over which services should receive messages. Publish and subscribe systems provide a scalable, secure environment in which many subscribers can register to receive messages from multiple publishers. You can use the trigger facility within DB2 Universal Database to automatically publish messages as part of a trigger invocation.

MQPUBLISH

This function publishes data to MQSeries. This function requires the installation of either MQSeries Publish/Subscribe or MQSeries Integrator.

```
►►──MQPUBLISH──(─┬───────────────────────────────────────────────┬──────────►
                 └─publisher-service──,─┬──────────────────────┬─┘
                                        └─service-policy──,────┘

►─msg-data─┬──────────────────────────────────┬─)──────────────────────────►◄
           └─,──topic─┬──────────────────────┬┘
                      │            (1)       │
                      └─,──correl-id─────────┘
```

Notes:

1 The *correl-id* cannot be specified unless a *service* and a *policy* are also specified.

MQSUBSCRIBE

This function registers interest in MQSeries messages that are published on a specified topic. The subscriber-service specifies a logical destination for messages that match the specified topic. Messages that match topic will be placed on the queue defined by subscriber-service and can be read or received through a subsequent call to MQREAD, MQRECEIVE, MQREADALL, or MQRECEIVEALL. This function requires the installation and configuration of an MQSeries based publish and subscribe system, such as MQSeries Integrator or MQSeries Publish/Subscribe.

```
►►──MQSUBSCRIBE──(──┬────────────────────────────────────────────┬──────►
                    └─subscriber-service──,──┬─────────────────────┬─┘
                                             └─service-policy──,─┘

►─topic──)─────────────────────────────────────────────────────────────►◄
```

MQUNSUBSCRIBE

This function is used to unregister an existing message subscription. The subscriber-service, service-policy, and topic are used to identify which subscription is canceled. This function requires the installation and configuration of an MQSeries based publish and subscribe system, such as MQSeries Integrator or MQSeries Publish/Subscribe.

```
►►──MQUNSUBSCRIBE──(──┬────────────────────────────────────────────┬────►
                      └─subscriber-service──,──┬─────────────────────┬─┘
                                               └─service-policy──,─┘

►─topic──)─────────────────────────────────────────────────────────────►◄
```

An example of simple data publication is when one application notifies other applications about events of interest. The application does this by sending a message to a queue that is monitored by another application. The contents of the message might be either a user-defined string, composed from database columns, or a string-valued function call, or any valid expression that yields a string of the correct type.

If you need more control over which services should receive any particular message, then you need to use the publish and subscribe functions. Many subscribers can register to receive messages from multiple publishers. You can specify a topic that you can associate with your message. For example, a DB2 application can publish a message to the service point *Weather*. The message is *Sleet*, and the topic is *Austin*.

```
values DB2MQ1C.MQPublish ('Weather Bulletins','Sleet','Austin')
```

This notifies the interested subscribers that the weather in Austin is sleet. Subscribers register an interest in receiving this kind of information with the following statement:

```
values DB2MQ1C.MQSUBSCRIBE('aSubscriber', 'Austin')
```

When the subscriber is no longer interested in subscribing to a particular topic, that subscriber must explicitly unsubscribe using a statement such as:

```
values DB2MQ1C.MQUNSUBSCRIBE('aSubscriber','Austin')
```

Commit environment for DB2 WebSphere MQ functions

A transaction is commonly referred to in DB2 Universal Database as a unit of work. A unit of work is a recoverable sequence of operations within an application process. It is used by the database manager to ensure that a database is in a consistent state. Any reading from or writing to the database is done within a unit of work. A unit of work starts when the first SQL statement is issued on the database. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. DB2 Universal Database provides two versions of commit when you use DB2 WebSphere user-defined functions:

- A non-transactional UDF with a schema name of DB2MQC
- A single-phase commit with a schema name of DB2MQ1C

The commit environment for MQ user-defined functions is also dependent on the type of CONNECT that your application includes. The CONNECT statement establishes a connection between an application process and its server. A type 1 CONNECT supports the single database per unit of work (Remote Unit of Work) semantics. A type 2 CONNECT supports the multiple databases per unit of work (Application-Directed Distributed Unit of Work) semantics. You can also specify a SYNCPOINT of ONEPHASE. A SYNCPOINT defines how COMMITs or ROLLBACKs are coordinated among multiple database connections. With a SYNCPOINT of ONEPHASE, updates can only occur against one database in the unit of work, and all other databases are read-only.

Non-transactional functions—schema DB2MQ: If your application uses non-transactional user-defined functions, any DB2 COMMIT or ROLLBACK operations are independent of the WebSphere MQ operations. If you roll back a transaction, the MQ functions do not discard the messages that you sent to a queue within the current unit of work.

In this environment, WebSphere MQ controls its own queue operations. A DB2 COMMIT or ROLLBACK does not affect when or if your application adds or deletes messages to or from an WebSphere MQ queue.

Single-phase commit—schema DB2MQ1C: If your application uses one-phase commit over your data sources, and a transaction is rolled back, the application might discard the message, or produce an error. This can result in an inconsistent state. The rules for one-phase commit with SYNCPOINT=ONEPHASE are as follows:

- Updates are allowed only for one data source
- Messaging functions cannot be combined with other updates

Table 27. DB2 MQ user-defined function semantics

Connection type	One-phase commit (schema name=DB2MQ1c)
Type 1 (ONEPHASE)	`select db2mq1c.mqsend (e.LASTNAME || ' ' || d.DEPTNAME) from EMPLOYEE e, DEPT d where e.DEPARTMENT = d.DEPTNAME` An application can select and send; it can update only one data source
Type 2 (TWOPHASE)	Message functions not allowed

When the DB2 MQ function is part of the DB2 unit of work: You can use DB2 MQ UDFs as part of the DB2 unit of work, or transaction in many kinds of DB2 operations.

Multiple connections

This describes a scenario where two users connect to the same database. They execute the DB2 MQ UDFs. One connection sends a message. The other connection receives. The second connection does not see the message of the first connection before the first connection commits. The second connection sees the messages of the first connection after the commit. If the first connection issues a roll back, the second connection does not see the message.

Table 28. Two users connecting to the same database

Connection 1	Connection 2
db2 +c // Turn auto commit off	
values db2mq1c.mqsend (“test message”)	

	`//The connection can not`
	`//see the`
	`//message yet:`
	`values db2mq1c.mqreceive();`

commit;	

	`//Now the connection`
	`//can see the message:`
	`values db2mq1c.mqreceive();`

Triggers

DB2 MQ UDFs can be part of a single or a multiple statement BEFORE or AFTER trigger.

```
create table EMPLOYEE
  (NAME VARCHAR(30), LASTNAME VARCHAR(30) NOT NULL PRIMARY KEY);

create trigger AFTER_TEST
    after insert on EMPLOYEE
    referencing NEW as NEWEMP
    for each row mode DB2SQL
    VALUES db2mq.mqsend(newemp.lastname);

insert into EMPLOYEE values ('MORGAN', 'TONG');

create trigger BEFORE_TEST
    no cascade before update of NAME on EMPLOYEE
    referencing NEW as NEWNAME OLD as OLDNAME
    for each row mode db2sql
    values db2mq.mqsend (oldname.lastname);

update EMPLOYEE set NAME = 'RAY';
```

.

Restrictions:

You can integrate the messaging techniques with database operations on most DB2 SQL statements. If using a DB2 MQ function results in an error, DB2 Universal Database automatically rolls the transaction back. Here are some examples of statements that cannot use DB2 MQ functions:

- If a user issues an application savepoint.
- If a user tries to use the DB2 MQ UDFs from within an atomic compound SQL statement

Related reference:

- "CONNECT (Type 1) statement" in the *SQL Reference, Volume 2*
- "CONNECT (Type 2) statement" in the *SQL Reference, Volume 2*
- "SET CLIENT Command" in the *Command Reference*
- "MQSEND scalar function" in the *SQL Administrative Routines*
- "MQRECEIVE scalar function" in the *SQL Administrative Routines*
- "MQREAD scalar function" in the *SQL Administrative Routines*
- "MQPUBLISH scalar function" in the *SQL Administrative Routines*
- "MQSUBSCRIBE scalar function" in the *SQL Administrative Routines*
- "MQUNSUBSCRIBE scalar function" in the *SQL Administrative Routines*
- "MQREADALL table function" in the *SQL Administrative Routines*
- "MQRECEIVEALL table function" in the *SQL Administrative Routines*

How to use WebSphere MQ functions within DB2

WebSphere® MQ and DB2® message operations combine database operations in a single unit of work as an atomic transaction. The most basic form of messaging with the DB2 MQ functions occurs when all database applications connect to the same DB2 UDB server. Clients can be local to the database server or distributed in a network environment.

In a simple scenario, client A invokes the MQSEND function to send a user-defined string to the location that is defined by the default service. DB2 UDB executes the MQSeries® functions that perform this operation on the database server. At some later time, client B invokes the MQRECEIVE function. This removes the message at the head of the queue that is defined by the default service. The function then returns it to the client. DB2 UDB executes the MQSeries functions that perform this operation on the database server.

Database clients can use simple messaging in a number of ways:

- Data collection

  The application receives Information in the form of messages from one or more sources. An information source can be any application. The application receives the data from queues and stores the data in database tables for additional processing.
- Workload distribution

  The application posts work requests to a queue that is shared by multiple instances of the same application. When an application instance is ready to perform some work, it receives a message that contains a work request from the head of the queue. Multiple instances of the application can share the workload that is represented by a single queue of pooled requests.
- Application signaling

  In a situation where several processes collaborate, you can use messages to coordinate their efforts. These messages might contain commands or requests to perform work. For more information about this technique, see Application-to-application connectivity.

The following scenario extends basic messaging to incorporate remote messaging. Assume that machine A sends a message to machine B.

1. The DB2 UDB client executes an MQSEND function call, specifying a target service that has been defined to be a remote queue on machine B.

2. The MQSeries functions perform the work to send the message. The MQSeries server on machine A accepts the message. The server guarantees that it will deliver the message to the destination. The service and the current configuration of machine A defines the destination. The server determines that the destination is a queue on machine B. The server then attempts to deliver the message to the MQSeries server on machine B, retrying as needed.
3. The MQSeries server on machine B accepts the message from the server on machine A and places it in the destination queue on machine B.
4. An MQSeries client on machine B requests the message at the head of the queue.

When you use MQSEND, you choose what data to send, where to send it, and when to send it. This type of messaging is called *send and forget*. The sender only sends a message, relying on MQSeries to ensure that the message reaches its destination.

The following examples use the DB2MQ1C schema for one-phase commit, with the default service DB2.DEFAULT.SERVICE and the default policy DB2.DEFAULT.POLICY. For more information about one-phase commit, see Overview of WebSphere MQ and DB2 application integration. All of the examples assume that auto commit is off. Therefore, a COMMIT is needed. Without the COMMIT, you might still be holding locks until the end of the transaction.

Example: The following CREATE TRIGGER statement sends a message that consists of the first and last names that are inserted into table employees:

```
 CREATE TRIGGER T1
 AFTER INSERT ON employee REFERENCING new AS newemp
    FOR EACH ROW MODE DB2SQL
   VALUES DB2MQ.MQSEND(newemp.name)
```

Example: Assume that you have an EMPLOYEE table, with VARCHAR columns LASTNAME, FIRSTNAME, and DEPARTMENT. Also assume that auto commit is turned off. To send a message that contains this information for each employee in DEPARTMENT 5LGA, issue the following SQL SELECT statement:

```
SELECT DB2MQ1C.MQSEND (LASTNAME || ' ' || FIRSTNAME || ' ' || DEPARTMENT)
  FROM EMPLOYEE WHERE DEPARTMENT = '5lGA';
COMMIT;
```

Message content can be any combination of SQL statements, expressions, functions, and user-specified data. Because this MQSEND function uses DB2 MQ transactional user-defined functions with one-phase commit semantics, the COMMIT statement ensures that the message is added to the MQSeries queue.

