Information Management software

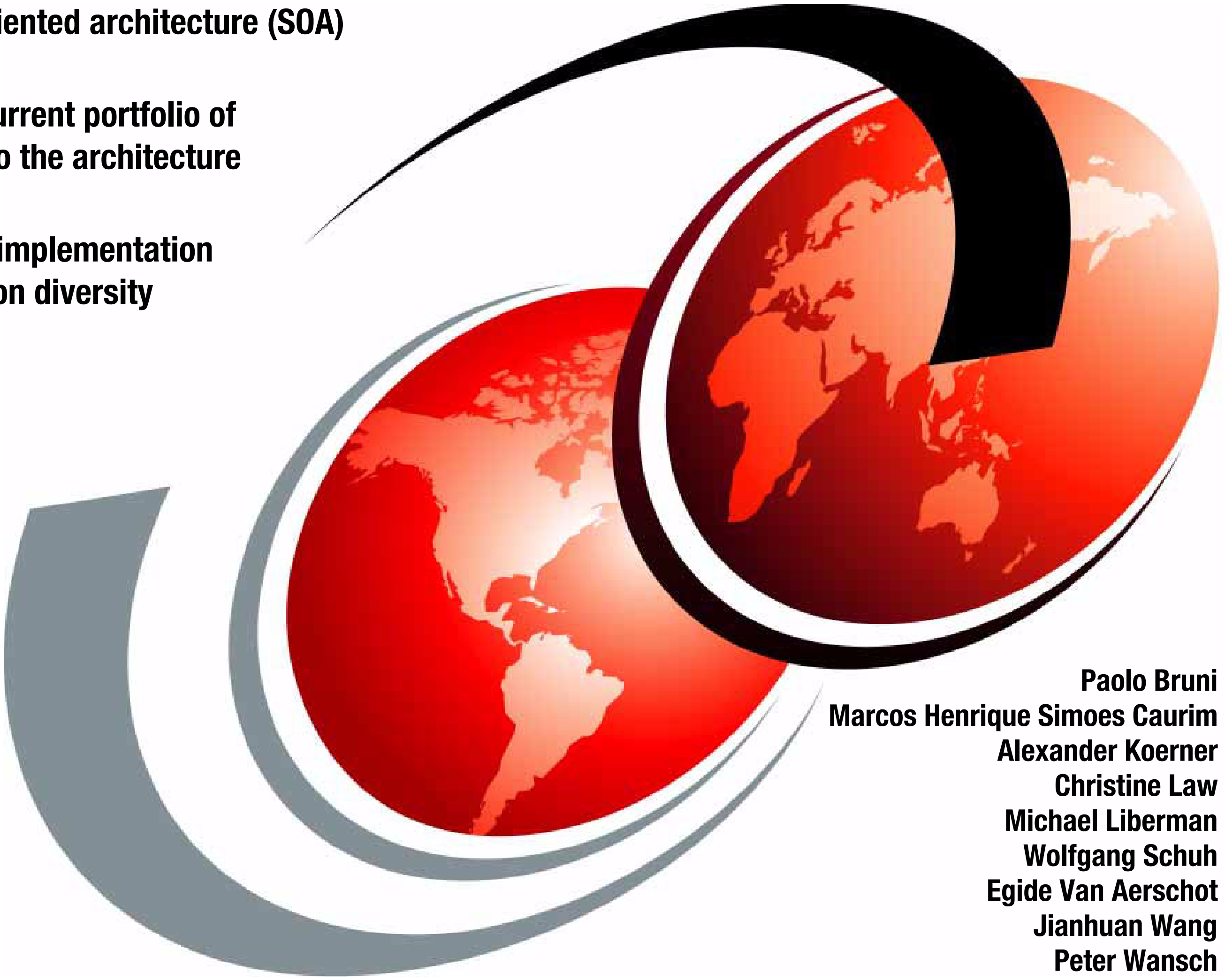
IBM

Powering SOA with IBM Data Servers

Understand the role of data servers within service-oriented architecture (SOA)

Map the current portfolio of products to the architecture

Follow an implementation premised on diversity



Paolo Bruni
Marcos Henrique Simoes Caurim
Alexander Koerner
Christine Law
Michael Liberman
Wolfgang Schuh
Egide Van Aerschot
Jianhuan Wang
Peter Wansch

Redbooks

ibm.com/redbooks

International Technical Support Organization

Powering SOA with IBM Data Servers

December 2006

SG24-7259-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xxvii.

First Edition (December 2006)

This edition applies to DB2 UDB for z/OS Version 8 (program number 5625-DB2) and DB2 Version 9.1 for z/OS (program number 5635-DB2). It also applies to DB2 UDB for Linux, UNIX and Windows Version 9.1, IMS Version 9 and Informix Dynamic Server Version 9.

© Copyright International Business Machines Corporation 2006. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2006. All rights reserved.

Figures

© Copyright IBM Corp. 2006. All rights reserved.

Tables

© Copyright IBM Corp. 2006. All rights reserved.

Examples

© Copyright IBM Corp. 2006. All rights reserved.

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 other 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.

The following paragraph does not apply to the United Kingdom or any other country 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.

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.

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 illustrates 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.

Trademarks

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

developerWorks®
eServer™
ibm.com®
iSeries™
i5/OS®
pSeries®
z/OS®
zSeries®
AIX®
ClearCase®
ClearQuest®
Cloudscape™
CICS®
Database 2™
DataBlade™
DataPower®
Distributed Relational Database Architecture™
Domino®
DB2 Connect™
DB2 Universal Database™
DB2®
DRDA®
Encina®
Enterprise Storage Server®
Everyplace®
Informix®
IBM®
IMS™
Language Environment®
Lotus Notes®
Lotus®
MVS™
MVS/ESA™
Notes®
OMEGAMON®
OS/2®
OS/390®
OS/400®
Parallel Sysplex®
Power PC®
PowerPC®
PurifyPlus™
Rational Suite®
Rational Unified Process®
Rational®
Redbooks™
Redbooks (logo) ™
RequisitePro®
RACF®
System z™
Team Unifying Platform™
Tivoli Enterprise™
Tivoli®
VisualAge®
VTAM®
WebSphere®
Workplace™
Workplace Client Technology™
Workplace Collaborative Learning™
Workplace Forms™
Workplace Managed Client™
Workplace Messaging®
Workplace Team Collaboration™
Workplace Web Content Management™

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

ITIL, is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

EmbeddedJava, Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaMail, JavaOS, JavaScript, JavaServer, JavaServer Pages, JDBC, JDK, JRE, JSP, JVM, J2EE, Solaris, Streamline, Sun, Sun Microsystems, 100% Pure Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ActiveX, Expression, Internet Explorer, Microsoft, Visual Basic, Visual Studio, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

i386, Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

Flexibility in business has become equal in importance with operational efficiency. Service-oriented architecture (SOA) can help businesses respond more quickly and cost-effectively to the changing market conditions by promoting reuse and interconnection of existing IT assets rather than time-consuming and costly reinvention.

SOA has been the top fashionable topic in IT for a few years now. This is because there is a consensus of opinions among enterprise architects that SOA is the key to making the IT department a catalyst for growth and innovation.

This IBM® Redbook helps you get started with SOA by showing the implementation of the minimum requirements: The creation of Web services that allow access to data that is stored in data servers or applications and the realization of interaction services for business to consumer integration. The data servers included in our scenario are DB2 for z/OS, DB2 for Linux, UNIX and Windows, Informix Dynamic Server and IMS.

This redbook is a roadmap showing how SOA can significantly improve the IT business value.

The team that wrote this redbook

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

Paolo Bruni is an Information Management Project Leader at the International Technical Support Organization, San Jose Center, since 1998. In this capacity he has authored several Redbooks on DB2® for z/OS® and related tools, and he has conducted workshops and seminars worldwide. During Paolo’s many years with IBM, in development and in the field, his work has been related mainly to database systems.

Marcos Henrique Simoes Caurim is a Senior IT Specialist in the DBA Service Center at IBM Brazil, São Paulo. He has been working in IT for eight years. During this time, he has worked on several projects with a Brazilian Bank and six of these years while working with DB2 for z/OS. He holds a master’s degree of Computer Science from the University Paulista, São Paulo, Brazil. He is a certified DB2 administrator on OS/390®. During the past two years Marcos has worked in the DBA Service Center (IBM Services), supporting DB2 for z/OS IBM internal accounts.

Alexander Koerner is a certified Senior IT Specialist in the EMEA Information Management Technical Sales organization, based in Munich, Germany. He joined Informix® in October 1989. He was instrumental in starting and leading the SAP/R3 on Informix project, developing an Informix adaptor to Apple's (NeXT's) Enterprise Object Framework, and contributing to the success of many strategic projects across the region. Alexander is currently leading the technical sales activities in EMEA for Informix 4GL to EGL conversions, internal and external enablements on EGL, and also actively covers topics such as IDS, XML, Web services, and DataBlade™ technology. His activities also include presentations at conferences and events, such as the IBM Information Management Conference, IBM Informix Infobahns, regional IUGs, XML One, and ApacheCon. He is a member of the German Informatics Society and holds a master’s degree in Computer Science from the Technical University of Berlin.

© Copyright IBM Corp. 2006. All rights reserved.

Christine Law is an IBM Certified Solutions Expert with the DB2 Advanced Support Team at the IBM Toronto Laboratory. She holds a bachelor's degree of Mathematics in Computer Science from the University of Waterloo, Canada. She has extensive application development experience on Linux®, UNIX® and Windows® platforms with different programming languages and scripting languages. She is a developerWorks® author and her expertise is in DB2 application development, specializing in JDBC™, SQLJ, stored procedures and embedded SQL.

Michael Liberman is a Senior Consultant and an IT Architect, working for Tangram Soft in Israel. He has 10 years of experience in the Information Technology field as a Consultant, Developer, Designer, Analyst, Architect, and Team Leader. He is an IBM DB2 Certified Solution Expert. His areas of expertise include DB2 administration on all platforms, WebSphere® Application Server, Java™ technology, object-relational mapping, and the design of large scale information systems. He was a member of the International DB2 User Group planning committee (IDUG). He holds a bachelor's degree in Computer Science from the Singalovski College in Tel-Aviv.

Wolfgang Schuh is an IT Architect within BCS Application Services, IBM Austria. He has worked in various business transformation projects for the last six years. During that time, Wolfgang has assumed a variety of IT project roles, including Systems Analyst, IT Team Lead, and Application Architect. His current involvement is in J2EE™-based projects, employing Portal and Web Services technologies. He holds a master's degree in Computer Science from the University of Technology, Vienna.

Egide Van Aerschot has been working for the IBM Program Support Center in Montpellier, France since 1998 and is a member of the New Technology Center team, supporting and providing education for J2EE projects, IBM WebSphere Business Integration, and connections to established systems. Before his current position, he worked for IBM Belgium as an Account Systems Engineer, responsible for many projects related to transactional processing with major Belgium customers. During this period, he also participated in several residencies in the United States. He graduated from the University of Louvain as a Civil Engineer.

Jianhuan Wang (Max) is a Senior Certified IT Architect at the IBM Service organization, based in South Carolina. Before joining IBM, he worked at the Verizon Telecommunication Corp. in Atlanta, Georgia. He has 15 years experience in the IT industry. His areas of expertise include cross-platform architecture design, OOA, SOA and infrastructure solution. Max holds a bachelor's degree from East China University of Science and Technology in Shanghai, China; a master's degree in Computer Science from the University of Maryland; and an MBA from Salisbury University in Maryland, U.S.A.

Peter Wansch is a Software Engineer in DB2 for z/OS at IBM Silicon Valley Laboratory in San Jose, California. He has nine years of experience in DB2 client and Web application development as a Consultant, Developer, Architect, and Project Manager. He holds a master's degree in Computer Engineering from the University of Technology, Vienna, Austria. He is a Sun™ Certified Java developer, certified DB2 administrator on Linux, UNIX, Windows and OS/390, and a certified AIX® administrator. He has written extensively about Java network programming and media streaming. Peter worked for three years at the Toronto Laboratory where he was responsible for the development of the DB2-provided stored procedures that are used by Control Center and SAP® Management Console CCMS.

Left to right: Christine, Wolfgang, Marcos, Michael, Paolo, Alexander, Max, Egide and Peter (photo courtesy of Nagraj Alur)

Special thanks to Keshava Murthy and Sitaram Vemulapalli, developers of IBM Informix Dynamic Server's SQL and Extensibility component, for providing the section on WebSphere Message Queuing and IDS.

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

Rich Conway
Bob Haimowitz
Emma Jacobs
Leslie Parham
Sangam Racherla
International Technical Support Organization

Kenneth Blackman
Stephen Brodsky
Kyle Charlet
Curt Cotner
Haley Fung
Shyh-Mei Ho
Christopher Holtz
Rose Levin
Susan Malaika
Kevin McBride
Angela Migliaccio
Roger Miller
Betty Patterson
Bryan Patterson
Vivek Prasad

Klaus Roder
Michael Schenker
Maryela Weihrauch
Peggy Zagelow
Maureen Zoric
IBM Silicon Valley Lab

Scott Lashley
IBM Portland

Mac Devine
IBM Raleigh

Jonathan Leffler
IBM Menlo Park

Peter Kovari
EMEA Software Lab Services, Hursley, UK

Rod Fleming
IBM Dallas

Keshava Murthy
Sitaram Vermulapalli
IBM Menlo Park, CA,Informix Development

Srini Bhagavan
Shawn Moe
Lenexa

Peter Kohlmann
Randy Horman
Tim Vincent
IBM Toronto Lab

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

General introduction to SOA

In Part 1 we provide a brief introduction to SOA in these chapters:

SOA is being implemented by many enterprises, and several vendors are accelerating the pace in providing tools and services to help in the implementation.