The DB2 MQSeries functions allow an application to read or receive messages. The difference between reading and receiving is that reading returns the message at the head of a queue without removing it from the queue. Receiving causes the message to be removed from the queue. A message that is retrieved using a receive operation can be retrieved only once. A message that is retrieved using a read operation allows the same message to be retrieved many times.

The following examples use the DB2MQ1C schema for one-phase commit, with the default service DB2.DEFAULT.SERVICE and the default policy DB2.DEFAULT.POLICY. For more information about one-phase commit, see Overview of WebSphere MQ and DB2 application integration.

Example: The following SQL SELECT statement reads the message at the head of the queue that is specified by the default service and policy. Assume that autocommit is turned off.

```
SELECT DB2MQ1C.MQREAD() FROM SYSIBM.SYSDUMMY1;
COMMIT;
```

You invoke the MQREAD function once because SYSIBM.SYSDUMMY1 has only one row. The SELECT statement returns a VARCHAR(32000) string. If no messages are available to be read, the result of the statement is a null value.

Example: The following SQL SELECT statement materializes the contents of a queue as a DB2 UDB table:

```
SELECT T.* FROM TABLE(DB2MQ1C.MQREADALL()) T;
```

The result table T of the table function consists of all the messages in the queue and the metadata about those messages. The queue is defined by the default service. The first column of the materialized result table is the message itself, and the remaining columns contain the metadata. The SELECT statement returns both the messages and the metadata.

To return only the messages, issue the following statement:

```
SELECT T.MSG FROM TABLE(DB2MQ1C.MQREADALL()) T;
```

The result table T of the table function consists of all the messages in the queue and the metadata about those messages. The queue is defined by the default service This SELECT statement returns only the messages.

Example: The following SQL SELECT statement tries to send the message from the queue. Assume that auto commit is turned off.

```
 SELECT DB2MQ1C.MQSEND(name) FROM employees e;
 ROLLBACK;
```

The ROLLBACK statement means that the message is not actually sent because it is in the same unit of work as the DB2 UDB operation.

Example: The following SQL SELECT statement gets all of the messages from the default service queue.

```
SELECT t.msg FROM table(DB2MQ1C.MQRECEIVEALL()) t;
COMMIT;
```

The result table T of the table function consists of all the messages in the default service queue and the metadata about those messages. The SELECT statement returns only the messages.

Related concepts:

- "Overview of WebSphere MQ and DB2 application integration" on page 192

Related tasks:

- "Installing DB2 WebSphere MQ functions" on page 190

Application-to-application connectivity

You typically use application-to-application connectivity to solve the problem of putting together a diverse set of application subsystems. To facilitate application integration, MQSeries® provides the means to interconnect applications. This section describes one common scenario, called *request-and-reply* communication.

The request-and-reply method enables one application to request the services of another application. One way to do this is for the requester to send a message to the service provider to request that some work be performed. When the provider completes the work, the provider might decide to send results, or just a confirmation of completion, back to the requester. Unless the requester waits for a reply before continuing, MQSeries must provide a way to associate the reply with its request.

MQSeries provides a correlation identifier to correlate messages in an exchange between a requester and a provider. The requester marks a message with a known correlation identifier. The provider marks its reply with the same correlation identifier. To retrieve the associated reply, the requester provides that correlation identifier when receiving messages from the queue. The provider returns the first message with a matching correlation identifier to the requester.

The following examples use the DB2MQ1C schema for single-phase commit. For more information about single-phase commit, see "Commit environment for DB2 WebSphere MQ functions" on page 197.

Example: The following SQL SELECT statement sends a message consisting of the string "`Msg with corr id`" to the service, MYSERVICE. It uses the policy, MYPOLICY with a correlation identifier CORRID1:

```
SELECT DB2MQ1C.MQSEND ('MYSERVICE', 'MYPOLICY', 'Msg with corr id', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
COMMIT;
```

You invoke the MQSEND function once because SYSIBM.SYSDUMMY1 has only one row. Because this MQSEND uses the DB2MQ1C schema, which is the one-phase commit UDF, the message is part of the DB2® transaction.

Example: The following SQL SELECT statement receives the first message that matches the identifier CORRID1. It receives the message from the queue that is specified by the service, MYSERVICE. It uses the policy, MYPOLICY:

```
SELECT DB2MQ1C.MQRECEIVE ('MYSERVICE', 'MYPOLICY', 'CORRID1')
  FROM SYSIBM.SYSDUMMY1;
```

The SELECT statement returns a VARCHAR(32000) string. If no messages are available with this correlation identifier, the result of the statement is a null value, and the queue does not change.

You can use WebSphere® MQSeries user-defined functions that are available in XML Extender to pass only XML messages between DB2 and the various WebSphere MQSeries implementations. First, you enable the database for XML extender. Then, you enable the MQSeries XML Extender functions in the following way:

```
enable_MQXML -n DATABASE -u USER -p PASSWORD
```

The following table is a brief description of some of the MQSeries XML functions. These functions have a DB2XML database schema. They are not under MQ UDF transactional control.

Table 29. MQSeries XML functions

MQSeries XML Functions	Description
DB2XML.MQSendXML	Send an XML message to the queue.
DB2XML.MQReadXML	A nondestructive read of matching XML message(s) from the queue.
DB2XML.MQReadAllXML	A nondestructive read of all XML messages from the queue
DB2XML.MQReadXMLCLOB	A nondestructive read of matching XML CLOB message(s) from the queue.
DB2XML.MQReadAllXMLCLOB	A nondestructive read of all XML CLOB messages from the queue

Related concepts:

- "MQSeries Enablement" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "MQReadAllXML function" in the *DB2 XML Extender Administration and Programming*
- "MQReadXMLCLOB function" in the *DB2 XML Extender Administration and Programming*
- "MQReadAllXMLCLOB function" in the *DB2 XML Extender Administration and Programming*
- "MQSENDXML function" in the *DB2 XML Extender Administration and Programming*

Asynchronous messaging in DB2 Information Integrator

Programs can communicate with each other by sending data in messages rather than using constructs like synchronous remote procedure calls. With asynchronous messaging, the program that sends the message proceeds with its processing after sending the message without waiting for a reply. If the program needs information from the reply, the program suspends processing and waits for a reply message. If the messaging programs use an intermediate queue that holds messages, the requester program and the receiver program do not need to be running at the same time. The requester program places a request message on a queue and then exits. The receiver program retrieves the request from the queue and processes the request.

Asynchronous operations require that the service provider is capable of accepting requests from clients without notice. An asynchronous listener is a program that monitors message transporters, such as WebSphere® MQ, and performs actions based on the message type. An asynchronous listener can use WebSphere MQ to receive all messages that are sent to an endpoint. An asynchronous listener can also register a subscription with a publish or subscribe infrastructure to restrict the messages that are received to messages that satisfy specified constraints.

The following examples show some common uses of asynchronous messaging:

Message accumulator
You can accumulate the messages that are sent asynchronously so that the

listener checks for messages and stores those messages automatically in a database. This database, which acts as a message accumulator, can save all messages for a particular endpoint, such as an audit trail. The asynchronous listener can subscribe to a subset of messages, such as *save only high value stock trades*. The message accumulator stores entire messages, and does not provide for selection, transformation, or mapping of message contents to database structures. The message accumulator does not reply to messages.

Message event handler
The asynchronous event handler listens for messages and invokes the appropriate handler (such as a stored procedure) for the message endpoint. You can call any arbitrary stored procedure. The asynchronous listener lets you select, map, or reformat message contents for insertion into one or more database structures.

The following lists some of the benefits of using asynchronous messaging database interactions:

- The client and database do not need to be available at the same time. If the client is available intermittently, or if the client fails between the time the request is issued and the response is sent, it is still possible for the client to receive the reply. Or, if the client is on a mobile computer and becomes disconnected from the database, and if a response is sent, the client can still receive the reply.
- The content of the messages in the database contain information about when to process particular requests. The messages in the database use priorities and the request contents to determine how to schedule the requests.
- An asynchronous message listener can delegate a request to a different node. It can forward the request to a second computer to complete the processing. When the request is complete, the second computer returns a response directly to the endpoint that is specified in the message.
- An asynchronous listener can respond to a message from a supplied client, or from a user-defined application. The number of environments that can act as a database client is greatly expanded. Clients such as factory automation equipment, pervasive devices, or embedded controllers can communicate with DB2® Universal Database either directly through WebSphere MQ or through some gateway that supports WebSphere MQ.

Related concepts:

- "MQListener in DB2 Information Integrator" on page 205

Related tasks:

- "Configuring and running MQListener" on page 207

Related reference:

- "db2mqlsn - MQ Listener Command" in the *Command Reference*
- "Parameters used in MQListener configuration" on page 214

MQListener in DB2 Information Integrator

IBM® DB2® Information Integrator provides an asynchronous listener, named MQListener. MQListener is a framework for tasks that read from WebSphere® MQ queues and call DB2 Universal Database™ stored procedures with messages as they arrive.

MQListener combines messaging with database operations. You can configure the MQListener daemon to listen to the WebSphere MQ message queues that you specify in a configuration database. MQListener reads the messages that arrive from the queue and then calls DB2 UDB stored procedures with the messages as input parameters. If the message requires a reply, MQListener creates a reply from the output that is generated by the stored procedure. The message retrieval order is fixed at the highest priority first, and then within the priority the first message in is the first message served.

MQListener runs as a single multi-threaded process. Each thread or task establishes a connection to its configured message queue for input. Each task also connects to a DB2 UDB database on which to run the stored procedure. The information about the queue and the stored procedure is stored in a table in the configuration database. The combination of the queue and the stored procedure is a task.

MQListener tasks are grouped together into named configurations. By default, the configuration name is empty. If you do not specify the name of a configuration for a task, MQListener uses the configuration with an empty name.

MQListener can integrate the message queue read and write operations with the stored procedure into a single transaction. When you run transactional tasks a message cannot be lost even if your computer fails after you read the message from the queue, but before the stored procedure receives the message. By default, only the call to the stored procedure is transactional. If you want to combine into the same transaction the operations of removing the message from the queue and calling the stored procedure, configure the WebSphere MQ environment as a coordinator by using the *–mqcoordinated* parameter with the db2mqlsn command. You must configure the pertinent queue manager to coordinate with the proper resource according to WebSphere MQ guidelines. If you do not want to specify transactional queue operations, the queue manager should not be configured as a transaction manager. Do not run a nontransactional task with a queue manager that is configured as a transaction coordinator.

As part of the MQListener configuration, you specify the configuration user (-configUser) and the run user (-dbUser). The **configuration userconfiguration user** and the **run userrun user** can be separate users with different access rights. The run user does not inherit the privileges of the configuration user. In a normal MQListener scenario, a user runs the MQListener application. The only right that the user who runs MQListener requires is the ability to access WebSphere MQ functions, which generally means being a member of the *mqm* group in Windows® and UNIX® operating systems. The user who executes MQListener is typically the configuration user.

The stored procedure interface for MQListener takes the incoming message as input and returns the reply, which might be NULL, as output:

```
schema.proc(in inMsg inMsgType, out outMsg outMsgType)
```

The data type for *inMsgType* and the data type for *outMsgType* can be VARCHAR, VARCHAR FOR BIT DATA, CLOB, or BLOB of any length. The input data type and output data type can be different data types.