This Web site is a good starting point to verify what IBM provides in this area:

`http://www-306.ibm.com/software/solutions/soa/?ca=dti-tilesoa&S_TACT=106AH51W`

© Copyright IBM Corp. 2006. All rights reserved.

1

SOA: Why it is good for you

Have you been worried about the increasing cost and complexity of IT solutions? Do you want to integrate different systems together? Do you want to intelligently recycle existing strategic applications?

This chapter briefly introduces the main concepts of Standard Oriented Architecture (SOA) in these topics:

- What is service-oriented architecture?
- Why is SOA important to our clients?
- How is IBM delivering SOA solutions?
- The service-oriented enterprise

© Copyright IBM Corp. 2006. All rights reserved.

1.1 What is service-oriented architecture?

Service-oriented architecture (SOA) is a collection of services on a network where they communicate with one another in order to carry out business processes. The communication can either be either data passing or trigger several services implementing some activity. The services are loosely coupled, have platform-independent interfaces and are fully reusable. SOA can be considered an evolution in the architecture for IT solutions: It capitalizes on best practices of previous architectures. SOA brings the framework from which to build the solutions required today: more dynamic and with less integration costs. SOA is the architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners.

For more information about SOA, see IBM Systems Journal Vol. 44, No. 4, 2005 - Service-Oriented Architecture, and the redbook *The Value of the IBM System z and z/OS in Service-Oriented Architecture*, REDP-4152, and this Web site:

http://www.ibm.com/software/solutions/soa/?ca=dti-tilesoa&S_TACT=106AH51W

1.2 Why is SOA important to our clients?

The main problem of most corporate IT systems is due to the way companies automate their activities. This has typically involved separate departments and lines of business independently trying to optimize their own functions, and as a result, paying less attention to broader business objectives.

IT client today want open rather than proprietary software; quantifiable business benefits, rather than technology for its own sake; and fully integrated business processes and solutions tailored to their needs, rather than piece-part products that do not work together.

Change and flexibility are key to innovation, and a service-oriented architecture enables the high levels of business adaptability that companies need to truly innovate.

1.3 How is IBM delivering SOA solutions?

A service-oriented approach means looking at a business as linked services and considering the outcomes they bring. Built on Web services and open standards, IBM's SOA solutions help businesses tap into their existing technology investments and link together previously fragmented data and business processes on demand – creating a more complete view of operations, potential bottlenecks and areas for growth.

The ideas behind SOA are not new, but advances in standards and development tools have now made them easier to develop. Rather than designing applications from the ground up, SOA allows developers to reuse code between departments and combine resources from all over the company. SOA applications are also designed so that different parts can operate independently of one another. Because of this, any one feature can be changed without breaking other parts of the application. This makes an SOA much more responsive to changing business requirements than traditional software development, where one feature change could derail an entire application.

Companies that master SOA technology are able to operate more efficiently than their competitors and are quicker to adapt to changing business conditions in their industries.

For example, a retailer deciding whether to issue a credit card to a customer could use the technology to tap different sources and pull together information about a customer's credit-worthiness and buying habits. A bank can use the same computing services to handle account transfer requests, whether they are coming from a teller, an ATM, or a Web application, avoiding the need for multiple applications. A manufacturer could measure more closely what is happening in its production process, then make adjustments that feed back instantly through its chain of suppliers.

The bottom line is more flexible, less costly solutions.

Products

IBM announced new software supporting the entry points, a set of industry-specific models to support SOA and several new services designed for the successful creation of an SOA throughout its life cycle. In addition, IBM Global Business Services (formerly BCS) consultants continue to enhance their SOA skills and knowledge of the entry points to assist clients in implementing successful SOA projects.

This new software helps with the integration across the solution islands implemented through the years and currently available on existing systems.

IBM has announced new software and services to enable customers to take advantage of the strategic trend towards SOA.

Major barriers to SOA success are often related to determining how to get into an SOA, avoiding additional costs and ensuring that investments are allocated toward a business strategy that will withstand market fluctuations and company changes. The new software and services are said to help address these challenges and are based on IBM's expertise with SOA customer engagements in companies of all sizes across all industries.

IBM has identified five entry points to enable customers to more easily approach and initiate an SOA project. These entry points include people-, process- and information-centric approaches as well as connectivity and the ability to reuse existing assets. The company has announced four new software releases targeted toward these SOA starting points as well as a set of industry-specific models to support SOA. The company has also announced new SOA services.

For supporting a people-centric approach to SOA, WebSphere Portal version 6.0 integrates IBM Workplace™ and collaborative technologies, making it easier for users to build and deploy composite applications that can be tailored by industry, role or task. The new release takes advantage of AJAX to create a more responsive user environment. It provides a workflow builder that uses the process engine from WebSphere Process Server, open standards-based software powered by WebSphere Enterprise Service Bus (ESB) that helps simplify the integration of business processes. These capabilities, along with templates designed to accelerate application deployment, allow organizations to increase the effectiveness of their business and IT staff through the application of an SOA.

To improve business visibility and deliver a process-centric approach to SOA, IBM announced WebSphere Business Monitor version 6.0. The software provides an aerial view of the business and enables customers to proactively identify potential issues before they impact productivity. New features in WebSphere Business Monitor include business alerts, links to third-party reports that combine real-time performance and historical analysis, and scorecards to track the status and metrics of projects. For example, the software allows a business user to create an automated alert signaling each time a competitor makes a market-shifting move. The alert can be integrated with third-party research, customer feedback and recommended next steps that could help minimize the impact of a competitor's move on the market.

For an information-centric approach to SOA, IBM has delivered industry-specific models to help clients successfully launch their SOA initiatives. The enhanced IBM Banking Information FrameWork and IBM Insurance Application Architecture models provide a set of critical processes, workflows and activities to help organizations re-engineer their business processes to implement strategic initiatives such as master data management. These IBM models, methodologies and services bring together key components to provide a single view of customer information, thereby enabling a successful SOA and enterprise master data management strategy.

WebSphere Application Server

WebSphere Application Server provides the runtime environment for J2EE applications. WebSphere Application Server provides the J2EE containers required to execute the applications, as well as the services that enable the execution of specific Java application components. WebSphere Application Server hosts a JMS-based messaging engine that provides enhanced messaging functionality and a Web services engine that is capable of hosting and invocating SOAP-based Web services.

WebSphere MQ

WebSphere MQ provides the transport mechanism for messages. WebSphere MQ supports assured, asynchronous, once-only delivery of messages across a broad range of hardware and software platforms like Windows, z/OS, .NET, and J2EE and is accessible using various programming languages and interfaces like Visual Basic®, C++, Java, COBOL, PL/I, as well as MQI and JMS.

WebSphere Enterprise Service Bus

The enterprise service bus (ESB) layer provides flexible connectivity and integration for Web services and messaging-focused applications.

WebSphere Enterprise Service Bus (WebSphere ESB) provides ESB functionality for standards-based applications. Built on WebSphere Application Server, WebSphere ESB takes advantage of its many features, including high availability, scalability, and performance. Beyond the basic programming-based mediation functionality available in WebSphere Application Server, WebSphere ESB provides a mediation layer with pre-built mediations for XML transformation, content-based routing, and message logging. Like WebSphere Process Server, WebSphere ESB supports Service Component Architecture (SCA) and Service Data Objects (SDO) to provide a unified programming model.

WebSphere Message Broker

WebSphere Message Broker provides advanced ESB functionality for universal support of messaging applications. Built on WebSphere MQ, WebSphere Message Broker takes advantage of the services provided by the messaging infrastructure and enhances them by adding a runtime environment that supports message processing like message transformation and routing.

WebSphere Process Server

WebSphere Process Server provides services to enable the integration of composite applications. It addresses issues of system and technology heterogeneity by applying the IBM SOA programming model consisting of SCA and SDO. SCA and SDO allow the definition of service component interfaces, implementations, and references in a technology-neutral way that can be bound later to the technology chosen. On top of this technology-neutral service component layer is a Business Process Execution Language (BPEL)-based orchestration and runtime environment for the dynamic composition of the service components to processes.

WebSphere DataPower SOA Appliances

IBM WebSphere DataPower® SOA Appliances represent an important element in IBM's holistic approach to service-oriented architecture (SOA). IBM SOA appliances are purpose-built, easy-to-deploy network devices that simplify, help secure, and accelerate your XML and Web services deployments while extending your SOA infrastructure. These new appliances offer an innovative, pragmatic approach to harness the power of SOA while simultaneously enabling you to leverage the value of your existing application, security, and networking infrastructure investments.

IBM Rational Software

The IBM Rational® Software Development Platform helps companies transition to a business-driven development environment for SOA by providing tools and process guidance that leverage industry-standard and emerging programming models to simplify and accelerate business process modeling and the design, construction and assembly of service-oriented applications.

The Rational Software Development Platform helps companies to:

- ► Ensure business requirements drive services design and construction
- ► Model and assemble service-oriented applications that automate and integrate business processes
- ► Re-purpose existing assets as services to extend their utility and assemble them into new solutions
- ► Deliver high quality services-based solutions on time and on budget

Services

IBM Global Services is strengthening its SOA capabilities with new services designed for the successful creation of an SOA throughout its life cycle - model, assemble, deploy, manage, while ensuring customers are equipped with the knowledge and skills to independently extend an SOA. These new services help customers with readiness, implementation and management of an SOA.

The services include the Infrastructure Services Readiness Engagement for SOA to help assess and ensure that the customers' infrastructure is prepared to undergo the creation of an SOA. The additional IBM Design and Implementation Services for SOA include support for WebSphere middleware, DataPower appliances, and Tivoli® management and security for SOA. For management, IBM now offers the IT Service Management Design for SOA, IBM Performance Management Testing for SOA, and the SOA Business Dashboard and new maintenance services for SOA.

IBM's new SOA entry points skills training program, based on the five entry points to SOA, will also be delivered throughout IBM Global Services. All of the SOA services complement IBM's software and appliances, yet are not a requirement for SOA success.

What clients of IT seek today is very different than what they sought in the past: open rather than proprietary software; quantifiable business benefits, rather than technology for its own sake; and fully integrated business processes and solutions tailored to their needs, rather than piece-part products that do not work together.

1.4 The service-oriented enterprise

A service-oriented approach means looking at a business as linked services and considering the outcomes they bring. Built on Web services and open standards, SOA is a way for businesses to tap into their existing technology investments and link together previously fragmented data and business processes on demand – creating a more complete view of operations, potential bottlenecks, and areas for growth.

1.4.1 SOA at work

The companies that master SOA technology are able to operate more efficiently than their competitors and be quicker to adapt to changing business conditions in their industries.

For example, a retailer deciding whether to issue a credit card to a customer could use the technology to tap different sources and pull together information about a customer's credit-worthiness and buying habits. A bank can use the same computing services to handle account transfer requests, whether they are coming from a teller, an ATM, or a Web application, avoiding the need for multiple applications. A manufacturer could measure more closely what is happening in its production process, then make adjustments that feed back instantly through its chain of suppliers. The bottom line: More flexible, less costly solutions.

1.4.2 The need for governance

SOA requires an efficient technology governance mechanism to make sure that IT efforts meet business needs, and as a means of controlling what services are deployed and how those services are used. Without some level of governance in place, companies could end up producing overlapping and redundant point services that add minimal value to the business as a whole.

- Reduce or focus on synchronous, HTTP-based Web service invocation

  Our redbook greatly focusses on synchronous HTTP Web services calls. In reality a product like Message Broker would be used more prominently to queue Web service requests within an intranet infrastructure. For instance in our scenario, the create account Web service call from the e-banking application to the backend data server is an excellent candidate for queuing using message broker. We do not have to implement this, but we should certainly elaborate on alternative implementations. A message broker in fact is a highly integrated part of the Enterprise Service Bus concept.

  See: `http://publib-b.boulder.ibm.com/abstracts/sg247137.html`

- Show alternatives that our customers already use for bringing their earlier or existing data to the Web

  Many customers use classic federation.

  See: `http://www.ibm.com/software/data/integration/iicf/`

  to connect their z/OS data sources to a SOA architecture, by coding their own services and using connectors from classic federation. Also using federation may provide a useful alternative to connect data sources to SOA especially when you have to leverage a large number of not necessarily IBM servers but including IBM servers, for example, a Web service that federates a request to File.Net, Documentum, Oracle and DB2 for z/OS. Both federation products are part of the Information Server platform and make sense to be positioned here. While federation comes with a performance trade-off, it is real time. Data Stage can be used to build consolidated data from different sources. In our example, Data Stage would be used to build the "offering" database for customers that is then quickly accessed by the Web application. Federating that would not be the right approach in this particular example.

  Ascential already has extremely useful development tools that let you point to a stored procedure for instance, and automatically generate and deploy a Web service as part of the Information Server platform. However, Information Server is not for free and will only be used if many, complex data sources exist that need to be managed as a whole. For a shop that only has DB2 for z/OS, it is not the right choice and our focus is on data servers, but it would be worth it's weight in gold if we can intelligently position it. If a shop only has SQL data sources and simple data consolidation needs (for instance to build a combined warehouse), even SQL replication may be a good, simple way to implement that instead of the expensive Information Server platform. The answer what products to use depends on the particular business requirements.