Related concepts:

- "Asynchronous messaging in DB2 Information Integrator" on page 204

Related tasks:

- “Configuring MQListener” on page 210
- “Configuring and running MQListener” on page 207

Related reference:
- “db2mqlsn - MQ Listener Command” in the *Command Reference*
- “Parameters used in MQListener configuration” on page 214

Configuring and running MQListener

Use this procedure to configure the environment for MQListener and to develop a simple application that receives a message, inserts the message in a table, and creates a simple response message.

Procedure:

To configure and run MQListener:
1. Configure MQListener to run in the DB2 Universal Database environment.
2. Configure WebSphere MQ for MQListener.
3. Configure MQListener.
4. Create the stored procedure to work with MQListener.
5. Run a simple MQListener application.

Related concepts:
- “MQListener in DB2 Information Integrator” on page 205

Related tasks:
- “Configuring MQListener to run in the DB2 Universal Database environment” on page 207
- “Configuring WebSphere MQ for MQListener” on page 209
- “Configuring MQListener” on page 210
- “Creating a stored procedure to use with MQListener” on page 211

Related reference:
- “MQListener examples” on page 212
- “Parameters used in MQListener configuration” on page 214
- “WebSphere MQ queues used in MQListener” on page 215

Configuring MQListener to run in the DB2 Universal Database environment

Configure your database environment so that your applications can use messaging with database operations.

Prerequisites:

Create a database for the MQListener configuration and a database for the stored procedures that you call when a message arrives (if valid databases are not already available). You can use the same database for the configuration and the stored procedures.

The configuration users must have the following privileges and authorizations:

- Read and write access to the DB2 UDB table SYSMQL.LISTENERS. MQListener run users do not need access to SYSMQL.LISTENERS
- Authority to run the configuration package MQLConfi

The run users must have the authority to run the MQLRun package.

Procedure:

To configure MQListener to run with DB2 Universal Database databases:

1. Issue the following command to connect. Substitute the appropriate values for your database environment:
   ```
   db2 connect to ConfigDB user DBAdmin using DBAdminPwd
   ```
2. Run the *MQLInstall.sql* script, which creates a table that stores the MQListener configuration. The script is in the following path:
   - .../sqllib/bin in a UNIX environment
   - ...\sqllib\bin in a Windows environment
   ```
   db2 -td; -f MQLInstall.sql
   ```
3. Issue the following commands to grant access to the configuration user. Substitute the appropriate values for your database environment:
   ```
   db2 grant all privileges on table SYSMQL.LISTENERS to ConfigUser
   db2 connect reset
   ```
4. Bind the MQListener packages and grant access to the packages. You must bind the MQLConfig package in the configuration database. Issue the following commands. Substitute the appropriate values for your database environment:
   ```
   db2 connect to ConfigDB user DBAdmin using DBAdminPwd
   db2 bind MQLConfig.bnd
   db2 grant execute on package MQLConfi to ConfigUser
   db2 connect reset
   ```

   The name MQLConfi satisfies the eight-character limitation on package name lengths.
5. Bind the MQLRun package in each of the run databases. Issue the following commands for each run database and each run user in that database:
   ```
   db2 connect to RunDB user DBAdmin using DBAdminPwd
   db2 bind MQLRun.bnd
   db2 grant execute on package MQLRun to RunUser
   db2 connect reset
   ```

Related concepts:

- “MQListener in DB2 Information Integrator” on page 205

Related tasks:

- “Configuring WebSphere MQ for MQListener” on page 209
- “Configuring MQListener” on page 210
- “Creating a stored procedure to use with MQListener” on page 211

Related reference:

- “Parameters used in MQListener configuration” on page 214
- “WebSphere MQ queues used in MQListener” on page 215
- “MQListener examples” on page 212

Configuring WebSphere MQ for MQListener

You can run a simple MQListener application with a simple WebSphere MQ configuration. More complex applications might need a more complex configuration. Configure at least two kinds of WebSphere MQ entities: the queue manager and some local queues. Configure these entities for use in such instances as transaction management, deadletter queue, backout requeue and backout retry threshold.

Prerequisites:

Issue the WebSphere MQ control commands while in the *mqm* group. The *mqm* group is used by the WebSphere MQ administrators and for internal MQ programs. All members of this group have access to all resources.

Procedure:

To configure WebSphere MQ for a simple MQListener application:

1. Create a queue manager.

   ```
   crtmqm TransQM
   ```
2. Start the queue manager.

   ```
   strmqm TransQM
   ```
3. Optional: Configure the queue manager to coordinate transactions with DB2 Universal Database.

   If you configure the queue manager to coordinate transactions with DB2 Universal Database, then MQListener applications can remove a message and call a stored procedure in a single transaction. Configure the queue manager for DB2 UDB coordination:

   - Provide the name of a shared library, which is called a switch load file, that WebSphere MQ can use to find the DB2 Universal Database X/Open resource manager functions, and an extended architecture open string (`xa_open`) that is specific to DB2 UDB
   - Create the MQStart routine in the switch load file by compiling a small C program that returns the DB2 UDB global variable `db2xa_switch`. See the *WebSphere MQ: System Administration Guide* for specific information on how to create the switch load file. MQStart returns a structure of pointers to the functions that implement the X/Open resource manager functions in DB2 UDB. The following example shows the required format for the extended architecture open string, with appropriate values that are substituted for MQListener configuration parameters:

     ```
     DB=RunDB, UID=RunUser, PWD=RunUserPwd, TPM=MQ, TOC=P
     ```

     If you use TPM=MQ in the extended architecture open string as in this example, you do not need to set the DB2 TP_MON_NAME instance variable.

   WebSphere MQ obtains the parameters that it needs based on the operating system environment. On Windows operating systems the parameters are in the Windows registry. You can specify the parameters by using the WebSphere MQ MQServices. On UNIX operating systems the parameters are in the queue manager configuration file. Use a valid text editor for your environment to specify the parameters to use.
4. Create your local queues by using the WebSphere MQ script facility.
   a. Create a file that contains the following commands (for this example the file is mqconfig.mqs):

```
define qlocal('DLQ')
alter qmgr deadq('DLQ')
define qlocal('Backout')
define qlocal('Admin')
define qlocal('In') boqname('Backout') bothresh(3)
define qlocal('SYSTEM.SAMPLE.REPLY')
```

b. Redirect the mqconfig.mqs file into the script interpreter by issuing the following command:

```
runmqsc TransQM < mqconfig.mqs
```

Related tasks:

- "Configuring MQListener" on page 210

Related reference:

- "Parameters used in MQListener configuration" on page 214
- "WebSphere MQ queues used in MQListener" on page 215

Configuring MQListener

Use the MQListener command, **db2mqlsn**, to configure MQListener. Issue the command **db2mqlsn** from a command line in any directory. On a Windows system, issue the command in a DB2 UDB command line processor window to insure proper message display. The add parameter with the db2mqlsn command updates a row in the DB2 table SYSMQL.LISTENERS. .

Restrictions:

- Use the same queue manager for the request queue and the reply queue.
- On Windows systems, each thread can connect to one queue manager.
- On UNIX systems, each process can connect to one queue manager. If you specify different queue managers within the same MQListener configuration on a UNIX system, you receive a run-time error from WebSphere MQ.
- MQListener does not support logical messages that are composed of multiple physical messages. MQListener processes physical messages independently.

Procedure:

To specify MQListener configuration:

- To add an MQListener configuration, issue the following command:

```
db2mqlsn add
  -configDB ConfigDB
  -config aConfiguration
  -configUser ConfigUser
  -configPwd ConfigUserPwd
  -queueManager TransQM
  -inputQueue In
  -procSchema RunUser
  -procName aProc
  -dbName RunDB
  -dbUser RunUser
  -dbPwd RunUserPwd
  -mqCoordinated
```

- To display all of the tasks in a configuration, issue the following command:

```
db2mqlsn show
  -configDB ConfigDB
  -config aConfiguration
  -configUser ConfigUser
  -configPwd ConfigUserPwd
```

- To remove the messaging tasks, issue the following command:

```
db2mqlsn remove
  -configDB ConfigDB
  -config aConfiguration
  -configUser ConfigUser
  -configPwd ConfigUserPwd
  -queueManager TransQM
  -inputQueue In
```

- To get help with the command and the valid parameters, issue the following command:

```
db2mqlsn help
```

- To get help for a particular parameter, issue the command with a specific parameter, as in the following example:

```
db2mqlsn help <command>
```

Related reference:

- "db2mqlsn - MQ Listener Command" in the *Command Reference*
- "Parameters used in MQListener configuration" on page 214

Creating a stored procedure to use with MQListener

The run database contains the stored procedure that is run when a message arrives. The run user is the user in whose name MQListener connects to the run database to run the stored procedure. Use the following parameters with the db2mqlsn add command to define the run database and the run user:

- -dbName
- -dbUser

The run user must be able to connect to the run database and run the stored procedure. The run user does not need to be the owner of the stored procedure. The run user also does not need access to the MQListener configuration.

MQListener uses the stored procedure, aProc, to store a message in a table. The stored procedure returns the string OK if the message is successfully inserted into the table.

Prerequisites:

The stored procedure requires a C compiler.

Procedure:

The following steps create DB2 Universal Database objects that you can use with MQListener applications.

1. Create a simple table as the run user (you can use the DB2 UDB command line processor):

   ```
   CREATE TABLE aTable (val VARCHAR(25) CHECK (val NOT LIKE 'fail%'))
   ```

The table contains a check constraint so that messages that start with the characters `fail` cannot be inserted into the table. The check constraint is used to demonstrate the behavior of MQListener when the stored procedure fails.

2. Create the following stored procedure:

```
CREATE PROCEDURE aProc (IN pin VARCHAR(25), OUT pout VARCHAR(2))
BEGIN
        INSERT INTO aTable VALUES(pin);
        SET pout = 'OK';
END
```

Related tasks:

- "Configuring MQListener to run in the DB2 Universal Database environment" on page 207

Related reference:

- "Parameters used in MQListener configuration" on page 214

MQListener examples

The following examples show a simple MQListener application. The application receives a message, inserts the message in a table, and generates a simple response message. To simulate a processing failure, the application includes a check constraint on the table that contains the message. The constraint prevents any string that begins with the characters `fail` from being inserted into the table. If you attempt to insert a message that violates the check constraint, the example application returns an error message and requeues the failing message to the backout queue.

To run MQListener with all of the tasks specified in a configuration, issue the following command:

```
db2mqlsn run
  -configDB ConfigDB
  -config aConfiguration
  -configUser ConfigUser
  -configPwd ConfigUserPwd
  -adminQueue Admin
  -adminQMgr TransQM
```

The following examples show how to use MQListener to send a simple message and then inspect the results of the message in the WebSphere MQ queue manager and the database. The examples include queries to determine if the input queue contains a message, or if a record is placed in the table by the stored procedure. Many tools support these operations, including the DB2 Universal Database command line processor, DB2 UDB Command Center, some WebSphere MQ command line utilities, sample programs, the MQ Explorer, and the MQ API exerciser. Consider using DB2 Universal Database and WebSphere MQ tools for more complex applications.

MQListener example 1: Running a simple application:

1. Start with a clean database table:

```
db2 delete from aTable
```

2. Send a datagram to the input queue:
   a. Place the string `a sample message` in a file named sampleMsg1.txt
   b. Use the WebSphere MQ sample program **amqsput** to put the message on the queue:

```
amqsput In TransQM < sampleMsg1.txt
```

3. Query the table to verify that the sample message is inserted:
   ```
   db2 select * from aTable
   ```
4. Display the number of messages that remain on the input queue to verify that the message has been removed:
   a. Place the following command in the file checkIn.mqs:
      ```
      display queue('In') curdepth
      ```
   b. Redirect the command into the script interpreter:
      ```
      runmqsc TransQM < checkIn.mqs
      ```

MQListener example 2: Sending requests to the input queue and inspecting the reply:

The following example statements send a request to the input queue and inspect the reply:

1. Start with a clean database table:
   ```
   db2 delete from aTable
   ```
2. Send a request to the input queue:
   a. Place the string `another sample message` in a file named sampleMsg2.txt
   b. Use the WebSphere MQ sample program **amqsreq** to send the request to the input queue:
      ```
      amqsreq In TransQM < sampleMsg2.txt
      ```
      The **amqsreq** program sets the reply-to queue in the request to SYSTEM.SAMPLE.REPLY
3. Query the table to verify that the sample message is inserted:
   ```
   db2 select * from aTable
   ```
4. Display the number of messages that remain on the input queue to verify that the message is removed.
   ```
   display queue('In') curdepth
   ```
5. Look at the SYSTEM.SAMPLE.REPLY queue for the reply by using the WebSphere MQ sample program amqsget. Verify that the OK string is generated by the stored procedure:
   ```
   amqsget SYSTEM.SAMPLE.REPLY TransQM
   ```

MQListener example 3: Testing an unsuccessful insert operation:

If you send a message that starts with the string `fail`, the constraint in the table definition is violated, and the stored procedure fails.

1. Start with a clean database table:
   ```
   db2 delete from aTable
   ```
2. Send a request to the input queue:
   a. Place the string `failing sample message` in a file named sampleMsg3.txt
   b. Use the WebSphere MQ sample program **amqsreq** to send the request to the input queue:
      ```
      amqsreq In TransQM < sampleMsg3.txt
      ```
      The **amqsreq** program sets the reply-to queue in the request to SYSTEM.SAMPLE.REPLY
3. Query the table to verify that the sample message is not inserted:

```
db2 select * from aTable
```

4. Display the number of messages that remain on the input queue to verify that the message is removed:
   ```
   display queue('In') curdepth
   ```
5. Read from the SYSTEM.SAMPLE.REPLY queue and find an exception report rather than an OK reply:
   ```
   amqsget SYSTEM.SAMPLE.REPLY  TransQM
   ```
6. Read from the Backout queue and find the original message:
   ```
   amqsget Backout TransQM
   ```

Related concepts:
- "MQListener in DB2 Information Integrator" on page 205

Related tasks:
- "Configuring WebSphere MQ for MQListener" on page 209
- "Configuring MQListener" on page 210
- "Creating a stored procedure to use with MQListener" on page 211
- "Configuring MQListener to run in the DB2 Universal Database environment" on page 207
- "Configuring and running MQListener" on page 207

Related reference:
- "Parameters used in MQListener configuration" on page 214
- "WebSphere MQ queues used in MQListener" on page 215

Parameters used in MQListener configuration

ConfigDB
The configuration database, which can be any valid DB2 Universal Database, contains an MQListener configuration table. The configuration table contains information about the queues to which MQListener should listen and the stored procedures MQListener should call.

ConfigUser
The user ID in whose name you access the configuration database. The configuration user does not need to be a database administrator. You can specify the configuration user and password in the MQListener command. If you do not specify a configuration user and password, and your database installation supports implicit connections, by default the configuration user is the user under whose account the MQListener is running.

ConfigUserPwd
The password that is used with the configuration user ID.

RunDB
The run database is the database that contains the stored procedures that are run when a message arrives. The stored procedures can be in different databases from the configuration database.

RunUser
The user in whose name you access the run database to run the stored procedure. The run user does not need any privilege except the ability to connect to the run database and run the stored procedure.

RunUserPwd
The password that is associated with the run user.

Related concepts:

- "MQListener in DB2 Information Integrator" on page 205

Related reference:

- "db2mqlsn - MQ Listener Command" in the *Command Reference*

WebSphere MQ queues used in MQListener

In a simple MQListener application, you typically use the following WebSphere MQ queues:

Deadletter queue
The deadletter queue (DLQ) in WebSphere MQ holds messages that cannot be processed. MQListener uses this queue to hold replies that cannot be delivered, for example, because the queue to which the replies should be sent is full. A deadletter queue is useful in any MQ installation especially for recovering messages that are not sent.

Backout queue
For MQListener tasks in which WebSphere MQ is the transaction coordinator, the Backout queue serves a similar purpose to the deadletter queue. MQListener places the original request in the Backout queue after the request is rolled back a specified number of times (called the backout threshold).

Administration queue
The administration queue is used for routing control messages such as *shutdown* and *restart* to MQListener. If you do not supply an administration queue, then the only way to shut down MQListener is to issue a **kill** command.

Application input and output queues
The application uses input queues and output queues. The application receives messages from the input queue. The application sends replies and exceptions to the output queue, which is SYSTEM.SAMPLE.REPLY to conform to the usage in WebSphere MQ sample program, amqsreq.

Related tasks:

- "Configuring WebSphere MQ for MQListener" on page 209
- "Configuring and running MQListener" on page 207

Appendix A. Script examples for Cottonwood Distributors, Inc. and YBar, Inc. scenarios

```
----------------------------------------------------
-- Catalog remote DB2 UDB server machine and database
----------------------------------------------------
uncatalog node DB2_TPCH;
catalog tcpip node DB2_TPCH remote x.xx.xx.xx server 50000;

uncatalog database tpcd;
catalog database tpcd at node db2_tpch;

----------------------------------------------------
-- DB2 UDB wrapper, nicknames, MQTs, and indexes
----------------------------------------------------
drop wrapper drda;
create wrapper drda;

create server db2_tpch type db2/udb version  8.1
  wrapper drda authorization "demo" password "xxxxx"
  options (dbname 'TPCD');
create user mapping
  for user SERVER db2_tpch
  OPTIONS ( REMOTE_AUTHID 'demo', REMOTE_PASSWORD 'xxxxx' );

create nickname db2_part for db2_tpch.tpcd.part;
create nickname db2_supplier for db2_tpch.tpcd.supplier;
create nickname db2_partsupp for db2_tpch.tpcd.partsupp;
create nickname db2_nation for db2_tpch.tpcd.nation;
create nickname db2_region for db2_tpch.tpcd.region;
create nickname db2_customer for db2_tpch.tpcd.customer;
create nickname db2_orders for db2_tpch.tpcd.orders;
```

Figure 72. federated.sql (Part 1 of 9)

© Copyright IBM Corp. 2003, 2004

```
----------------------------------------------------
-- Oracle wrapper, nicknames, MQTs and indexes
----------------------------------------------------
drop wrapper net8;
create wrapper net8;

create server oraserver type oracle version 8
  wrapper net8
  options (node 'iidemo2');
create user mapping
  for user SERVER oraserver
  OPTIONS ( REMOTE_AUTHID 'demo', REMOTE_PASSWORD 'xxxxx' );

create nickname ora_part for oraserver.demo.part;
create nickname ora_supplier for oraserver.demo.supplier;
create nickname ora_partsupp for oraserver.demo.partsupp;
create nickname ora_customer for oraserver.demo.customer;
create nickname ora_orders for oraserver.demo.orders;
create nickname ora_lineitem for oraserver.demo.lineitem;
```

Figure 72. federated.sql (Part 2 of 9)

```
----------------------------------------------------
-- Informix wrapper, nicknames, MQTs, and indexes
----------------------------------------------------
drop wrapper informix;
create wrapper informix;

create server infserver type informix version 9
  wrapper informix
  options (node 'ol_informix',dbname 'tpcd2');
create user mapping
  for user SERVER infserver
  OPTIONS ( REMOTE_AUTHID 'informix', REMOTE_PASSWORD 'informix' );

create nickname inf_part for infserver."informix"."part";
create nickname inf_supplier for infserver."informix"."supplier";
create nickname inf_partsupp for infserver."informix"."partsupp";
create nickname inf_customer for infserver."informix"."customer";
create nickname inf_orders   for infserver."informix"."orders";
create nickname inf_lineitem for infserver."informix"."lineitem";
```

Figure 72. federated.sql (Part 3 of 9)

```
----------------------------------------------------
-- Union views over federated nicknames
----------------------------------------------------
DROP VIEW part_fed;
CREATE  VIEW part_fed (
  p_partkey, p_mfgr, p_type, p_size, p_retailprice) AS
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM db2_part
UNION ALL
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM inf_part
UNION ALL
  SELECT p_partkey, p_mfgr, p_type, p_size,  p_retailprice
  FROM ora_part;

DROP VIEW partsupp_fed;
CREATE VIEW partsupp_fed (
   ps_partkey, ps_suppkey, ps_supplycost) AS
   SELECT ps_partkey, ps_suppkey, ps_supplycost
   FROM  db2_partsupp
UNION ALL
   SELECT ps_partkey, ps_suppkey, ps_supplycost
   FROM  inf_partsupp
UNION ALL
   SELECT ps_partkey, ps_suppkey, ps_supplycost
   FROM  ora_partsupp;

DROP VIEW supplier_fed;
CREATE VIEW supplier_fed (
   s_suppkey, s_name, s_address ) AS
   SELECT s_suppkey, s_name, s_address
   FROM  db2_supplier
UNION ALL
   SELECT s_suppkey, s_name, s_address
   FROM inf_supplier
UNION ALL
   SELECT s_suppkey, s_name, s_address
   FROM ora_supplier;
```

Figure 72. federated.sql (Part 4 of 9)

```
----------------------------------------------------
-- Create Wrapper, Server and nickname for XML
----------------------------------------------------

drop wrapper xml_files;

create wrapper XML_files library 'db2lsxml.dll';
create server LOCAL_XML_FILES wrapper XML_FILES;
create nickname Employees_From_XML (
  doc Varchar(100) OPTIONS(DOCUMENT 'FILE'),
  Employee_Number Varchar(5) OPTIONS(XPATH './@SerialNum'),
  First_Name Varchar(50) OPTIONS(XPATH './/Firstname'),
  Middle_Initial Varchar(50) OPTIONS(XPATH './/Initial'),
  Last_Name Varchar(50) OPTIONS(XPATH './/Lastname'),
  Department_Number Varchar(50) OPTIONS(XPATH './/Department'),
  Phone_Number Varchar(50) OPTIONS(XPATH './/PhoneNumber'),
  Job Varchar(50) OPTIONS(XPATH './/Job'),
  Education_Level Varchar(50) OPTIONS(XPATH './/EDLevel'),
  Gender Varchar(50) OPTIONS(XPATH './/Sex'),
  Hire_Date Varchar(50) OPTIONS(XPATH './/HireDate'),
  Birth_Date Varchar(50) OPTIONS(XPATH './/BirthDate'),
  Annual_Salary Varchar(50) OPTIONS(XPATH './/Salary'),
  Annual_Bonus Varchar(50) OPTIONS(XPATH './/Bonus'),
  Commission Varchar(50) OPTIONS(XPATH './/Comm'),
  cid Varchar(16) OPTIONS(PRIMARY_KEY 'YES'))
 FOR SERVER LOCAL_XML_FILES
 OPTIONS (XPATH '//Employee');
```