2

SOA: From abstract to concrete

In this chapter we describe the key propositions of implementing a service-oriented architecture (SOA) in these topics:

- The business value of SOA
- Business-centric starting points for SOAs
- Infrastructure starting points for SOA
- An implementation roadmap for SOA
- The scope of this redbook

© Copyright IBM Corp. 2006. All rights reserved.

2.1 The business value of SOA

The key value proposition for implementing a SOA is to enable growth and lower the total cost of ownership (TCO) of your IT assets including data and applications by service-enabling them.

Service-enablement of these assets will improve utilization and integration of existing assets and will allow you to change and adapt your business model in an agile way by building new applications that fully reuse your existing assets.

Existing assets in your portfolio, see Figure 2-1, may include:

- Industry-specific system to manage the products and services you offer such as banking software
- CRM systems manage to manage customer information including all operational data as well as offers for each customer or customer segment
- Calendaring, e-mail, messaging and contact management systems to schedule activities and communication between employees and customers
- Human resource systems to manage employee information and employee services
- Document and records management systems to manage physical and electronic documents and records.

Figure 2-1 Existing IT assets

Typically, these systems are implemented using vendor-provided or in-house software solutions using a variety of technologies on different hardware and software platforms using middleware from a number of vendors. In addition, there exist business processes that define who and how these assets are used for your business to provide products and services to customers while meeting strategic revenue and profit objectives.

2.2 Business-centric starting points for SOA

Through business-centric SOA, companies can tie IT projects to the business need, directly addressing the firm's immediate pain points (Figure 2-2 on page 13).

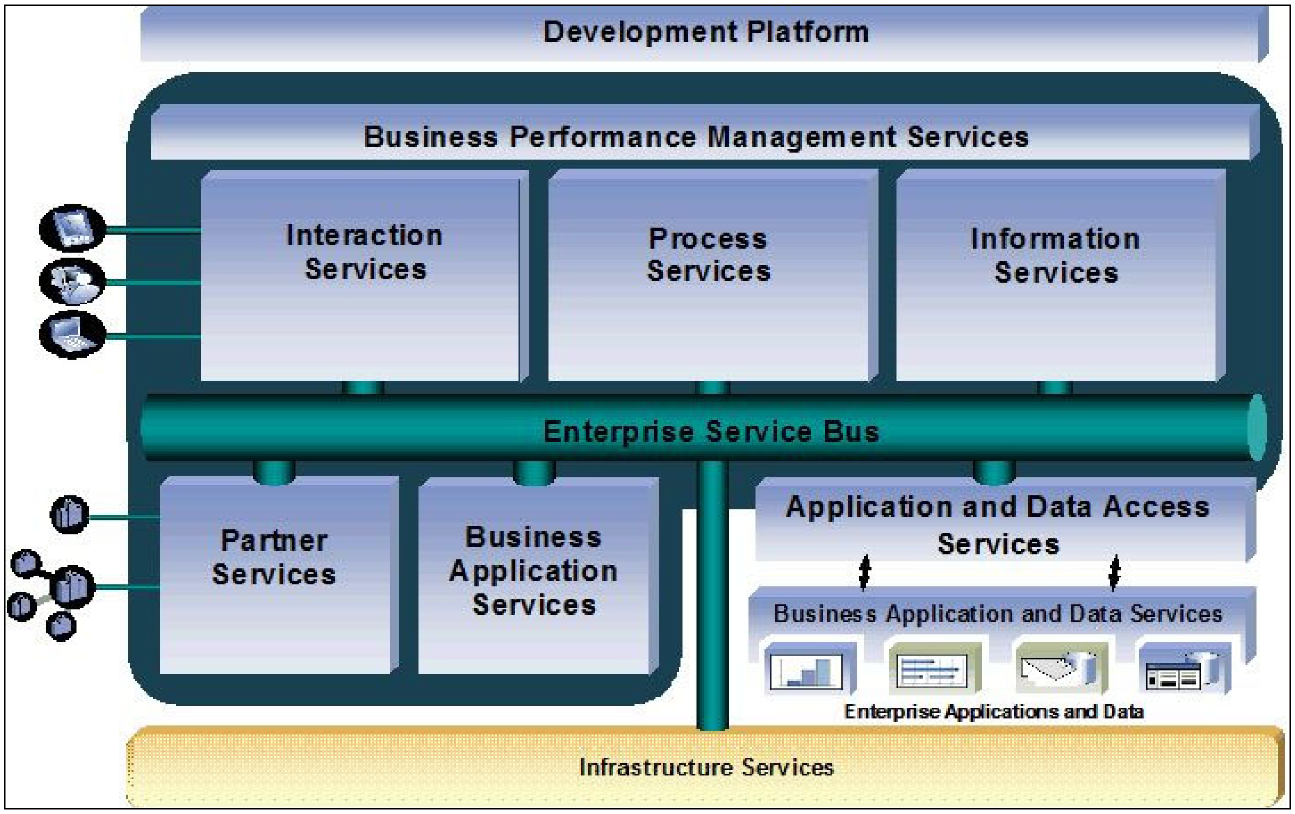

Development Platform
Business Performance Management Services
Interaction Services
Process Services
Information Services
Enterprise Service Bus
Partner Services
Business Application Services
Application and Data Access Services
Business Application and Data Services
Enterprise Applications and Data
Infrastructure Services

Figure 2-2 Business centric SOA

The starting points for implementing SOA are:

- **People**

  Improve productivity though people collaboration using Interaction Services

  In today's challenging business environment, enabling people to rapidly act upon and interact with targeted business process and information is critical to achieving operational efficiency and agility. A business can improve people productivity by aggregating views that deliver information and interaction in the context of a business process. This enables human and process interaction with consistent levels of service.

  You can start by building a view of a key business process by aggregating information to help people make better decisions. Next steps: tighter management of performance with alert-driven dashboards that link to more processes.

  Businesses must also focus on growth – driving the need for tools and solutions that can adapt business processes and tasks and enable greater collaboration and interaction.

  IBM's approach to SOA is a business-centric one, focused on how to enable people's action and interaction. IBMs Workplace software and solutions such as Business Dashboards help to drive business results.

  – Increase organizational productivity and operational efficiency
  – Drive innovation and responsiveness through collaborative interactions
  – Help reduce integration costs and the time to build/deploy new services
  – Increase access and simplify process change and control

  IBM's offerings to improve collaboration through composite applications are:

  – IBM WebSphere Portal

  The foundation for deploying composite applications.
- IBM Workplace Collaboration Services

  A range of reusable collaborative components.
- IBM WebSphere Portlet Factory

  Automate portlet development.
- IBM Workplace Forms™

  Create XML forms-based processes.
- IBM WebSphere Everyplace® Deployment

  Bring information and applications to a wide range of devices.
- IBM Workplace Dashboards

  Prebuilt portlets for real-time visibility to business processes.
- IBM WebSphere Everyplace Connection Manager

  Security-rich access to E-mail, PIM, and Lotus® Domino® and WebSphere-based applications.
- IBM Workplace for Business Strategy Execution

  Helps manage and execute business objectives.
- IBM Workplace Designer

  Easily create components for IBM Workplace applications.
- IBM Workplace for Business Controls & Reporting

  Simplify risk assessment and control management.
- IBM Lotus Notes® - "Hannover"

  Next release of IBM Lotus features composite applications.
- IBM Workplace Managed Client™

  A rich user experience with innovative productivity tools.

- **Process**

  Improve business process management for continuous innovation through process services.

  Deploy innovative business models quickly with re-usable and optimized processes, adapting the enterprise to changing opportunities and threats.

  IBM offers WebSphere software to help improve the efficiency, flexibility, and control of key business processes. IBM combines expertise in your industry with world-class software to deliver a comprehensive approach to business process management (BPM). IBM delivers rich functionality through WebSphere and allows you to model an existing process, remove bottlenecks, optimize and deploy enhanced processes. You can also monitor the process and expand across and beyond the enterprise to suppliers and partners. Leaders in the industry are already using WebSphere to leverage leading best practices, unmatched industry expertise, and the ability to employ hundreds of pre-existing process models to speed time to value.

  - WebSphere Business Modeler and WebSphere Business Monitor

    Metrics of critical business processes and services, and prompted actions to improve performance

    Key Performance Indicators for business units

    National language availability

  - WebSphere Process Server and WebSphere Integration Developer

    Multiple platforms including Systems z availability, and national language support for BPM

    - Complete toolbox for building composite SOA applications
    - Virtual Real-time process enhancements
    - Architected for reuse and flexibility

- **Information**

  Delivering information as a service through Information Services.

  Improve business insight and reduce risk with trusted information services delivered in-line and in-context.

  Information availability is crucial for meeting key business challenges, such as streamlining business processes, meeting customer expectations, and driving employee productivity.

  However, most companies face daunting complexity in their information architectures. The average Fortune 500 company has over 48 disparate financial systems and 3 enterprise resource planning systems. This complexity means that business-critical information can be disaggregated, disorganized and difficult to understand - making it hard to provide the consistent and timely information needed for effective business management.

  In the midst of this information glut, many companies are introducing more flexibility into their applications and processes by re-architecting their systems into smaller, more accessible and reusable services, by leveraging service-oriented architecture (SOA). SOA is an approach that treats elements of business processes and the underlying IT infrastructure as secure, standardized services that can be managed, reused and combined to address changing business priorities.

  As companies go through this transformation process, the maturity of their information architectures can have a large impact upon their results. In fact, Gartner advises that you should “develop an enterprise information management strategy as part of your SOA architecture. You will waste your investment in SOA unless you have enterprise information that SOA can exploit.”

  IBM offers a range of capabilities for deploying and managing information within a SOA that span the SOA life cycle:

- **Model**

  Understand information assets and link to business context

  - Discover information metadata
  - Develop data & content models
  - Map information to business processes

- **Assemble**

  Compose information services

  - Extract, cleanse, transform & federate heterogeneous information

- **Deploy**

  Service information requests

  - Deliver unified data & content
  - Deliver business context
  - Discover relationships

- **Manage**

  Monitor and manage Information

  - Ensure performance, availability and security meet service levels
- **Governance and processes**

  Align business with IT information needs
  - Monitor information usage over time
  - Define and refine information management rules & policies

The IBM Information Management portfolio delivers the necessary building blocks for an information infrastructure – enabling businesses to model, assemble, deploy and manage information to create insight.

- Information Integration services

  Allows real-time, integrated access to business information regardless of location or format. Provides the broadest set of information integration capabilities to semantically align information across disparate sources.
- Content Management services

  Allows information management through its life cycle once it is declared as a critical asset. IBM empowers you to access, protect and deliver more forms of information than any other vendor.
- Business Intelligence services

  Gives you the tools for powerful analysis from integrated data so you can derive insights that give you a competitive advantage.
- Master Data management services

  Enables you to create and manage consolidated “master” data for customer and product information across heterogeneous environments.
- Information accelerators

  Decreases time to value and risk associated with providing information in an SOA, leveraging best practices to support innovative applications and processes developed by IBM or our partner network

2.3 Infrastructure starting points for SOA

From an IT perspective, there are two key starting points to build an Infrastructure that connects your existing services into a SOA: connectivity and reuse.

2.3.1 Connectivity: Underlying connectivity to support business-centric SOA

Connectivity has always been a requirement. But SOA brings new levels of flexibility. As well as acting as a building block for additional SOA initiatives, connectivity provided through SOA has distinct, standalone value.

Flexible business requires flexible IT. Your connectivity infrastructure is critical for integrating your existing and new applications, processes and services. Establish higher levels of service while providing a reliable, highly secure and assured delivery of data across all processes.

IBM can connect more of your assets and services and has the product breadth to support the most connected SOA platforms, to help you maximize re-use.

Start your SOA with WebSphere MQ for reliable service connectivity, WebSphere Enterprise Service Buss (ESB) for Web service-based mediated connectivity or WebSphere Message

Broker, the advanced ESB for robust connectivity beyond Web services. In addition, SOA appliances can enable specialized connectivity needs in an alternate form factor. Now you can add flexibility and speed when developing new customer service processes.

- WebSphere ESB and Message Broker

  IBM Systems z support

  - Expose non-services applications as services
  - WebSphere DataStage TX plug-in for Advanced ESB
  - Additional support for 64 bit platforms
- WebSphere DataPower Appliances

  ESB functionality in an appliance form factor

  - Simplify SOA with drop-in devices
  - Help secure Web services
  - Accelerate and scale SOA with high-performance XML processing
- WebSphere MQ and Extended Security Edition

  Supports message-based connectivity between applications or files

  - Enhances security for your SOA messaging backbone
- WebSphere Adapters

  Rapid connection for hundreds of endpoints into your SOA

  - First class support for SAP and Oracle applications
  - Enhanced support for data, messaging, and Web services

2.3.2 Reuse: Creating flexible, service-based business applications

Cut costs, reduce cycle times and expand access to core applications through reuse. Analysts estimate it is up to five times less expensive to re-use existing applications than to write new applications.

Create and reuse services to increase time to value, plus allow expanded access to core applications-quickly. Start with identifying high-value existing IT assets and service, enable them for reuse or create new services for later reuse and employ a registry or repository to facilitate oversight and control of how you reuse services.

Use portfolio management to consider which assets you need to run your company. Identify high-value existing IT assets and service-enable them for reuse. Satisfy remaining business needs by creating new services. Finally, create a registry/repository to provide centralized access to and control of these reusable services.

Did you know it is five time less expensive to re-use existing applications than to write new applications?

- WebSphere Application Server
  - Enhanced tools and JDK™ 5 innovations make creating and deploying re-usable services quicker and easier
  - New Web services standards make interactions more flexible and secure
  - New, integrated support for virtual real-time multimedia elements, like voice, video and instant messaging
- WebSphere Application Server Community Edition
  - New Web-tier clustering and simplified deployment
  - Improved application portability to WebSphere Application Server Family

  - WebSphere Extended Deployment
  - Now available on IBM System z™
  - Supports new workloads: process server, commerce, and portal
  - Enhanced Reuse of IBM Systems z Assets
- CICS®
  - Enhanced Web services for better access to trusted, high quality applications
- WebSphere Developer for System z and WebSphere Studio Asset Analyzer
  - Visually compose services into process flows
- WebSphere Commerce
  - New cross-channel business services for gift registry and contact center
  - Reuse of e-commerce processes in other channels such as point of sale, call center and kiosk
  - Web services integration with existing order management systems for seamless, cross-channel order and inventory
  - Integrated development environment for Web services

2.4 An implementation roadmap for SOA

Why is SOA different and here to stay?

Its objective is clearly to help businesses grow faster and lowering their total cost of ownership at the same time. SOA is built on open industry standards for which all key vendors in the industry offer components that integrate at some level. It does not lock you into a proprietary technology that ties you to a specific vendor. It ultimately gives customers more choices.

The entry cost for service-enabling your existing IT assets is low so that you can implement proof of concepts solutions quickly. Most vendors provide free of charge ways to enable your existing middle-ware or applications as services and there exist open source or community offerings that will allow you to implement infrastructure as well as interaction and process services for complete SOA solutions so you can immediately see a return on investment. As your adoption of SOA grows, you can choose to add SOA offerings from vendors that will pay off in a very short time by increasing your revenue and decreasing your TCO.

It is no big bang approach. A SOA typically starts in the following sequence:

1. Implementing Access Services to your applications and data

   This gets you started on SOA. The minimum requirement is that you create Web services that allow access to data stored in data servers or applications.

2. Implementing Partner services for business to business process integration

   At a minimum you make Web services available to business partners.

3. Implementing Interaction services for business to consumer integration

   At a minimum you allow employees or customer representatives to access data and application services through Web or client technologies

4. Implementing Information services to leverage data from all your enterprise data for competitive

   Cleans, transform, aggregate information so that they can be consumed by partner and interaction services

5. Implementing Process services to orchestrate your services across your entire SOA

   Fully automate business process by orchestrating access, partner, interaction and information services

6. Implement an Enterprise service bus to improve availability and implement quality of service objectives

   Make your services robust and meet quality of service expectations through implementing an enterprise service bus

7. Implement infrastructure services to connect your SOA to your enterprise infrastructure such as call centers

   Extend the reach of your services infrastructure to infrastructure not typically connected or well integrated in all of your business services

8. Implement Business Application Services to rapidly plug new service offerings into your existing infrastructure

   Plug-in new applications that are integrated into your SOA from day one to rapidly pay off your investment

Depending on your business needs and the size and complexity of your business you may choose to incrementally implement all services in your SOA, or simply up to Step 2 (or any higher).

2.5 The scope of this redbook

This redbook will help you implement Steps 1, 2, 3 listed in the previous section.

Most business applications use a data server to hold enterprise data and they frequently use stored procedures as the business logic that operates on data stored in data servers.

Chapter 3, “Web services and service-oriented architecture” on page 23 describes the technologies used for Web services. If you are already familiar with these technologies, proceed to Chapter 5, “Development tools” on page 103 for an overview of the development tools used for SOA. These chapters will enable you to develop Web services to access data and stored procedures in IBM data servers and how to enable them with partners (for both providing Web services to partners and consuming Web services from partners).

Part 5, “Assembling and developing a scenario” on page 395 shows you how to assemble and develop interaction services using WebSphere portal technology as an example.

Steps 4 to 8 are covered in different Redbooks. In particular, if you want to coordinate your Web services using WebSphere Integration Developer and WebSphere Process Server as described in the redbook *Getting Started with WebSphere Integration Developer and WebSphere Process Server*, SG24-7130.

To learn more about Web services technology in WebSphere Application Server 6 refer to *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

Part 2

SOA technologies

In Part 2 we introduce the fundamental technical components of the SOA. We look at standards and technologies, Web services, XML, and portals in these chapters:

- Chapter 3, “Web services and service-oriented architecture” on page 23
- Chapter 4, “SOA and user interfaces with portals” on page 69
- Chapter 5, “Development tools” on page 103

© Copyright IBM Corp. 2006. All rights reserved.

3

Web services and service-oriented architecture

In this chapter we provide an introduction to service-oriented architecture (SOA). We also introduce Web services as an implementation of SOA. The main goal is to explicitly state the design principles in order to assist architects and designers in creating SOAs that are likely to achieve the benefits of SOA.

In this chapter we discuss these topics:

- Drivers for Web services and SOA
- Standards and technologies for Web services and SOA
- An overview of SOA and Web service
- Web services (WS) and SOA work together
- SOA and Web service architecture design considerations
- Additional information for SOA

© Copyright IBM Corp. 2006. All rights reserved.

3.1 Drivers for Web services and SOA

SOA is currently often implemented with Web services. The sources listed in 3.6, “Additional information for SOA” on page 66 provide more information, but since there is some variation in their description we provide he an introduction in order to place the remaining content of this redbook in context. The IT industry has strived to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The key benefit of the Web service model is that it permits different distributed services to run on variety of software platforms and architectures, and allows them to be written in different programming languages.

The drivers behind this vision include:

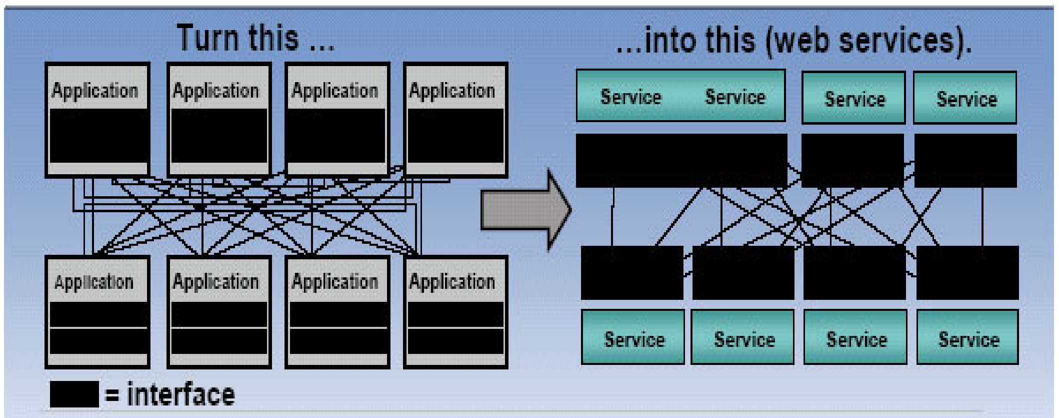
- Increasing the speed at which businesses can implement new products and processes or change existing ones. The greatest strength of Web services is their ability to enable interoperability in a heterogeneous environment. As long as the various systems are enabled for Web services, they can use the services to easily interoperate with each other.
- Reducing implementation and ownership costs. Most enterprises have an enormous amount of data stored in existing enterprise information systems, and the cost to replace these systems may not meet businesses’ expectations. Web services let enterprises application developers reuse and even commodify these existing information assets.
- Enabling flexible pricing models by outsourcing elements of the business or moving from fixed to variable pricing, based on transaction volumes. Web service standards have opened a large marketplace for tools, products, and technologies. This gives organizations a wide variety of choices, and they can select configurations that best meet their application requirements.
- Simplifying the integration work that is required by mergers and acquisitions. Since a main objective of Web services is improving interoperability, exposing existing applications or services as Web services increases their reach to different clients. This occurs regardless of the client’s platform: it does not matter if the client is based on the Java or Microsoft® platforms or even if it is based on a wireless platform. In short, a Web service can help you extend your applications and services to a rich set of client types.
- Achieving better IT utilization and return on investment. Because Web services introduce a common standard across the Web, vendors in the interest of staying competitive, are more likely to develop better tools and technologies. These tools and technologies will attract developers because they emphasize increased programming productivity. As a result, the entire industry benefits.
- Simplifying the enterprise architecture and computing model.

Really achieving these goals affects the entire scope of a business’s processes and IT systems. Although several systems that cover some elements of this scope have been implemented, there has not been a single, broadly accepted approach.

The combination of SOA, an approach that draws together proven techniques from several proceeding architecture and design styles, with new open standards and integration technologies has the potential to provide a consistent approach.

Figure 3-1 on page 25 illustrates how Web services decouples interfaces from applications. SOA uses a programming model that allows a rich abstraction of both the business application and the interface. By abstracting, the interfaces can be clearly separated from the business applications. This enables you to reduce the number and complexity of those interfaces. Also it allows you to reuse both the interfaces and the business applications.

Web service decouples interfaces from applications

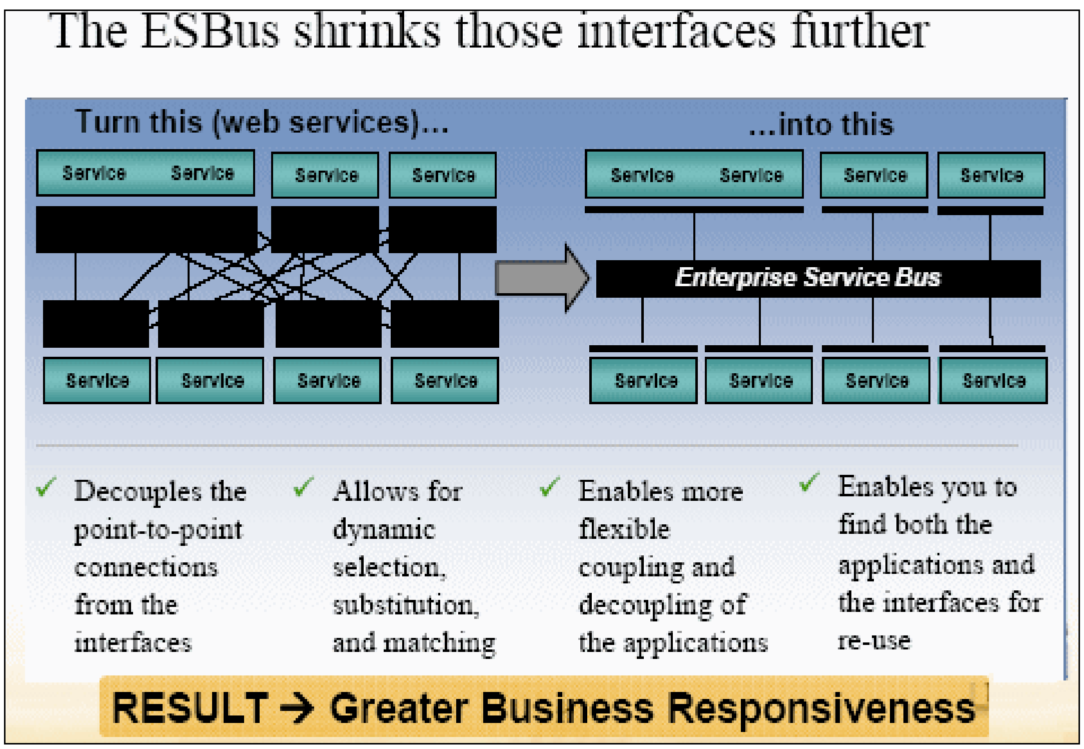


- ✓ Rich business abstractions describe the application interface
- ✓ Decouples the interfaces from the business applications
- ✓ The number and complexity of the interfaces is reduced
- ✓ Business applications and their interfaces become reusable

But separate connection points still leaving bloated interfaces ....

Figure 3-1 Web service decouples interfaces from applications

While we can use a programming model to decouple interfaces with applications, IBM has developed a much efficient tool Enterprise Service Bus (ESB) to manage those interfaces. Figure 3-2 on page 26 illustrates the use of ESB to manage Web services. It virtualizes the interface, or in other words, it decouples the point-to-point connections from the interfaces themselves. The interfaces are put into a third party broker which helps you manage the interfaces better. This enables faster and more flexible coupling and decoupling of applications. Because you can find all of the applications and the interfaces, you can then reuse both. All of these elements are connected through WebSphere ESB which provides connectivity infrastructure for integrating applications and services to power your SOA systems.



The ESBus shrinks those interfaces further

- ✓ Decouples the point-to-point connections from the interfaces
- ✓ Allows for dynamic selection, substitution, and matching
- ✓ Enables more flexible coupling and decoupling of the applications
- ✓ Enables you to find both the applications and the interfaces for re-use

RESULT → Greater Business Responsiveness

Figure 3-2 Enterprise Service Bus (ESB) to manage Web services

To get the most of advantages from SOA, there is a strong synergy between SOA and consistency of business processes across the enterprise. Full scale implementation will involve business process improvement. SOA can be delivered with existing technology. Most of today's production service-oriented architectures do not yet use Web services – they use existing mature technologies, such as XML and asynchronous messaging. Web services offer the benefits of standards and the promise of interoperability. Figure 3-3 on page 27 illustrates an SOA conceptual structure: how the major SOA components work together, and include: presentation layer, business process, services, components and existing resources.