Figure 72. federated.sql (Part 5 of 9)

```
create nickname Employees_From_XML_Included (
  Employee_Number Varchar(5) OPTIONS(XPATH './@SerialNum'),
  First_Name Varchar(50) OPTIONS(XPATH './/Firstname'),
  Middle_Initial Varchar(50) OPTIONS(XPATH './/Initial'),
  Last_Name Varchar(50) OPTIONS(XPATH './/Lastname'),
  Department_Number Varchar(50) OPTIONS(XPATH './/Department'),
  Phone_Number Varchar(50) OPTIONS(XPATH './/PhoneNumber'),
  Job Varchar(50) OPTIONS(XPATH './/Job'),
  Education_Level Varchar(50) OPTIONS(XPATH './/EDLevel'),
  Gender Varchar(50) OPTIONS(XPATH './/Sex'),
  Hire_Date Varchar(50) OPTIONS(XPATH './/HireDate'),
  Birth_Date Varchar(50) OPTIONS(XPATH './/BirthDate'),
  Annual_Salary Varchar(50) OPTIONS(XPATH './/Salary'),
  Annual_Bonus Varchar(50) OPTIONS(XPATH './/Bonus'),
  Commission Varchar(50) OPTIONS(XPATH './/Comm'),
  cid Varchar(16) OPTIONS(PRIMARY_KEY 'YES'))
 FOR SERVER LOCAL_XML_FILES
 OPTIONS (FILE_PATH 'c:\cdi_data_files\CDI_Employees.xml', XPATH '//Employee');
```

Figure 72. federated.sql (Part 6 of 9)

```
----------------------------------------------------
-- Create read functions and views over MQ queues for CR
----------------------------------------------------

CREATE FUNCTION NEW_SUPPLIERS_READ()
    RETURNS TABLE  ( SUPPLIER_NAME VARCHAR(80),
                          SUPPLIER_PHONE VARCHAR(12),
                          PART_KEY DOUBLE,
                          PART_PRICE DOUBLE,
                          MAN_DAYS DOUBLE,
                          MAX_QUANTITY DOUBLE,
     CORRELID VARCHAR(80))
    LANGUAGE SQL
    NOT DETERMINISTIC
    EXTERNAL ACTION
    READS SQL DATA
    RETURN
 SELECT
         VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),80),
         VARCHAR(DB2MQ.GETCOL(T.MSG,',',2),12),
         DOUBLE(DB2MQ.GETCOL(T.MSG,',',3)),
         DEC(DB2MQ.GETCOL(T.MSG,',',4),8,4),
         BIGINT(DB2MQ.GETCOL(T.MSG,',',5)),
         BIGINT(DB2MQ.GETCOL(T.MSG,',',6)),
  CORRELID
 FROM TABLE (DB2MQ.MQREADALL('DB2.DEFAULT.SERVICE',
      'DB2.DEFAULT.POLICY')) AS T;

create view READ_NEW_SUPPLIERS_FROM_QUEUE
  as select * from table(new_suppliers_read()) t
  where CORRELID = 'CDI_NEW_SUPPLIER';
```

Figure 72. federated.sql (Part 7 of 9)

```
----------------------------------------------------
-- Create destroy functions and views over MQ queues for CR
----------------------------------------------------
CREATE FUNCTION NEW_SUPPLIERS_REC()
    RETURNS TABLE  ( SUPPLIER_NAME VARCHAR(80),
                          SUPPLIER_PHONE VARCHAR(12),
                          PART_KEY DOUBLE,
                          PART_PRICE DOUBLE,
                          MAN_DAYS DOUBLE,
                          MAX_QUANTITY DOUBLE,
     CORRELID VARCHAR(80))
    LANGUAGE SQL
    NOT DETERMINISTIC
    EXTERNAL ACTION
    READS SQL DATA
    RETURN
 SELECT
        VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),80),
        VARCHAR(DB2MQ.GETCOL(T.MSG,',',2),12),
        DOUBLE(DB2MQ.GETCOL(T.MSG,',',3)),
        DEC(DB2MQ.GETCOL(T.MSG,',',4),8,4),
        BIGINT(DB2MQ.GETCOL(T.MSG,',',5)),
        BIGINT(DB2MQ.GETCOL(T.MSG,',',6)),
  CORRELID
 FROM TABLE (DB2MQ.MQRECEIVEALL('DB2.DEFAULT.SERVICE',
  'DB2.DEFAULT.POLICY', 'CDI_NEW_SUPPLIER', 1)) AS T;

create view RECEIVE_NEW_SUPPLIERS_FROM_QUEUE
  as select * from table(new_suppliers_rec()) t;
```

Figure 72. federated.sql (Part 8 of 9)

```
----------------------------------------------------
-- Create temporary tables used in processing
----------------------------------------------------

-- For running custom java programs
drop table aux_table;
create table aux_table (part_key integer,
  supplier_key int, supply_cost double);

-- For running XML composition
drop table UCustomers;
create table Ucustomers (x_doc DB2XML.XMLCLOB not logged);

-- For storing the web request
drop table request_bid;
drop table request_status;
create table request_bid (reqkey integer not null,
  partkey integer not null, bid double not null);
create table request_status (reqkey integer not null,
  partkey integer not null, suppkey integer not null,
  newquote double not null,currentquote double not null,
  status varchar(15) not null);

-- For shredding XML documents to DB2
IMPORT FROM c:\CDI_Data_Files\Setup\CDI_Employees.ixf of
  IXF CREATE INTO EMPLOYEES_DB2;
CREATE TABLE EMPLOYEES_FROM_XML_FILE_SHRED LIKE EMPLOYEES_DB2;
```

Figure 72. federated.sql (Part 9 of 9)

```
// Import all necessary classes
import java.lang.*;
import java.sql.*;
import java.util.*;

// USAGE: db, user, password, timeout

/**
 * Class Listener:
 * Class which will check a message queue for messages and
 *    call the "Director" stored procedure
 * to process the message.
 */
public class CDIListener{

 /**
  * Method checkQueue
  * Method to check the queue for messages.
    * If present calls the "Director" stored procedure
  *  otherwise it will wait for a user specified number of seconds
  */
```

Figure 73. CDIListener.java (Part 1 of 12)

```
public void checkQueue(String db, String user,
          String pass, int timeout) {

  /* Local variables */
  String urlDB2 = null;
  Connection connDB2 = null;
  PreparedStatement stmtDB2 = null;
  PreparedStatement stmtDB2_2 = null;
  ResultSet rsDB2 = null;
  ResultSet rsDB2_2 = null;
  ResultSet rsTotal = null;
  String query = null;

  int mqMsgType  = 0;
  int mqMsgKey   = 0;
  int mqMsgPart  = 0;
  int mqMsgQuant = 0;
  double mqMsgPrice = 0;

  String comment = null;
  int intValue = 0;
  int numRecords = 0;

  System.out.println("\nStarting listener execution");
  System.out.println("\nConnecting to DB2");
```

Figure 73. CDIListener.java (Part 2 of 12)

```
// Print out parms
System.out.println("\nUsing startup parms of: \n Database: " +
     db + "\n User: " + user + "\n Password: **********\n
     Sleep interval: " + timeout + " milliseconds");

try {
 //Load drivers' classes
 System.out.println("\nLoading DB2 driver");
 Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

 //URL for databases to be connected
 System.out.println("\nSetting URL to database");
 urlDB2 = "jdbc:db2:" + db;

 //Get database connections
 System.out.println("\nGetting DB2 connection");
 connDB2 = DriverManager.getConnection(urlDB2, user, pass);
} catch (Exception e) {
 e.printStackTrace();
}
```

Figure 73. CDIListener.java (Part 3 of 12)

```
/*Issue a receive on the MQ queue*/
System.out.print("\nStart MQ queue processing...");
System.out.flush();

double test = 0;

/* Loop forever to read the queue */
while (test == 0) {
 // System.out.println("\nChecking queue...");
 try {
  stmtDB2 = connDB2.prepareStatement
           ("SELECT VARCHAR(DB2MQ.GETCOL(T.MSG,',',1),1),
             INT(DB2MQ.GETCOL(T.MSG,',',2)),
             INT(DB2MQ.GETCOL(T.MSG,',',3)),
             INT(DB2MQ.GETCOL(T.MSG,',',4)),
             DOUBLE(DB2MQ.GETCOL(T.MSG,',',5))
             FROM TABLE (DB2MQ.MQRECEIVEALL('DB2.DEFAULT.SERVICE',
             'DB2.DEFAULT.POLICY','CDI_IN_MSG',1)) AS T");
      rsDB2 = stmtDB2.executeQuery();
 }
 catch (SQLException e) {
  if (e.getErrorCode() == 100) {
   test = 0;
  }
  else {
   System.out.println(e);
   test = 1;
  }
 }
 catch (Exception e) {
  e.printStackTrace();
  test = 1;
 }
```

Figure 73. CDIListener.java (Part 4 of 12)

```
try {
  // Get a message off the queue
  if (rsDB2 != null)
  while(rsDB2. next()) {
  // Get the type from the MQ message and
           //   call appropriate function
  mqMsgType  = rsDB2.getInt(1);

  // if END message, exit
  if (mqMsgType == 0) {
    System.out.println("\nFound END message on queue");
    test = 1;
  }
  else {
   // Found a customer buy request message
   if (mqMsgType == 1) {
    // get other values from message
    mqMsgKey   = rsDB2.getInt(2);
    mqMsgPart  = rsDB2.getInt(3);
    System.out.println("\nFound message type="+ mqMsgType + ";
                        customer="+ mqMsgKey + "; part=" + mqMsgPart);

    // Update the REQUEST_BID table
    System.out.print("Executing update to
                    local REQUEST_BID table...");
    // put customer order into REQUEST_BID table
                 //    using max part price + 45% markup
    stmtDB2_2 = connDB2.prepareStatement
                     ("INSERT into request_bid
                       values (" + mqMsgKey + "," + mqMsgPart + "," + "
                       (SELECT MIN(ps_supplycost)*1.45
                         FROM partsupp_fed
                         WHERE ps_partkey = " + mqMsgPart + "))");
         stmtDB2_2.executeUpdate();
    System.out.println("\nComplete");
    stmtDB2_2.close();
   }
```

Figure 73. CDIListener.java (Part 5 of 12)

```
   else {
    // Else if it's a supplier price update
    if (mqMsgType == 2) {
     // Get the other values off the queue
     mqMsgKey   = rsDB2.getInt(2);
     mqMsgPart  = rsDB2.getInt(3);
     mqMsgQuant = rsDB2.getInt(4);
     mqMsgPrice = rsDB2.getDouble(5);
     System.out.println("\nFound message
                         type="+ mqMsgType + ";
                         supplier="+ mqMsgKey + ";
                         part=" + mqMsgPart + ";
                         price=" + mqMsgPrice);
```

Figure 73. CDIListener.java (Part 6 of 12)

```
// Check if this supplier has supplier this part before
System.out.print("\nChecking if supplier
                        already supplies part...");
stmtDB2 = connDB2.prepareStatement
                       ("SELECT COUNT(*)
                         FROM partsupp_fed
                         WHERE ps_partkey = " + mqMsgPart + "
                         and ps_suppkey = " + mqMsgKey);
  rsDB2_2 = stmtDB2.executeQuery();
  rsDB2_2.next();
  intValue = rsDB2_2.getInt(1);
System.out.print(intValue + "...");
```