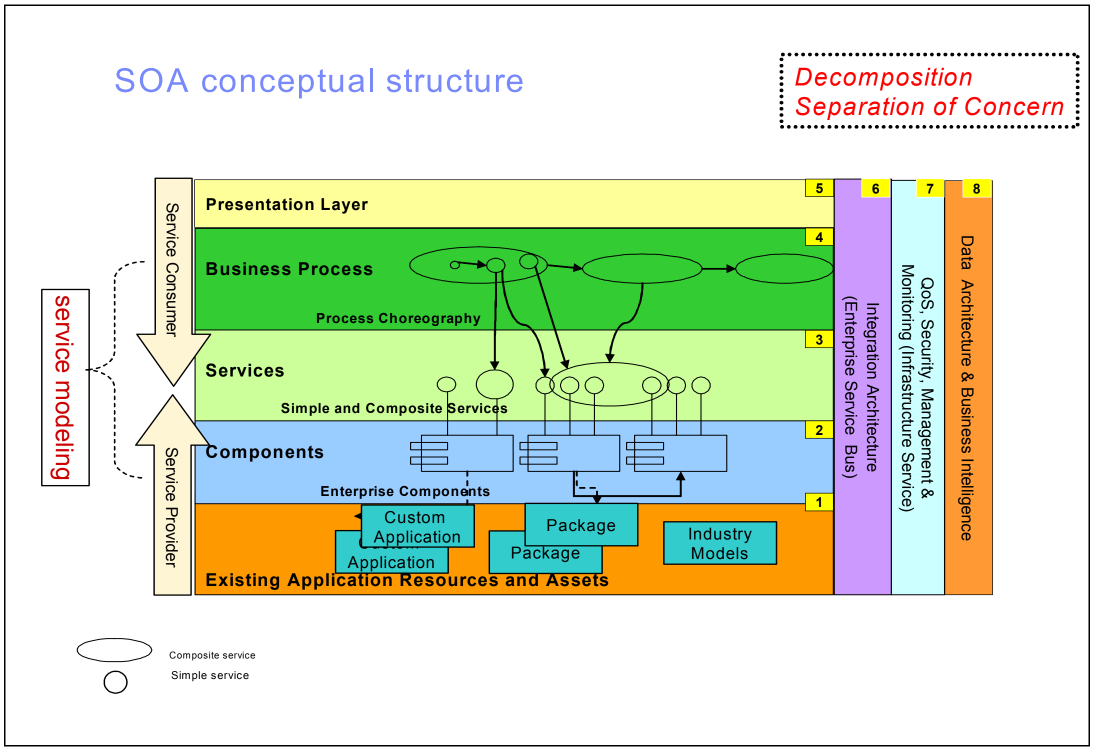


Figure 3-3 SOA conceptual structure

SOA and enables new opportunities for more flexible, rapid, and widespread integration in a model that is consistent with the exposure of business functions as services. In order to achieve those goals, we must follow some common SOA standards and technologies.

3.2 Standards and technologies for Web services and SOA

SOA is an integration architecture approach that is based on the concept of service. The business and infrastructure functions that are required to build distributed systems are provided as services that collectively, or individually, deliver application functionality to either user applications or other services. Web services are a recent set of technology specifications that leverage existing proven open standards such as XML, URL, and HTTP to provide a new system-to-system communication standard. Based on this communication model, additional higher-level Web services standards have also been defined to address transactions, security, business processes, and so forth: the higher-order functions that are required to get systems, applications, and processes (rather than objects and components) talking to each other.

3.2.1 Overview of Web services standards

The Web has brought new customers, new business models, extensions of opportunity, new transparency and improved collaboration between employees and employers, and in some cases reductions in infrastructure costs and complexity. The key to these successes was a universal server-to-client model that is consistent with a highly distributed environment, based on simple open standards and industry support.

Web services promises to do the same thing for the way systems talk to systems: integrating one business directly with another so that the process doesn't have to wait for people. The key is a universal program-to-program communication model based on simple open standards and industry support.

Web service standards establish a base of commonality and enable Web services to achieve wide acceptance and interoperability. Some of standards are core elements such as eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI):

One of the main aims of Web services is to provide a loose coupling between service consumer and service providers. While this is limited to a certain extent by a requirement for the consumers and providers to agree on a WSDL interface definition, Web services have been created with significant flexibility with regard to their location. Figure 3-4 shows the basic interaction model supported by Web services.

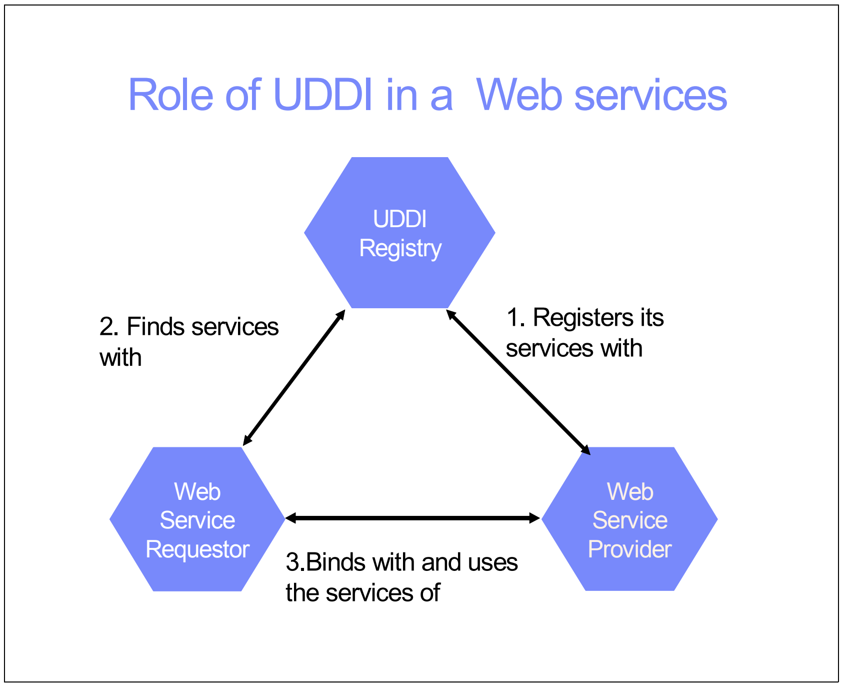

Role of UDDI in a Web services
UDDI Registry
2. Finds services with
1. Registers its services with
Web Service Requestor
Web Service Provider
3.Binds with and uses the services of

Figure 3-4 Web services Model

- **Service Provider:** Provides e-business services and publishes availability of these services through a registry. Each provider must decide which services to expose, how to implement trade-off between security and easy availability, how to price the services (or, if they are free, how to exploit them for other value). The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use the service.
- **Service Consumer:** Contacts the service registry to obtain a reference to a service provider, and the location of the service provider, then makes calls on the service provider.
- **Service Directory:** Provides support for publishing and locating services like telephone yellow pages. The service directory (also known as the service broker) is responsible for making the Web service interface and implementation access information available to any potential service requestor. The implementers of a broker have to make a decision about the scope of the broker. Public brokers are available all over the Internet, while private brokers are only accessible to a limited audience, for example users of a company-wide intranet. Furthermore, the scope of the offered information has to be decided. Some brokers will specialize in breadth of listings. Others will offer high levels of trust in the listed

services. Some will cover a broad landscape of services and others will focus on a given industry. Brokers will also be available that simply catalog other brokers. Depending on the business model, a broker may attempt to maximize look-up requests, number of listings, or accuracy of the listings.

- **Service Requestor:** Locates required services via the Service Broker binds to services via Service Provider. One important issue for users of services is the degree to which services are statically chosen by designers compared to those dynamically chosen at runtime. Even if most initial usage is largely static, any dynamic choice opens up the issues of how to choose the best service provider and how to assess quality of service. Another issue is how the user of services can assess the risk of exposure to failures of service suppliers.

The service-oriented architecture offers the following properties:

- Web services are self-contained.

  On the client side, no additional software is required. A programming language with XML and HTTP client support is enough to get you started. On the server side, merely a Web server and a SOAP server are required. It is possible to Web services enable an existing application without writing a single line of code.

- Web services are self-describing.

  Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration). The definition of the message format travels with the message; no external metadata repositories or code generation tool are required.

- Web services can be published, located, and invoked across the Web.

  This technology uses established lightweight Internet standards such as HTTP. It leverages the existing infrastructure. Some additional standards that are required to do so include SOAP, WSDL, and UDDI.

- Web services are language-independent and interoperable.

  Client and server can be implemented in different environments. Existing code does not have to be changed in order to be Web service enabled.

- Web services are inherently open and standard-based.

  XML and HTTP are the major technical foundation for Web services. A large part of the Web service technology has been built using open-source projects. Therefore, vendor independence and interoperability are realistic goals this time.

- Web services are dynamic.

  Dynamic e-business can become reality using Web services because, with UDDI and WSDL, the Web service description and discovery can be automated.

- Web services can be composed.

  Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation. Web services can be chained together to perform higher-level business functions. This shortens development time and enables best-of-breed implementations.

- Web services build on proven mature technology.

  There are a lot of commonalities, as well as a few fundamental differences to other distributed computing frameworks. For example, the transport protocol is text based and not binary.

- Web services are loosely coupled.

  Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that allows a more flexible re-configuration for an integration of the services in question.

- Web services provide programmatic access.

  The approach provides no graphical user interface; it operates at the code level. Service consumers have to know the interfaces to Web services but do not have to know the implementation details of services.

- Web services provide the ability to wrap existing applications.

  Already existing stand-alone applications can easily be integrated into the service-oriented architecture by implementing a Web service as an interface.

There are a lot of SOA technologies in the marketplace. How can all those technologies and standards work together? Figure 3-5 shows the major SOA component standard stack.

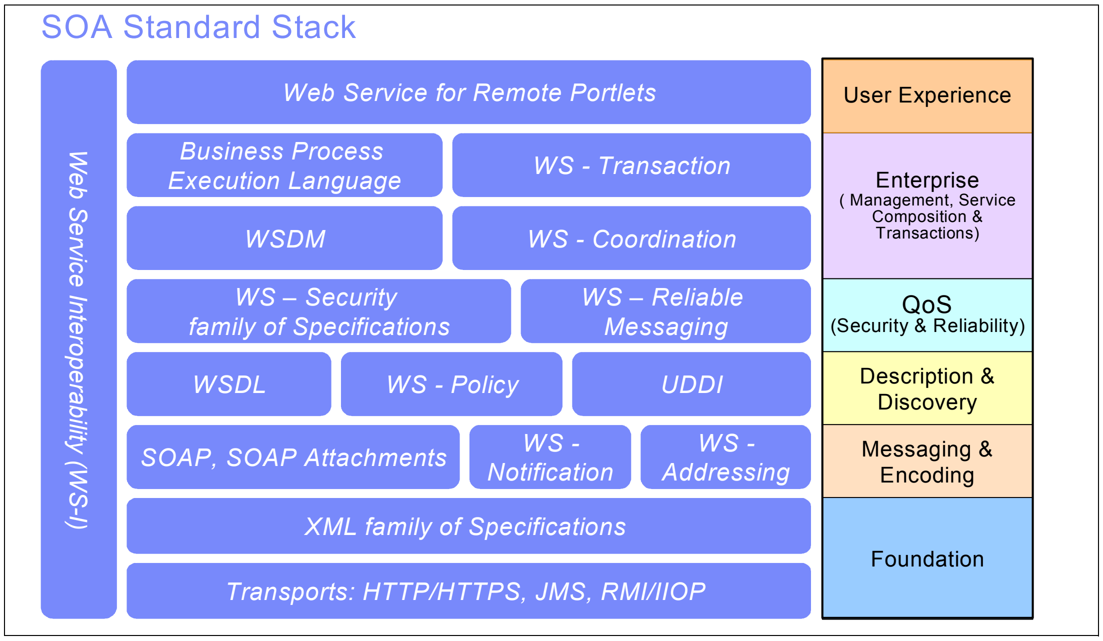
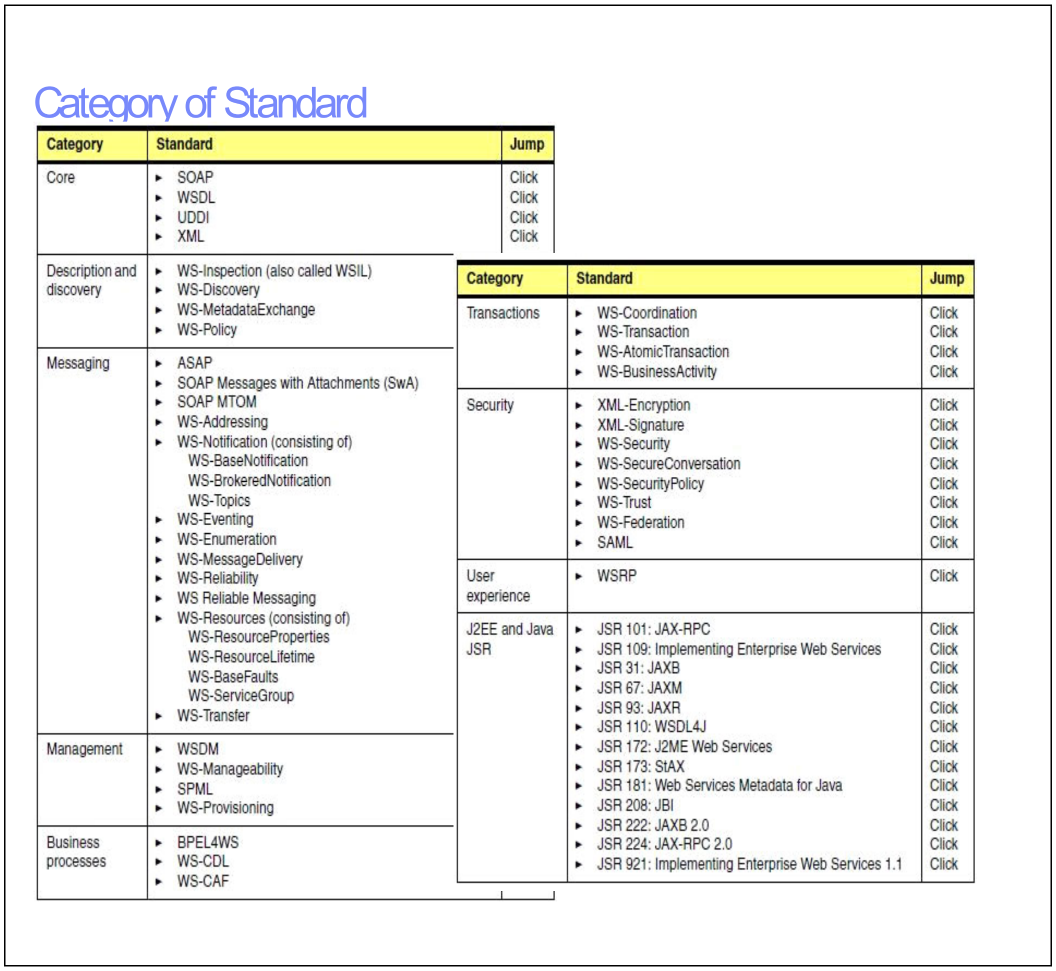