Figure 73. CDIListener.java (Part 7 of 12)

```
  // If supplier has supplier before
  if (intValue > 0) {

   // Get current price by supplier
 System.out.print("Yes!\nGetting current
                         minimum price...");
   stmtDB2 = connDB2.prepareStatement
                    ("SELECT MIN(ps_supplycost)
                      FROM partsupp_fed
                      WHERE ps_partkey = " + mqMsgPart + "
                      and ps_suppkey = " + mqMsgKey);
   rsDB2_2 = stmtDB2.executeQuery();
   rsDB2_2.next();
   intValue = rsDB2_2.getInt(1);

   System.out.println("Current price = " + intValue + "\n");
```

Figure 73. CDIListener.java (Part 8 of 12)

```
   //If new price less than or equal to
                //existing price, update database
                //and mark as accepted
   if (mqMsgPrice <= intValue) {
  comment = "'ACCEPT'";
   }
   // Else update the database but mark as review
   else {
  comment = "'REVIEW'";
   }
 System.out.print("\nExecuting " + comment + "
                       update to federated db2 table db2_partsupp...");
   stmtDB2 = connDB2.prepareStatement
                   ("UPDATE db2_partsupp
                     set ps_availqty = " + mqMsgQuant + ",
                     ps_supplycost = " + mqMsgPrice + "
                     where ps_partkey = " +  mqMsgPart + "
                     and ps_suppkey = " + mqMsgKey);
   stmtDB2.executeUpdate();
 System.out.println("Complete\n");
  }
```

Figure 73. CDIListener.java (Part 9 of 12)

```
// Else this is a new supplier for this part
else {
 System.out.print("No!\nExecuting 'NEW' insert to
                    federated db2 table db2_partsupp...");
 comment = "'NEW'";

 // Add new record to database
 stmtDB2 = connDB2.prepareStatement
                   ("INSERT into db2_partsupp
                       values (" + mqMsgPart + "," +
                       mqMsgKey + ","+ mqMsgQuant + "," +
                       mqMsgPrice + ",'New supplier added at: " +
                       new java.util.Date()+ " ')");
 stmtDB2.executeUpdate();
 System.out.println("Complete\n");
}
// Update the local table
System.out.print("\nUpdating local REQUEST_STATUS table...");
  stmtDB2_2 = connDB2.prepareStatement
                 ("INSERT into request_status
                     values (" + mqMsgKey + "," +
                     mqMsgPart + "," +mqMsgKey + ","+
                     mqMsgPrice + "," +
                     intValue + "," + comment + ")");
  stmtDB2_2.executeUpdate();
System.out.println("Complete\n");
```

Figure 73. CDIListener.java (Part 10 of 12)

```
        if (stmtDB2_2 != null)
         stmtDB2_2.close();
        if (rsDB2_2 != null)
         rsDB2_2.close();
       }
       else {
        System.out.println("\nError - unknown message type");
       }
      }
     }
    System.out.flush();
    } // End secondary while
    System.out.flush();
   } // End try

   catch (Exception e) {
    e.printStackTrace();
   }

   // Sleep for the necessary time
   try {
    if (rsDB2 != null)
     rsDB2.close();
    if (stmtDB2 != null)
     stmtDB2.close();
    Thread.sleep(timeout);
    System.out.print(".");
    System.out.flush();
   }
   catch (Exception e) {
    e.printStackTrace();
   }
  } // end Main while
```

Figure 73. CDIListener.java (Part 11 of 12)

```
  System.out.println("Listener stopped\n");
  System.out.println("\nListener stopped\n");

  /* Close all database connections */
  try {
   rsDB2.close();
   stmtDB2.close();
   connDB2.close();
  } catch (Exception e) {
   e.printStackTrace();
  }

  return;

 }

 /* Main program to invoke listener */
 public static void main(String argv[]) {

  String DBName = argv[0];
  String DBUser = argv[1];
  String DBPass = argv[2];
  String TimeCk = argv[3];

  CDIListener dp = new CDIListener();
  int timeout = Integer.parseInt(TimeCk)*1000;
  dp.checkQueue(DBName, DBUser, DBPass, timeout);
 }
}
```

Figure 73. CDIListener.java (Part 12 of 12)

```
/*
 * @(#)MessageFormatter.java
 *
 * CopyrightVersion 1.0
 *
 */

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.ServletException;
import proxy.soap.*;

public class MessageFormatter extends javax.servlet.http.HttpServlet {

 final static String htmlHeader1 = "<HTML><TITLE>MessageFormatter</TITLE>";
final static String message1 =
 "<p align=\"center\">
   <font color=\"navy\"
   face=\"verdana\" size=\"+2\">
<b>Thank You</b></font></p>
<p><font color=\"black\"
face=\"veranda\" size=\"+1\">";
```

Figure 74. MessageFormatter.java (Part 1 of 7)

```
final static String message2 = "";

 final static String htmlHeader2 = "</HTML>";
 static String quote = "";

 static Connection con = null;
 final static String url = "jdbc:db2:demo";

  // method to generate a random REQUEST number
 public int randomint() {
  double first = java.lang.Math.random();
  String help = java.lang.Double.toString(first);
  help = help.substring(3,7);
  int number = Integer.parseInt(help);
  number = number / 5;
  return number;
 }
```

Figure 74. MessageFormatter.java (Part 2 of 7)

```
// Overwriting doGet method to handle Http GET request

 public void doGet(HttpServletRequest req,
        HttpServletResponse res)
  throws javax.servlet.ServletException, java.io.IOException {
  try {


   PrintWriter pr = res.getWriter();
   pr.print(htmlHeader1);
   pr.print(message1);
   pr.print(message2);
   String method = "";
   String part = "";
   String price = "";
   String quantity = "";
   String key = "";
   String cust_name = "";
   String col_name1 = null;
   String col_name2 = null;
   String tab_name  = null;

   Connection con = null;
   String url = null;
   Statement stmt = null;
   ResultSet rs = null;

   WSProxy WSid = new WSProxy();
   boolean fCustomer = true;
```

Figure 74. MessageFormatter.java (Part 3 of 7)

```
if (req.getParameter("method")!=null) {
 // Get the common parms to the servlet from the REQ object
 key = req.getParameter("name");
 part = req.getParameter("part");
 method = req.getParameter("method");

 // If customer order
 if (method.equals("orderNewParts")) {
  fCustomer = true;
  // Get the quantity the customer is ordering
  quantity = req.getParameter("quantity");

  // set the table and column names for SELECT
  col_name1 = "c_name";
  col_name2 = "c_custkey";
  tab_name  = "db2_customer";

 }
 else {
  // Else if supplier update
  if (method.equals("setSupplierQuotes")) {
   fCustomer = false;
   // Get the price the supplier is updating
   price = req.getParameter("price");

   // set the table and column names for SELECT
   col_name1 = "s_name";
   col_name2 = "s_suppkey";
   tab_name  = "supplier_fed";
  }
  else {
   pr.println
              ("method = "+req.getParameter("method")+
                   " Not supported! <br>");
   return;
  }
 }
```

Figure 74. MessageFormatter.java (Part 4 of 7)

```
    // Get the real customer name from federated data source
    try {
     Class.forName
               ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
     url = "jdbc:db2:demo";
     con = DriverManager.getConnection(url,"demo","xxxxx");
     stmt = con.createStatement();
     rs = stmt.executeQuery
               ("SELECT " + col_name1 + " from " +
                 tab_name + " where " + col_name2 + " = " + key);
     while (rs.next()) {
      cust_name = rs.getString(1);
     }

     rs.close();
     stmt.close();
     con.close();
    }
    catch (Exception e) {
     pr.println(e);
     System.out.println(e);
     return;
    }


   }
   else {
    pr.println
            ("*** Severe error occurred-no method passed ***>");
    return;
   }
```

Figure 74. MessageFormatter.java (Part 5 of 7)

```
   // Write request status back to user
   try{
    // If this is a supplier update request
    if (!fCustomer) {
      String message1id=  "2,"
              + key + "," + part + ","
              + quantity + "," + price ;
           java.lang.String message1idTemp  = message1id;
     System.out.println
                ("message type 2 being written: "
                   + message1idTemp);
           org.tempuri.worftestweb
               .demo.newdadx.dadx.xsd
               .Stmt1ResultElement
               mtemp = WSid.stmt1(message1idTemp);
           if (mtemp == null)
      pr.println("*** Web Service error occurred ***>");
     else {
      pr.println
("<table border=\"1\"
bordercolor=\"navy\"
width=\"100%\">
<tr align+\"left\">
<td>Supplier</td>
<td>" + cust_name + "</td>
</tr>
<tr align+\"left\"><td>Part</td><td>" + part + "</td></tr>
<tr align+\"left\"><td>Price</td><td>" + price + "</td></tr>
</table>");
pr.println("<br><font color=\"red\" face=\"verdana\"
size=\"-1\">
Price update submitted for processing");
     }
    }
```

Figure 74. MessageFormatter.java (Part 6 of 7)

```
     // else must be a customer buy request
     else {
      String message1id=  "1," + key +"," + part + ",0, 0";
            String message1idTemp  = message1id;
      System.out.println("message type 1 being written: " + message1idTemp);
            org.tempuri.worftestweb.demo
                     .newdadx.dadx.xsd
                     .Stmt1ResultElement
                     mtemp = WSid.stmt1(message1idTemp);
            if (mtemp == null)
       pr.println("*** Web Service error occurred ***>");
      else {
       pr.println
                 ("<table border=\"1" bordercolor=\"navy\"
wide=\"100%\"><tr align+\"left\"><td>Customer</td>
<td>" + customer_name + "</td></tr>
<tr align+\"left\"><td>Part</td>
<td>" + part + "</td></tr>
<tr align+\"left\"><td>Price</td>
<td>" + quantity + "</td></tr></table>");
       pr.println("<br><font color=\"red\"
face=\"verdana\"
size=\"-1\">
Order submitted for processing");
      }
     }
    }
    catch (Exception e) {
     System.out.println(e);
     pr.println(e);
     return;
    }

   }
   catch (Exception e) {
   }
  }
}
```

Figure 74. MessageFormatter.java (Part 7 of 7)

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
   <dtdid>resume.dtd</dtdid>
   <validation>NO</validation>
   <Xcolumn>
     <table name="resume_skills_sidetable">
       <column name="skill"
         type="varchar(20)"
         path="/resume/skill"
         multi_occurrence="YES">
       </column>
     </table>
   </Xcolumn>
</DAD>
```

Figure 75. resume.dad

```
<?xml version="1.0"?>
<!DOCTYPE Employees SYSTEM "employees.dtd">
<Employees>
  <Employee SerialNum="12">
    <Firstname>Laurie</Firstname>
    <Lastname>Douglas</Lastname>
    <Job_Description>
       DB2 engine Development
    </Job_Description>
  </Employee>
  <Employee SerialNum="13">
    <Firstname>John</Firstname>
    <Lastname>Smith</Lastname>
    <Job_Description>
       Information Integration Technology Solutions
   </Job_Description>
  </Employee>
  <Employee SerialNum="14">
    <Firstname>George</Firstname>
    <Lastname>Jackson</Lastname>
    <Job_Description>
        Customer contact
    </Job_Description>
  </Employee>
</Employees>
```

Figure 76. All_Employees.xml

Related concepts:

- "The Cottonwood Distributors, Inc.—a warehouse example" on page 11
- "Discovering the data—the employee skills scenario" on page 12

Appendix B. 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.

Installing the DADX environment checker

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.

The DADX environment checker is included in the WORF installation, 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 where WORF is installed
- *xerces.jar* . For UNIX and Windows, these files are included in the binary distribution for Xerces-J 2.0.2 downloadable at http://xml.apache.org/. For OS/390 and z/OS, these files are included in the IBM XML Toolkit Version 1 Release 4 with PTF UW95866
- *soap.jar*, included in the binary distribution for SOAP 2.3 downloadable at http://xml.apache.org, or included in the WebSphere Application Server installation.
- *j2ee.jar*, version 1.3 or later. You can download this file from java.sun.com
- *qname.jar* . You can download this file from java.sun.com
- *wsdl4j.jar*. You can download this file from http://oss.software.ibm.com/developerworks/projects/wsdl4j.
- *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 WebSphere Application Server installation, or in the distribution for Jakarta Tomcat Version 3.2.x through 4.0.3 or later downloadable at http://www.apache.org/
- For UNIX and Windows: *db2java.zip*, included in the /java directory located where you installed DB2 Universal Database. For OS/390 and z/OS: *db2j2classes.zip*, included in the classes/ subdirectory where you installed DB2 Universal Database in HFS. You can also use *jcc.jar*. The `dbDriver` parameter in the *group.properties* files determines the driver package that you use.

For example, if you are running in the Windows environment, you must set your CLASSPATH to find the following files:

```
CheckersCommon.jar;
DADXEnvChecker.jar;
worf.jar;
```

© Copyright IBM Corp. 2003, 2004

```
xerces.jar;
j2ee.jar
qname.jar
wsdl4j.jar
soap.jar;
db2java.zip;
```

Related concepts:

- "Definition of a DADX file" on page 29

Related reference:

- "Error checking by the DADX environment checker" on page 240
- "Running the DADX environment checker" on page 238

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, and then send the output to *myOutputFile.txt* in the current 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. The DADX environment checker generates this file. The checker uses the file name *DADXEnvChecker_sample.txt* that was specified in the -out parameter.

Related concepts:
- "Definition of a DADX file" on page 29

Related reference:
- "Error checking by the DADX environment checker" on page 240
- "Installing the DADX environment checker" on page 237

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.

Related reference:

- "Checking errors in the DAD files" on page 243
- "Checking errors in the DADX files" on page 244
- "Error checking by the DADX environment checker" on page 240
- "Installing the DADX environment checker" on page 237
- "Running the DADX environment checker" on page 238
- "Checking errors in the NST files" on page 242
- "Checking errors in the web.xml file" on page 241

Error checking by the DADX environment checker

When you invoke the DADX environment checker with the `-srv` parameter, the first check that is made is 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

When you invoke the DADX environment checker without the `-srv` parameter, the first check that is made is 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.

Database error message

For some checks on NST and DADX files, the DADX environment checker performs the following actions:

1. Attempts to establish a connection to the database by using data contained in the file `group.properties`

2. Queries the database with which the group is associated
3. Checks the files of a group for errors

If the connection to the database fails, the DADX environment checker issues an error message. The following example shows a typical error message:

```
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
```

Related tasks:

- "Customizing the group.properties file" on page 64
- "Defining the web.xml and group.properties files" on page 59

Related reference:

- "Checking errors in the DAD files" on page 243
- "Checking errors in the DADX files" on page 244
- "Checking errors in the NST files" on page 242
- "Checking errors in the web.xml file" on page 241

Checking errors in the web.xml file

The DADX environment checker checks the **WEB-INF\web.xml** file under the root directory of the Web Service module, which is services in this 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 a value of 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.
```

Otherwise, each <servlet> tag must have a corresponding <servlet-mapping> tag with the same servlet name. 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> tags 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).
```

Related tasks:

- "Defining the web.xml and group.properties files" on page 59

Related reference:

- "Error checking by the DADX environment checker" on page 240

Checking errors in the 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 for correct schema in **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.
```

Related reference:

- "Checking errors in the DAD files" on page 243
- "Checking errors in the DADX files" on page 244
- "Error checking by the DADX environment checker" on page 240
- "Installing the DADX environment checker" on page 237
- "Running the DADX environment checker" on page 238
- "Checking errors in the web.xml file" on page 241

Checking errors in the 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 Universal 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 according to 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 dadchecker tool at the WebSphere Application Development Web site.

Related reference:

- "Error checking by the DADX environment checker" on page 240
- "Web services samples – PartOrders.dadx" on page 127

Checking errors in the 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 >
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 according to 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\HelloSample.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/".
```

Related concepts:

- "Definition of a DADX file" on page 29

Related reference:

- "Error checking by the DADX environment checker" on page 240

Appendix C. XML schema for the DADX file

The following Extensible Markup Language (XML) schema, *dadx.xsd*, describes the DADX. All of the WORF schema files are in the dxxworf.zip file, which is part of the samples directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" xml:lang="en">
 <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>
 <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>
```

Figure 77. DADX schema (Part 1 of 17)

```
<complexType>
   <sequence>
    <element ref="dadx:documentation"
              minOccurs="0" maxOccurs="unbounded"/>
    <choice>
     <element ref="dadx:DQS"
                  minOccurs="0"/>
     <sequence>
      <element ref="dadx:implements" minOccurs="0"/>
      <element ref="dadx:result_set_metadata"
                     minOccurs="0" maxOccurs="unbounded"/>
      <element ref="dadx:operation"
                     maxOccurs="unbounded"/>
     </sequence>
    </choice>
   </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 77. DADX schema (Part 2 of 17)

```
<element name="DQS">
  <annotation>
   <documentation>
        Defines the DQS tag.
      </documentation>
  </annotation>
  <complexType/>
 </element>
```

Figure 77. DADX schema (Part 3 of 17)

```
 <element name="documentation">
  <annotation>
   <documentation>
        Defines WSDL documentation for the Web service or an operation.
      </documentation>
  </annotation>
  <complexType mixed="true">
   <choice minOccurs="0" maxOccurs="unbounded">
    <any minOccurs="0" maxOccurs="unbounded"/>
   </choice>
   <anyAttribute/>
  </complexType>
 </element>
```

Figure 77. DADX schema (Part 4 of 17)

```
<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>
```

Figure 77. DADX schema (Part 5 of 17)

```
<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"
             maxOccurs="unbounded"/>
   </sequence>
   <attribute name="name"
             type="NCName"
             use="required"/>
   <attribute name="rowName"
             type="NCName"
             use="required"/>
  </complexType>
 </element>
```

Figure 77. DADX schema (Part 6 of 17)

```
<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 runtime.
        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 isdefined
           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 77. DADX schema (Part 7 of 17)

```
 <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>
```

Figure 77. DADX schema (Part 8 of 17)

```
<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="dadx: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 77. DADX schema (Part 9 of 17)

```
<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 refering 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 77. DADX schema (Part 10 of 17)

```
<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 refering 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>
```

Figure 77. DADX schema (Part 11 of 17)

```
<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 77. DADX schema (Part 12 of 17)

```
<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 77. DADX schema (Part 13 of 17)

```
<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>
```

Figure 77. DADX schema (Part 14 of 17)

```
<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"/>
```

Figure 77. DADX schema (Part 15 of 17)

```
<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>
  <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>
```

Figure 77. DADX schema (Part 16 of 17)

```
<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 77. DADX schema (Part 17 of 17)

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66
- "Definition of a DADX file" on page 29

Related tasks:

- "Converting a document type definition to an XML schema" on page 86

Related reference:

- "DADX operation examples" on page 80

Appendix D. Web services 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 by 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. You compute byte k of data8 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 four-thirds (4/3).
4. Prefix the encoded string with "encoded:" to denote that it has been encoded.

Related concepts:

- "Defining the Web service with the document access definition extension file" on page 66
- "Definition of a DADX file" on page 29
- "Testing Web services applications – a scenario" on page 109
- "Overview of the Web services process" on page 32

Related reference:

- Appendix E, "Web services command reference," on page 263

© Copyright IBM Corp. 2003, 2004

Appendix E. Web services 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.

- 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.

- Example:

```
java com.ibm.etools.webservice.rt.dadx.Dadx2Dd
```

Check_install

Validates a DADX file.

- 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
```

Related concepts:

- “Introduction to using DB2 as a Web services provider – WORF” on page 25
- “Web services provider features” on page 30

Related tasks:

- “Generating deployment descriptors” on page 133

Related reference:

- Appendix D, “Web services encoding algorithm,” on page 261
- “Web services samples – PartOrders.dadx” on page 127

© Copyright IBM Corp. 2003, 2004

DB2 Information Integrator documentation

This topic provides information about the documentation that is available for DB2 Information Integrator. The tables in this topic provide the official document title, form number, and location of each PDF book. To order a printed book, you must know either the official book title or the document form number. Titles, file names, and the locations of the DB2 Information Integrator release notes and installation requirements are also provided in this topic.

This topic contains the following sections:

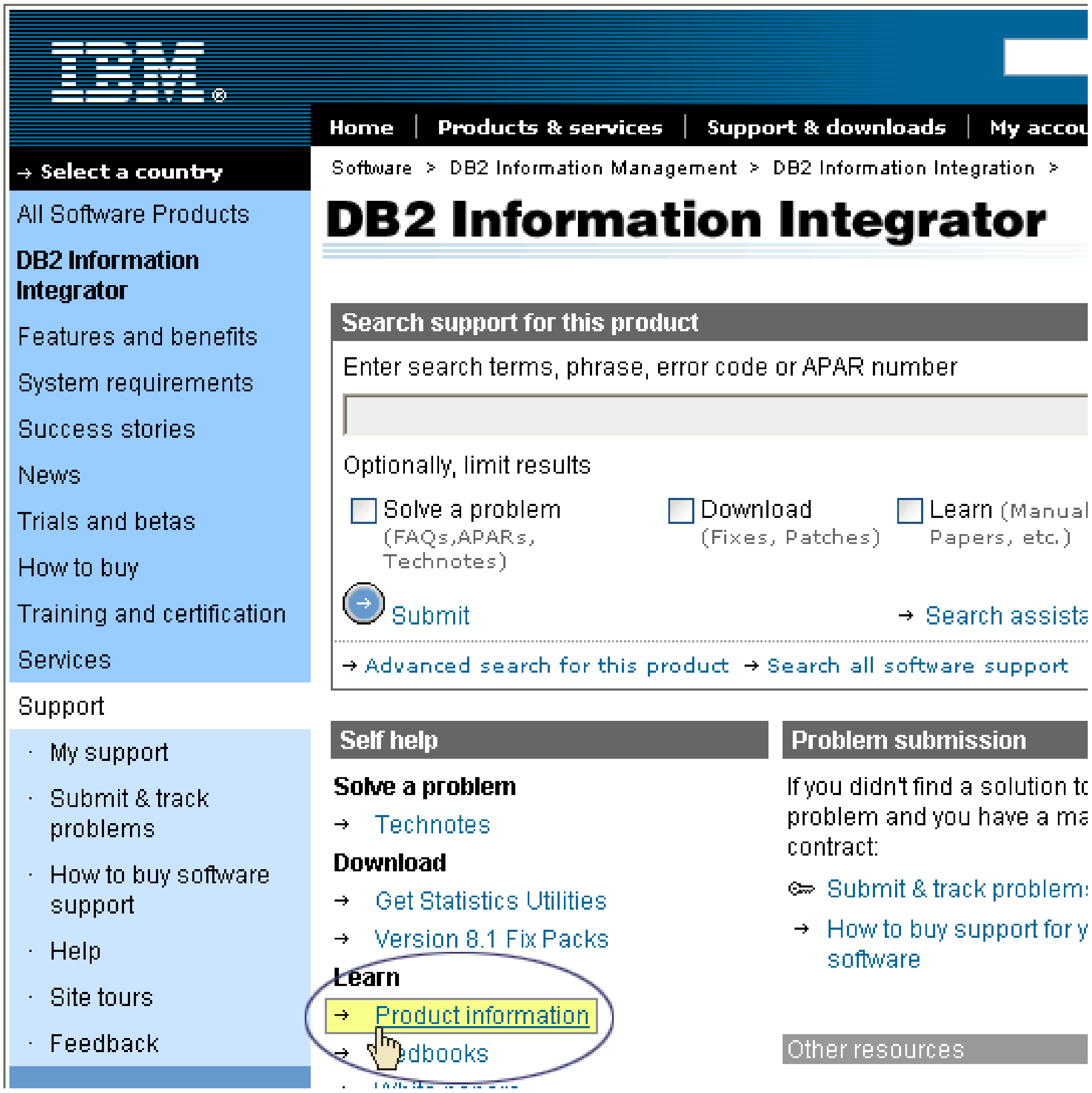
- Accessing DB2 Information Integrator documentation
- Documentation for replication function on z/OS
- Documentation for event publishing function for DB2 Universal Database on z/OS
- Documentation for event publishing function for IMS and VSAM on z/OS
- Documentation for event publishing and replication function on Linux, UNIX, and Windows
- Documentation for federated function on z/OS
- Documentation for federated function on Linux, UNIX, and Windows
- Documentation for enterprise search on Linux, UNIX, and Windows
- Release notes and installation requirements

Accessing DB2 Information Integrator documentation

All DB2 Information Integrator books and release notes are available in PDF files from the DB2 Information Integrator Support Web site at www.ibm.com/software/data/integration/db2ii/support.html.