SOA Standard Stack
Web Service Interoperability (WS-I)
Web Service for Remote Portlets
Business Process Execution Language
WS - Transaction
WSDM
WS - Coordination
WS – Security family of Specifications
WS – Reliable Messaging
WSDL
WS - Policy
UDDI
SOAP, SOAP Attachments
WS - Notification
WS - Addressing
XML family of Specifications
Transports: HTTP/HTTPS, JMS, RMI/IIOP
User Experience
Enterprise (Management, Service Composition & Transactions)
QoS (Security & Reliability)
Description & Discovery
Messaging & Encoding
Foundation

Figure 3-5 SOA standard stack with Web service

Also, we can view SOA technologies by categories. Figure 3-6 on page 31 shows the category of the standards, include: core technologies, description and discovery, messaging, management, business process transactions, security, user experience, J2EE and Java JSR.



Category of Standard

Category	Standard	Jump
Core	▸ SOAP	Click
	▸ WSDL	Click
	▸ UDDI	Click
	▸ XML	Click
Description and discovery	▸ WS-Inspection (also called WSIL)	
	▸ WS-Discovery	
	▸ WS-MetadataExchange	
	▸ WS-Policy	
Messaging	▸ ASAP	
	▸ SOAP Messages with Attachments (SwA)	
	▸ SOAP MTOM	
	▸ WS-Addressing	
	▸ WS-Notification (consisting of)	
	WS-BaseNotification	
	WS-BrokeredNotification	
	WS-Topics	
	▸ WS-Eventing	
	▸ WS-Enumeration	
	▸ WS-MessageDelivery	
	▸ WS-Reliability	
	▸ WS Reliable Messaging	
	▸ WS-Resources (consisting of)	
	WS-ResourceProperties	
	WS-ResourceLifetime	
	WS-BaseFaults	
	WS-ServiceGroup	
	▸ WS-Transfer	
Management	▸ WSDM	
	▸ WS-Manageability	
	▸ SPML	
	▸ WS-Provisioning	
Business processes	▸ BPEL4WS	
	▸ WS-CDL	
	▸ WS-CAF	

Category	Standard	Jump
Transactions	▸ WS-Coordination	Click
	▸ WS-Transaction	Click
	▸ WS-AtomicTransaction	Click
	▸ WS-BusinessActivity	Click
Security	▸ XML-Encryption	Click
	▸ XML-Signature	Click
	▸ WS-Security	Click
	▸ WS-SecureConversation	Click
	▸ WS-SecurityPolicy	Click
	▸ WS-Trust	Click
	▸ WS-Federation	Click
	▸ SAML	Click
User experience	▸ WSRP	Click
J2EE and Java JSR	▸ JSR 101: JAX-RPC	Click
	▸ JSR 109: Implementing Enterprise Web Services	Click
	▸ JSR 31: JAXB	Click
	▸ JSR 67: JAXM	Click
	▸ JSR 93: JAXR	Click
	▸ JSR 110: WSDL4J	Click
	▸ JSR 172: J2ME Web Services	Click
	▸ JSR 173: StAX	Click
	▸ JSR 181: Web Services Metadata for Java	Click
	▸ JSR 208: JBI	Click
	▸ JSR 222: JAXB 2.0	Click
	▸ JSR 224: JAX-RPC 2.0	Click
	▸ JSR 921: Implementing Enterprise Web Services 1.1	Click

Figure 3-6 Category of the standard

Figure 3-7 on page 32 shows the relationship between the core elements of the SOA. All elements use XML including XML namespaces and XML Schemas. Service requestor and provider communicate with each other. WSDL is one of technologies to make service interfaces and implementations available in the UDDI registry. WSDL is also the base for SOAP server deployment and SOAP client generation.

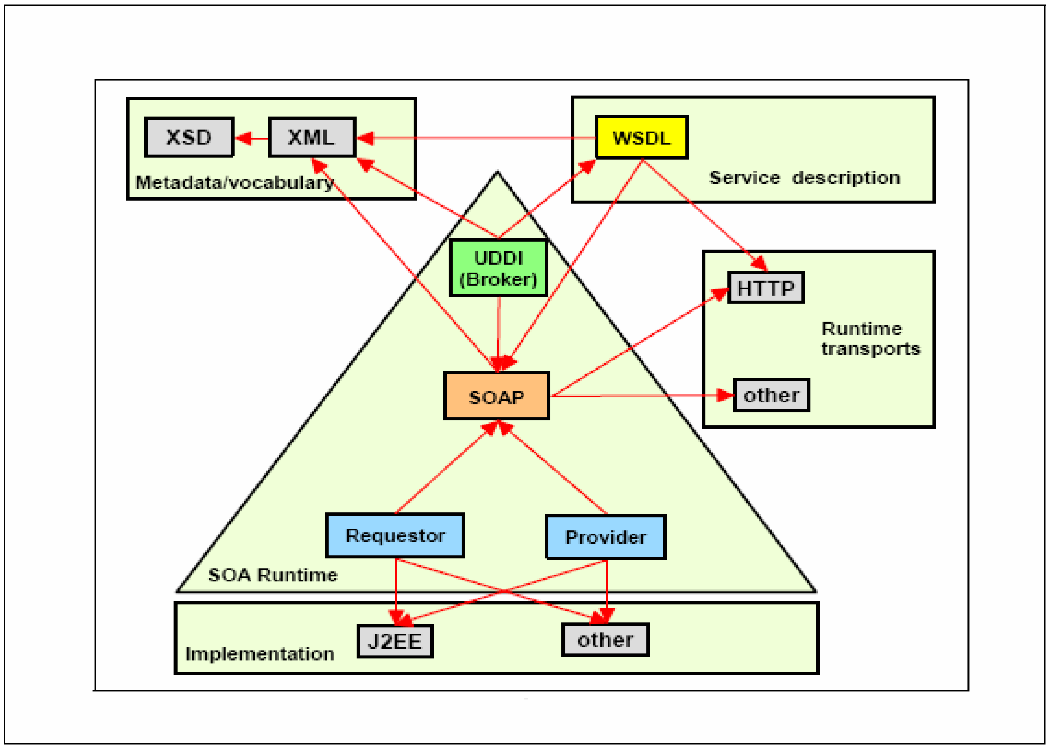


Figure 3-7 Core technologies for SOA

IBM has been a leader in SOA space. For SOA life cycle development, IBM has a set of tools and methods to develop, deploy and manage SOA applications. The IBM SOA Reference Architecture defines the IT services required to support an SOA. It includes development environment, services management, application integration, and runtime process services. The capabilities of the architecture can be implemented on a build-as-you-go basis as new requirements are addressed over time.

Figure 3-8 on page 33 shows the IBM SOA reference architecture and the supporting software.

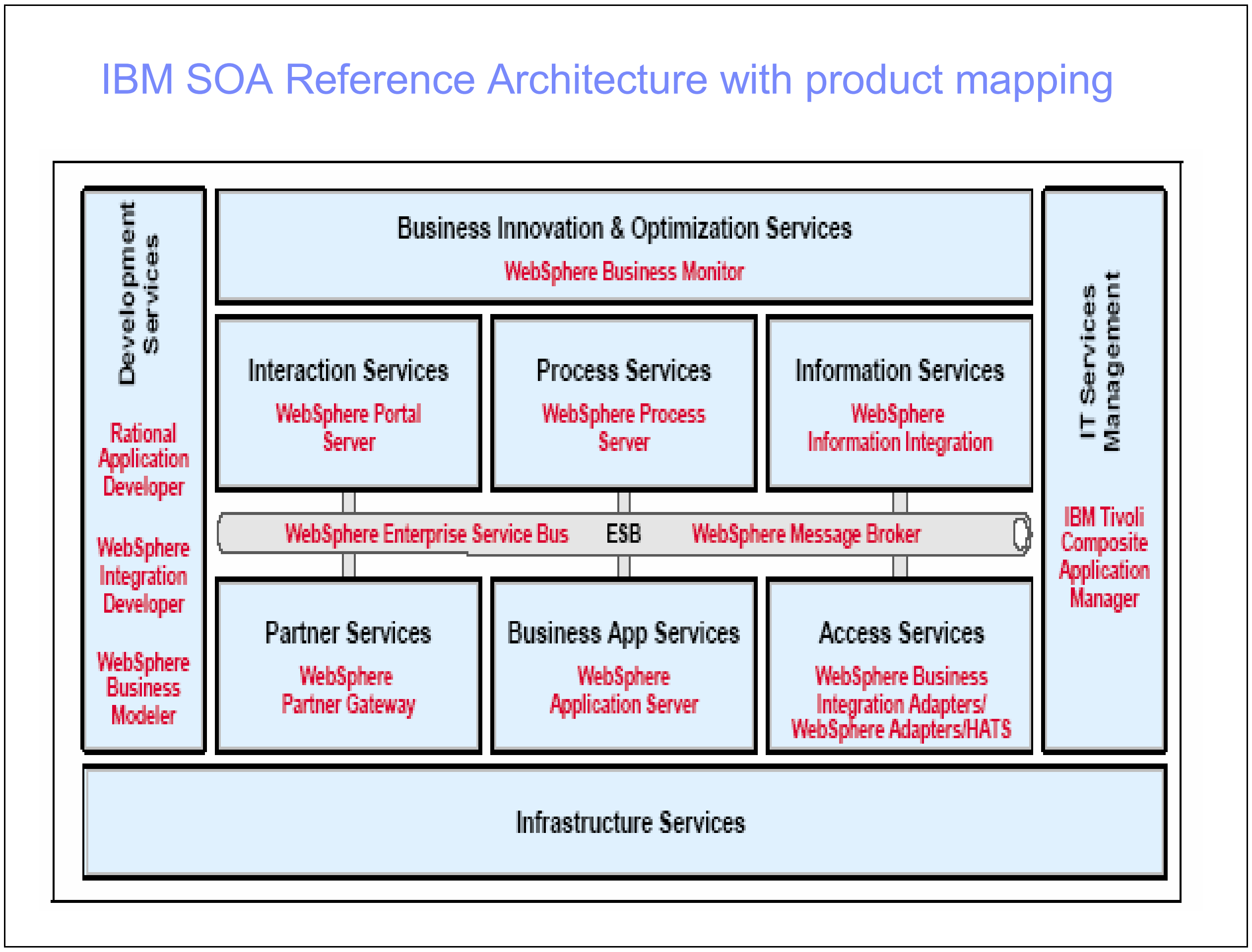

IBM SOA Reference Architecture with product mapping
Development Services
Rational Application Developer
WebSphere Integration Developer
WebSphere Business Modeler
Business Innovation & Optimization Services
WebSphere Business Monitor
Interaction Services
WebSphere Portal Server
Process Services
WebSphere Process Server
Information Services
WebSphere Information Integration
WebSphere Enterprise Service Bus
ESB
WebSphere Message Broker
Partner Services
WebSphere Partner Gateway
Business App Services
WebSphere Application Server
Access Services
WebSphere Business Integration Adapters/ WebSphere Adapters/HATS
IT Services Management
IBM Tivoli Composite Application Manager
Infrastructure Services

Figure 3-8 IBM SOA Reference Architecture with product mapping

3.2.2 eXtensible Markup Language

Extensible Markup Language (XML) and Extensible Stylesheet Language (XSL) stylesheets can be used on the server side to encode content streams and parse them for different clients, enabling you to develop applications for both a range of PC browsers and for emerging pervasive devices. The content is in XML and an XML parser is used to transform it to output streams based on XSL stylesheets that use Cascading Style Sheets (CSS).

This general capability is known as transcoding and is not limited to XML-based technology. The appropriate design decision here is how much control over the content transforms you need in your application. You will want to consider when it is appropriate to use this dynamic content generation and when there are advantages to having servlets or JavaServer™ Pages™ (JSPs) specific to certain device types.