To access the latest DB2 Information Integrator product documentation, from the DB2 Information Integrator Support Web site, click on the Product Information link, as shown in Figure 78 on page 266.

© Copyright IBM Corp. 2003, 2004


IBM
Home | Products & services | Support & downloads | My accou
→ Select a country

Software > DB2 Information Management > DB2 Information Integration >

DB2 Information Integrator

All Software Products
DB2 Information Integrator
Features and benefits
System requirements
Success stories
News
Trials and betas
How to buy
Training and certification
Services
Support
· My support
· Submit & track problems
· How to buy software support
· Help
· Site tours
· Feedback

Search support for this product

Enter search terms, phrase, error code or APAR number

Optionally, limit results

- [ ] Solve a problem (FAQs,APARs, Technotes)
- [ ] Download (Fixes, Patches)
- [ ] Learn (Manual Papers, etc.)

Submit
→ Search assista
→ Advanced search for this product → Search all software support

Self help

Solve a problem
→ Technotes

Download
→ Get Statistics Utilities
→ Version 8.1 Fix Packs

Learn
→ Product information
→ Redbooks

Problem submission

If you didn't find a solution tc problem and you have a ma contract:

Figure 78. Accessing the Product Information link from DB2 Information Integrator Support Web site

You can access the latest DB2 Information Integrator documentation, in all supported languages, from the Product Information link:
- DB2 Information Integrator product documentation in PDF files
- Fix pack product documentation, including release notes
- Instructions for downloading and installing the DB2 Information Center for Linux, UNIX, and Windows
- Links to the DB2 Information Center online

Scroll though the list to find the product documentation for the version of DB2 Information Integrator that you are using.

The DB2 Information Integrator Support Web site also provides support documentation, IBM Redbooks, white papers, product downloads, links to user groups, and news about DB2 Information Integrator.

You can also view and print the DB2 Information Integrator PDF books from the *DB2 PDF Documentation* CD.

To view or print the PDF documentation:

1. From the root directory of the *DB2 PDF Documentation* CD, open the *index.htm* file.
2. Click the language that you want to use.
3. Click the link for the document that you want to view.

Documentation about replication function on z/OS

Table 30. DB2 Information Integrator documentation about replication function on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication Installation and Customization Guide for z/OS	SC18-9127	DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about event publishing function for DB2 Universal Database on z/OS

Table 31. DB2 Information Integrator documentation about event publishing function for DB2 Universal Database on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about event publishing function for IMS and VSAM on z/OS

Table 32. DB2 Information Integrator documentation about event publishing function for IMS and VSAM on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Event Publisher for z/OS	GC18-9186	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Operations Guide for Event Publisher for z/OS	SC18-9157	DB2 Information Integrator Support Web site

Table 32. DB2 Information Integrator documentation about event publishing function for IMS and VSAM on z/OS (continued)

Name	Form number	Location
Planning Guide for Event Publisher for z/OS	SC18-9158	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about event publishing and replication function on Linux, UNIX, and Windows

Table 33. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site

Table 33. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about federated function on z/OS

Table 34. DB2 Information Integrator documentation about federated function on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Classic Federation for z/OS	GC18-9155	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Transaction Services Guide for Classic Federation for z/OS	SC18-9161	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about federated function on Linux, UNIX, and Windows

Table 35. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows

Name	Form number	Location
Application Developer's Guide	SC18-7359	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site

Table 35. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
C++ API Reference for Developing Wrappers	SC18-9172	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Data Source Configuration Guide	N/A	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Federated Systems Guide	SC18-7364	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Guide to Configuring the Content Connector for VeniceBridge	N/A	DB2 Information Integrator Support Web site
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Java API Reference for Developing Wrappers	SC18-9173	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migration Guide	SC18-7360	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Wrapper Developer's Guide	SC18-9174	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about enterprise search function on Linux, UNIX, and Windows

Table 36. DB2 Information Integrator documentation about enterprise search function on Linux, UNIX, and Windows

Name	Form number	Location
Administering Enterprise Search	SC18-9283	DB2 Information Integrator Support Web site
Installation Guide for Enterprise Search	GC18-9282	DB2 Information Integrator Support Web site
Programming Guide and API Reference for Enterprise Search	SC18-9284	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

Release notes and installation requirements

Release notes provide information that is specific to the release and fix pack level for your product and include the latest corrections to the documentation for each release.

Installation requirements provide information that is specific to the release of your product.

Table 37. DB2 Information Integrator Release Notes and Installation Requirements

Name	File name	Location
Installation Requirements for IBM DB2 Information Integrator Event Publishing Edition, Replication Edition, Standard Edition, Advanced Edition, Advanced Edition Unlimited, Developer Edition, and Replication for z/OS	Prereqs	• The *DB2 Information Integrator* product CD • DB2 Information Integrator Installation Launchpad
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	ReleaseNotes	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site

Table 37. DB2 Information Integrator Release Notes and Installation Requirements (continued)

Name	File name	Location
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

To view the installation requirements and release notes that are on the product CD:

- On Windows operating systems, enter:

  ```
  x:\doc\%L
  ```

  x is the Windows CD drive letter and *%L* is the locale of the documentation that you want to use, for example, *en_US.*
- On UNIX operating systems, enter:

  ```
  /cdrom/doc/%L/
  ```

  cdrom refers to the UNIX mount point of the CD and *%L* is the locale of the documentation that you want to use, for example, *en_US.*

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The following list specifies the major accessibility features in DB2® Version 8 products:

- All DB2 functionality is available using the keyboard for navigation instead of the mouse. For more information, see "Keyboard input and navigation."
- You can customize the size and color of the fonts on DB2 user interfaces. For more information, see "Accessible display."
- DB2 products support accessibility applications that use the Java™ Accessibility API. For more information, see "Compatibility with assistive technologies" on page 276.
- DB2 documentation is provided in an accessible format. For more information, see "Accessible documentation" on page 276.

Keyboard input and navigation

Keyboard input

You can operate the DB2 tools using only the keyboard. You can use keys or key combinations to perform operations that can also be done using a mouse. Standard operating system keystrokes are used for standard operating system operations.

For more information about using keys or key combinations to perform operations, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard navigation

You can navigate the DB2 tools user interface using keys or key combinations.

For more information about using keys or key combinations to navigate the DB2 Tools, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard focus

In UNIX® operating systems, the area of the active window where your keystrokes will have an effect is highlighted.

Accessible display

The DB2 tools have features that improve accessibility for users with low vision or other visual impairments. These accessibility enhancements include support for customizable font properties.

Font settings

You can select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

For more information about specifying font settings, see Changing the fonts for menus and text: Common GUI help.

© Copyright IBM Corp. 2003, 2004

Non-dependence on color

You do not need to distinguish between colors in order to use any of the functions in this product.

Compatibility with assistive technologies

The DB2 tools interfaces support the Java Accessibility API, which enables you to use screen readers and other assistive technologies with DB2 products.

Accessible documentation

Documentation for DB2 is provided in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

Syntax diagrams are provided in dotted decimal format. This format is available only if you are accessing the online documentation using a screen-reader.

Related concepts:

- "Dotted decimal syntax diagrams" in the *Infrastructure Topics (DB2 Common Files)*

Related tasks:

- "Keyboard shortcuts and accelerators: Common GUI help"
- "Changing the fonts for menus and text: Common GUI help"

Bibliography

- *WebSphere: WebSphere Solution Bundles: Implementation and Integration Guide*, SG24-6550
- *MQSeries: Application Messaging Interface*, SC34-5604-07
- *Information Integrator: Planning, Installation and Configuration Guide*
- *Information Integrator: Federated Systems Guide*
- *IBM DB2 Universal Database: Life Sciences Data Connect Planning, Installation, and Configuration Guide*, GC27-1235
- *IBM DB2 Universal Database: XML Extender Administration and Programming*, SC27-1234
- *IBM DB2 Universal Database: Data Warehouse Center Administration Guide*, SC27-1123
- *IBM DB2 Universal Database: Replication Guide and Reference*, SC27-1121
- *IBM Enterprise Information Portal for Multiplatforms: Planning and Installing Enterprise Information Portal*, GC27-0873
- *IBM Enterprise Information Portal for Multiplatforms: Managing Enterprise Information Portal*, SC27-0875
- *IMS: Administration Guide: Database Manager*, SC26-9419-02
- *IBM WebSphere: Application Server V4 for z/OS and OS/390: Installation and Customization*, GA22-7834-05
- *IBM WebSphere: Application Server V4.0.1 for z/OS and OS/390: System Management Scripting API*, SA22-7839
- *DB2 XML Extender: Administration and Programming*
- *DB2 XML Extender: DB2 XML Extender Administration and Programming, Version 7.2 Release Notes*
- *Using WSDL in a UDDI Registry 1.07*
- *WebSphere Handbook*
- *Web Services Description Language (WSDL) 1.1*
- *Dynamic e-business: The next stage of e-business and Web services*
- *Web services zone*

© Copyright IBM Corp. 2003, 2004

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in 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 106-0032, 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.

© Copyright IBM Corp. 2003, 2004

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 Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

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

The licensed program described in this 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 contains 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 contains 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. 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 IBM's application programming interfaces.

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

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years*_. All rights reserved.

Trademarks

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

IBM
AIX
DB2
DB2 Extenders
DB2 Universal Database
Domino
Informix
Intelligent Miner
Lotus
OS/390
UNIX
WebSphere
z/OS

The following terms are trademarks or registered trademarks of other companies:

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

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

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation 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.

Index

© Copyright IBM Corp. 2003, 2004

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:
- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

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

Product information

Information about DB2 Information Integrator is available by telephone or on the Web.

If you live in the United States, you can call one of the following numbers:
- To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
- To order publications: 1-800-879-2755

On the Web, go to www.ibm.com/software/data/integration/db2ii/support.html. This site contains the latest information about:
- The technical library
- Ordering books
- Client downloads
- Newsgroups
- Fix packs
- News
- Links to Web resources

Comments on the documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation. You can use any of the following methods to provide comments:
- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Printed in USA

SC18-7359-01

Spine information:

IBM DB2 Information Integrator Application Developer's Guide Version 8.2