XML is also used as a means to specify the content of messages between servers, whether the two servers are within an enterprise or represent a business-to-business connection. The critical factor here is the agreement between parties on the message schema, which is specified as an XML Document Type Definition (DTD) or Schema. An XML parser is used to extract specific content from the message stream. Your design will need to consider whether to use an event-based approach, for which the Simple API for XML (SAX API) is appropriate, or to navigate the tree structure of the document using the Document Object Model (DOM) API.

IBM's XML4J XML parser was made available through the Apache open source organization under the Xerces name. For open source XML frameworks, see:

http://xml.apache.org/

Defining XML documents

XML documents are defined using DTDs or XML Schemas. DTDs are a basic XML definition language, inherited from the Standard Generalized Markup Language (SGML) specification. The DTD specifies what markup tags can be used in the document along with their structure. DTDs have two major problems:

- Poor data typing

  In DTD elements can only be specified as EMPTY, ANY, element content, or mixed element-and-text content, and there is no standard way to specify null values for elements. Data typing such as date formats, numbers, or other common data types cannot be specified in the DTD. As a result, an XML document might comply with the DTD but still have data type errors that can only be detected by the application.

- Not defined in XML

  DTD uses its own language to define XML syntax that is not compliant to the XML specification. This makes it difficult to manipulate a DTD.

To solve these problems, the World Wide Web Consortium (W3C) defined a new standard to define XML documents called XML Schema. XML Schema provides the following advantages over DTDs:

- Strong typing for elements and attributes
- Standardized way to represent null values for elements
- Key mechanism that is directly analogous to relational database foreign keys
- Defined as XML documents, making them programmatically accessible

Even though XML Schema is a more powerful technology to define XML documents, it is also a lot harder to work with, so DTDs are still widely used to define XML documents. Additionally, simple documents can be easily defined using DTDs with similar results to using XML Schema.

Whether you use one or the other will depend on the complexity of the messages and the validation requirements of the application. Actually, in many cases both DTD and XML Schema are provided, so they can be used by the application depending on its requirements.

XSLT

Extensible Stylesheet Language Transformations (XSLT) is a W3C specification for transforming XML documents into other XML documents. The XSLT is built on top of the Extensible Stylesheet Language (XSL), which is a stylesheet language for XML. Unlike CSS, XSL is also a transformation language.

A transformation expressed in the XSLT language defines a set of rules for transforming a source tree to a result tree, and it is expressed in the form of a stylesheet. An XSLT processor is used for transforming a source document to a result document. There are currently a number of XSLT processors available on the market. DataPower has introduced an XSL just-in-time (JIT) compiler, which speeds up the time taken for the XSL transformation.

The XSLT processor has a performance overhead, so online processing of larger documents can be slow.

XML security

XML security is an important issue, particularly where XML is being used to by organizations to interchange data across the Internet. Several new XML security specifications are working their way through three standards bodies:

- World Wide Web Consortium (W3C)
- Internet Engineering Task Force (IETF)
- Organization for the Advancement of Structured Information Standards (OASIS)

We highlight a few specifications here:

- **XML Signature Syntax and Processing** is a specification for digitally signing electronic documents using XML syntax.

  A key feature of the protocol is the ability to sign parts of an XML document rather than the document in its entirety. This is necessary because an XML document might contain elements that will change as the document is passed along or various elements that will be signed by different parties.

  IBM WebSphere Studio provides you with the ability to create and verify XML digital signatures using a wizard.

- **XML encryption** will allow encryption of digital content, such as Graphical Interchange Format (GIF) images or XML fragments. XML Encryption allows parts of an XML document to be encrypted while leaving other parts open, encryption of the XML itself, or the super-encryption of data (that is, encrypting an XML document when some elements have already been encrypted).
- **XML Key Management Specification (XKMS)** establishes a standard for XML-based applications to use Public Key Infrastructure (PKI) when handling digitally signed or encrypted XML documents. XML signature addresses message and user integrity, but not issues of trust that key cryptography ensures.
- **Security Assertion Markup Language (SAML)** is the first industry standard for secure e-commerce transactions using XML. It aims to standardize the exchange of user identities and authorizations by defining how this information is to be presented in XML documents, regardless of the underlying security systems in place

Advantages of XML

There are many advantages of XML in a broad range of areas. Some of the factors that influenced the wide acceptance of XML are:

- Acceptability of use for data transfer

  XML is a standard way of putting information in a format that can be processed and exchanged across different hardware devices, operating systems, software applications, and the Web.

- Uniformity and conformity

  XML gives you an common format that could be developed upon and is accepted industry-wide.

- Simplicity and openness

  Information coded in XML is human readable.

- Separation of data and display

  The representation of the data is separated from the presentation and formatting of the data for display in a browser or other device.

- Industry acceptance

  XML has been accepted widely by the information technology and computing industry. Numerous tools and utilities are available, along with new products for parsing and transforming XML data to other data, or for display.

Disadvantages of XML

Some XML issues to consider are:

- Complexity

  While XML tags can allow software to recognize meaningful content within documents, this is only useful to the extent that the software reading the document knows what the tagged content means in human terms, and knows what to do with it.

- Standardization

  When multiple applications use XML to communicate with each other they need to agree on the tag names they are using. While industry-specific standard tag definitions often do exist, you can still declare your own nonstandard tags.

- Large size

  XML documents tend to be larger in size than other forms of data representation.

Example 3-1 is a XML and WSDL example for stock quote. It includes trade price, ticker symbol, price and so on.

Example 3-1 XML and WSDL

```
<?xml version="1.0"?>
<wsdl:definitions name="StockQuote" xmlns:wsdl="http://www.w3.org/2006/01/wsdl"
  targetNamespace="http://example.com/stockquote"
  xmlns:tns="http://example.com/stockquote"
  xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap">

  <wsdl:types>
    <xs:schema targetNamespace="http://example.com/stockquote"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="TradePriceRequest">
        <xs:complexType>
          <xs:all>
            <xs:element name="tickerSymbol" type="xs:string"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
      <xs:element name="TradePrice">
        <xs:complexType>
          <xs:all>
            <xs:element name="price" type="xs:float"/>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>

  <wsdl:interface name="StockQuoteInterface">
    <wsdl:operation name="GetLastTradePrice"
      pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <wsdl:input element="tns:GetLastTradePriceInput"/>
      <wsdl:output element="tns:GetLastTradePriceOutput"/>
    </wsdl:operation>
  </wsdl:interface>

  <wsdl:binding name="StockQuoteSoapBinding" interface="tns:StockQuoteInterface"
    type="http://www.w3.org/2006/01/wsdl/soap"
                                    wsoap:version="1.1"

wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP">
    <wsdl:operation ref="tns:GetLastTradePrice"
      wsoap:action="http://example.com/GetLastTradePrice"/>
  </wsdl:binding>

  <wsdl:service name="StockQuoteService" interface="tns:StockQuoteInterface">
    <wsdl:documentation>My first service</wsdl:documentation>
    <wsdl:endpoint name="StockQuoteEndPoint" binding="tns:StockQuoteBinding"
    address="http://example.com/endpoint/stockquote"/>
  </wsdl:service>

</wsdl:definitions>
```

3.2.3 SOAP and WSDL

Simple Object Access Protocol (SOAP) is an-XML based format for constructing messages in a transport independent way and a standard on how the message should be handled. SOAP messages consist of an envelope containing a header and a body. It also defines a mechanism for indicating and communicating problems that occurred while processing the message. These are known as SOAP faults.

The headers section of a SOAP message is extensible and can contain many different headers defined by different schemas. The extra headers can be used to modify the behavior of the middleware infrastructure. For example, the headers can include information about transactions that can be used to ensure that actions performed by the service consumer and service provider are coordinated.

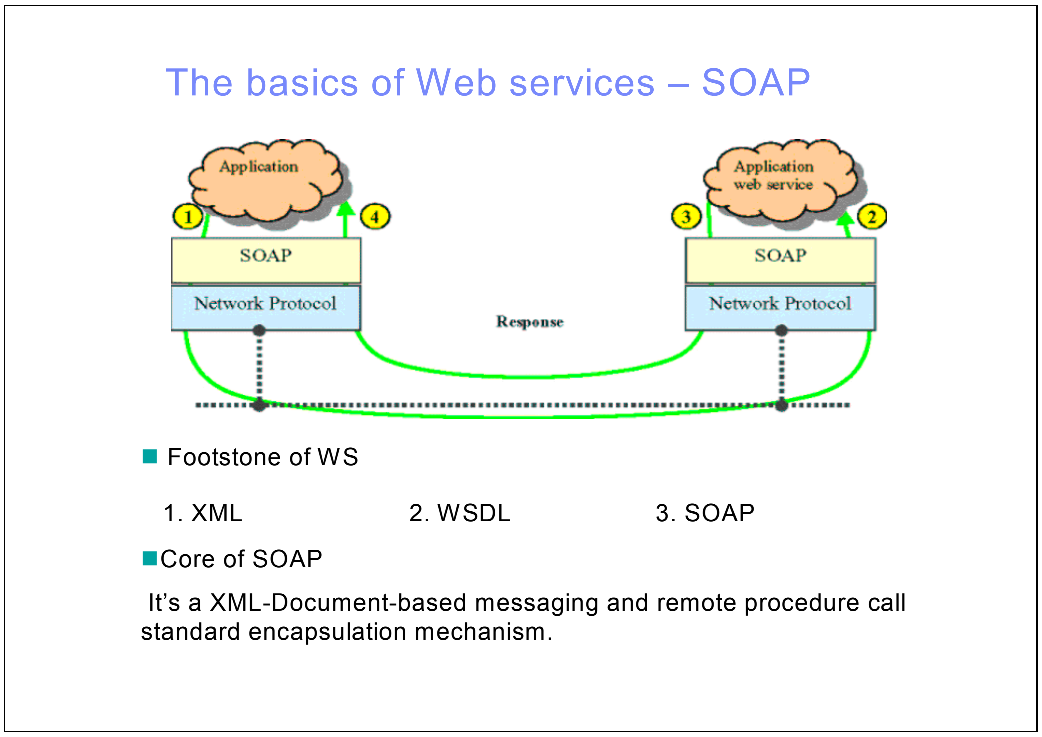
The body section contains the content of the SOAP message. When used by Web services, the SOAP body contains XML-formatted data. This data is specified in the WSDL describing the Web service.

When talking about SOAP, it is common to talk about SOAP in combination with the transport protocol used to communicate the SOAP message. For example, SOAP transported using HTTP is referred to as SOAP over HTTP or SOAP/HTTP.

The most common transport used to communicate SOAP messages is HTTP. This is to be expected because Web services are designed to make use of Web technologies. However, SOAP can also be communicated using JMS as a transport. When using JMS the address of the Web service is expressed in terms of a JMS connection factory and a JMS destination. Although using JMS provides a more reliable transport mechanism it is not an open standard, it requires extra and potential expensive investment and does not interoperate as easily as SOAP over HTTP.

The SOAP version 1.1 and 1.2 specifications are available from the World Wide Web Consortium.

Figure 3-9 on page 38 shows the conceptual model for SOAP and Web service.



The basics of Web services – SOAP

- Footstone of WS

  1. XML 2. WSDL 3. SOAP

- Core of SOAP

  It's a XML-Document-based messaging and remote procedure call standard encapsulation mechanism.

Figure 3-9 Conceptual SOAP model

Web Services Description Language (WSDL) is an XML-based interface definition language that separates function from implementation, and enables design by contract as recommended by SOA. WSDL descriptions contain a port type (the functional and data description of the operations that are available in a Web service), a binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP over HTTP), and a port (providing a specific address through which a Web service can be invoked using a specific protocol binding). It is common for these three aspects to be defined in three separate WSDL files, each importing the others.

WSDL describes Web services as a collection of communication endpoints, called ports. Messages also describe the data being exchanged. Operation is every action allowed at an endpoint. Port types are collections of operations possible on an endpoint.

The protocol and data format specifications for a port type are specified as a binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. In addition, WSDL specifies a common binding mechanism to bring together all protocol and data formats with an abstract message, operation, or endpoint. See Figure 3-7 on page 32.

The code samples in Example 3-2 and Example 3-3 on page 39 show how WSDL works with SOAP.

Example 3-2 A SOAP request

```
A SOAP Request

POST /EndorsementSearch HTTP/1.1
Host: www.snowboard-info.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 261
```

```
SOAPAction: "http://www.snowboard-info.com/EndorsementSearch"
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
      <manufacturer>K2</manufacturer>
      <model>Fatbob</model>
    </m:GetEndorsingBoarder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In response to the SOAP request of Example 3-2 on page 38, the server can send the SOAP response (HTTP header) message for the foregoing request as shown in Example 3-3. In natural language, it encapsulates the simple string response "Chris Englesmann".

Example 3-3 A SOAP response

```
A SOAP Response

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetEndorsingBoarderResponse xmlns:m="http://namespaces.snowboard-info.com">
      <endorsingBoarder>Chris Englesmann</endorsingBoarder>
    </m:GetEndorsingBoarderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The value of WSDL is that it enables development tooling and middleware for any platform and language to understand service operations and invocation mechanisms. For example, given the WSDL interface to a service that is implemented in Java, running in a WebSphere environment, and offering invocation through HTTP, a developer working in the Microsoft .Net platform can import the WSDL and easily generate application code to invoke the service. As with SOAP, the WSDL specification is extensible and provides for additional aspects of service interactions to be specified, such as security and transactions.

3.2.4 Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) servers act as a directory of available services and service providers. SOAP can be used to query UDDI to find the locations of WSDL definitions of services, or the search can be performed through a user interface at design or development time. The original UDDI classification was based on a U.S. government taxonomy of businesses, and recent versions of the UDDI specification have added support for custom taxonomies. Figure 3-4 on page 28 shows the famous triangle for Web services model. There are three elements: provider, register and requestor. A Web service provider registers its services with the UDDI registry. A Web service requestor looks up required services in the UDDI registry and, when it finds a service, the requestor binds directly with the provider to use the service.

The UDDI specification defines an XML schema for SOAP messages and APIs for applications wanting to use the registry. A provider registering a Web service with UDDI must furnish business, service, binding, and technical information about the service. The UDDI specification includes two categories of APIs for accessing UDDI services from applications:

- Inquiry APIs - enable lookup and browsing of registry information
- Publishers APIs - allow applications to register services with the registry

UDDI APIs behave in a synchronous manner. In addition, to ensure that a Web service provider or requestor can use the registry, UDDI uses SOAP as the base protocol. Note that UDDI is a specification for a registry, not a repository. As a registry it functions like a catalog, allowing requestors to find available services. A registry is not a repository because it does not contain the services itself.

A public UDDI directory is provided by IBM, Microsoft, and SAP, each of whom runs a mirror of the same directory of public services. However, there are many patterns of use that involve private registries. For more information, see the following articles:

- “The role of private UDDI nodes in Web services, Part 1: Six species of UDDI”, available at:

  `http://www.ibm.com/developerworks/webservices/library/ws-rpu1.html`
- “The role of private UDDI nodes, Part 2: Private nodes and operator nodes”, available at:

  `http://www.ibm.com/developerworks/webservices/library/ws-rpu2.html`

3.3 An overview of SOA and Web service

Service-oriented architecture is an approach to defining integration architectures based on the concept of a service. It applies successful concepts proved by Object Oriented development, Component Based Design, and Enterprise Application Integration technology. The goal of SOA can be described as bringing the benefits of loose coupling and encapsulation to integration at an enterprise level.

In order to describe SOA, it is first necessary to define what we understand by a “service” in this context. This is key as, unless we are confident that the services that we define really are well designed, we cannot be sure to achieve the promoted benefits of SOA. The most commonly agreed-on aspects of the definition of a service in SOA are:

  – Services are defined by explicit, implementation-independent interfaces.
  – Services are loosely bound and invoked through communication protocols that stress location transparency and interoperability.
  – Services encapsulate reusable business function.

The use of explicit interfaces to define and encapsulate services function is of particular importance. The interface encapsulates those aspects of process and behavior that are common to an interaction between two systems, while hiding the specifics of each implementation. By explicitly defining the interaction in this way, those aspects of either system (for example the platform they are based on) that are not part of the interaction are free to change without affecting the other system. Figure 3-10 on page 41 shows an example about IBM WebSphere service registry and repository service life cycle.

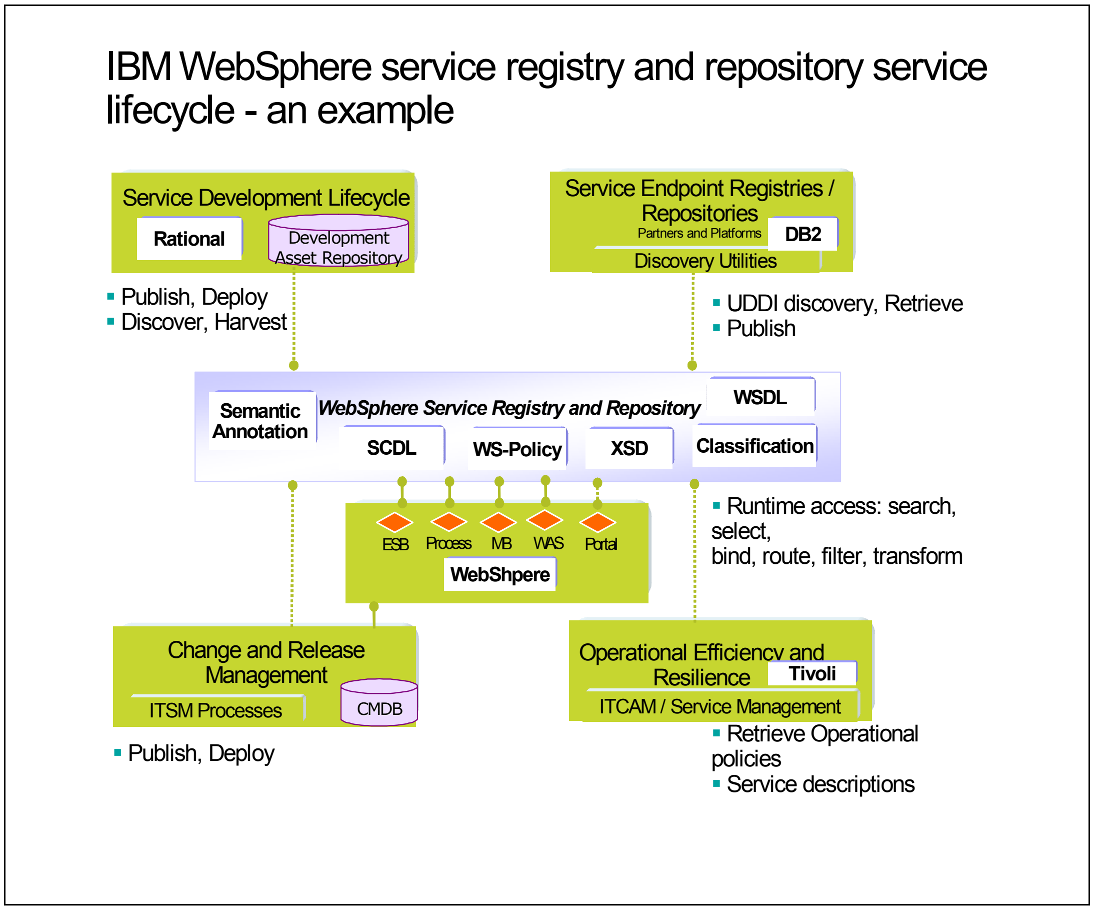

IBM WebSphere service registry and repository service lifecycle - an example
Service Development Lifecycle
Rational
Development Asset Repository
Publish, Deploy
Discover, Harvest
Service Endpoint Registries / Repositories
Partners and Platforms
DB2
Discovery Utilities
UDDI discovery, Retrieve
Publish
Semantic Annotation
WebSphere Service Registry and Repository
WSDL
SCDL
WS-Policy
XSD
Classification
ESB
Process
MB
WAS
Portal
WebShpere
Runtime access: search, select, bind, route, filter, transform
Change and Release Management
ITSM Processes
CMDB
Publish, Deploy
Operational Efficiency and Resilience
Tivoli
ITCAM / Service Management
Retrieve Operational policies
Service descriptions

Figure 3-10 IBM WebSphere service registry and repository service life cycle - an example

After the function has been encapsulated and defined as a service in an SOA, it can be used and reused by one or more systems that participate in the architecture. For example, when the reuse of a Java logging API could be described as "design time" (when a decision is made to reuse an available package and bind it into application code), the intention of SOA is to achieve the reuse of services at:

- Runtime: Each service is deployed in one place and one place only, and is remotely invoked by anything that must use it. The advantage of this approach is that changes to the service (for example, to the calculation algorithm or the reference data it depends on) need only be applied in a single place.
- Deployment time: Each service is built once but redeployed locally to each system or set of systems that must use it. The advantage of this approach is increased flexibility to achieve performance targets or to customize the service.

Note that in contrast to reusing service implementations at runtime, the encapsulation of functions as services and their definition using interfaces also enables the substitution of one service implementation for another. For example, the same service might be provided by multiple providers (such as a car insurance quote service, which might be provided by multiple insurance companies), and individual service requesters might be routed to individual service providers through some intermediary agent.

The encapsulation of services by interfaces and their invocation through location-transparent, interoperable protocols are the basic means by which SOA enables increased flexibility and reusability. In order to really understand how these benefits can be achieved we delve a little further into the details of good service design by considering these topics:

- Coupling and decoupling of aspects of service interactions
- Designing connectionless services
- Service granularity and choreography
- Implications of service-oriented architecture
- Web services interoperability

3.3.1 Coupling and decoupling of aspects of service interactions

A basic tenet of SOA is that the use of explicit service interfaces and interoperable, location-transparent communication protocols means that services are loosely coupled with each other. To understand how this is implemented in practice, and how it enables the benefits of SOA, we explore the meaning of loose coupling in more detail.

By loosely coupling services, we mean restricting the number of things that the requester application code and the provider application code know about each other. If a change is made to any aspect of a service that is coupled, then either the requester or the provider application code (or, more likely, both) will have to change. If a change is made by any party (the requester, provider, or mediating infrastructure) to any aspect of a service that is decoupled, then there should be no need to make subsequent changes in the other parties.

Notice that we are no longer discussing loosely coupled services, but coupled and decoupled aspects of services. We can also ask whether coupled and decoupled are the only two relationships that can exist for an aspect of a service between the requester and the provider. For example, the business behavior (the function and data model) obviously must be coupled in order for the requester and provider to interact. In order to flexibly integrate systems in a heterogeneous environment, it is best that the requester and provider platforms (for example AIX or Windows) be decoupled.

However, in a realistic situation, the interactions between requester and provider must also be secured, and the relationship between their transactional models will have to be understood in order to define how failures will be handled. These and other characteristics fall somewhere between coupled and decoupled the sense that those terms are used here.

As a working framework, we define the following relationship styles for service aspects among requesters and providers:

- An aspect is coupled if changes to the aspect by one party in the interaction (requester, provider, or mediating infrastructure) require corresponding changes by the other parties.
- An aspect is declared if its behavior is specified in the interface to the service, and service requesters and providers can only interact if they have matching declared behavior, and this behavior is consistent with the capabilities of the intermediary infrastructure supporting the interaction.

  There are two variations of declared behavior that provide some additional levels of flexibility:

  – An aspect is transformed if it is declared by both service requesters and service providers, but the infrastructure provides some transformation capability to enable interactions between service requesters and providers that declare mismatched behavior.

  – An aspect is negotiated if both requester and provider declare a spectrum of behaviors that they are able to support, and if the intermediary infrastructure is capable of negotiating an agreed behavior between them for each interaction.

- An aspect is decoupled if changes to the aspect by one party in the interaction do not require corresponding changes by the other parties. In order to clarify these ideas, it is useful to consider an example of each type of coupling:
  - Business data models are usually coupled between service requesters and providers; the application code of each must understand the information that is required to describe (for example) a Customer, Account, or Order.
  - Communication protocols can be declared in the service interface. In practice, this requires that applications code to a protocol-independent service API, such as a suitable implementation of JAX-RPC. If this is the case, then the protocol binding in the service interface definition can be changed (for example, from SOAP/HTTP to SOAP/JMS). This does not require changes to the application code, but it affects the behavior of the service API implementation, which will execute the service interaction through a different protocol.
  - Data formats are often transformed: It is very common, that is, to convert former or existing formats, such as COBOL copybooks, to XML formats when enabling service interfaces to earlier systems. Alternatively, different XML schema may be used by different systems in an SOA to describe the same data models. In either case, the supported format is, or can be, defined in a service interface, and middleware transformation capabilities can be used in the service infrastructure to perform the required transformations without affecting application code or behavior.
  - The identity of a service provider might be negotiated through a third-party broker component. The broker might use geographical location, client identify, membership scheme information, transaction value, or several other criteria to match the service requester with a suitable service provider.
  - The implementation platform is often decoupled; if two systems interact through interoperable protocols such as SOAP/HTTP or messaging middleware, then neither is aware in any way of the hardware, operating system, or perhaps even the application server platform supporting the other; either party can change any or all of these aspects without affecting the other.

We can apply these relationships to various aspects of service that can be identified in a SOA. For some aspects, SOA or other design principles specify the desired style of relationship; for other aspects, several relationships might be appropriate depending on specific scenarios. For each aspect, different techniques can be applied to implement the desired relationship. Table 3-1 identifies some service aspects, relationships, and techniques. It should be noted, however, that this area is the subject of ongoing debate and evolution, and the table should not be taken as definitive. Similarly, the available technologies are evolving rapidly, for example, the emerging WS-Policy specification will affect this area of design deeply in coming years.

Table 3-1 Service aspects, relationships and implementation techniques in SOA

Aspect	SOA intention	SOA intention techniques
Semantic interface	Coupled	► Business systems must share an understanding of the tasks and data that are processed by the service. ► Shared business object libraries or XML schemas can be exploited. ► Some transformations, aggregations or enrichments of data might be consistent with the interface semantics and implemented by the bus infrastructure; more likely such transformations would be related to service granularity and choreography.

Aspect	SOA intention	SOA intention techniques
Language and platform	Decoupled	► Language and platform-independent interface definition such as IDL, WSDL, XSD. ► Language and platform-independent data formats such as XML. ► Language and platform-independent communication protocols such as IIOP, SOAP, WebSphere MQ. ► Invocation APIs (for example, JAX-RPC), adapters, or ESB infrastructure to integrate applications to the interface definitions and data formats.
Data format	Declared or Transformed	► Language and platform-independent data formats such as XML. ► Adapters, XSL style sheets, or bus infrastructure required to support transformations between data formats, such as between COBOL copybooks and XML. ► Application development tool wizards can create language-specific representations of some data formats, particularly XML. ► Other aspects of data format that are critical to real-world SOA implementations are data encoding, code pages, and data compression, including XML compression techniques.
Protocol	Declared or Transformed	► Service invocation mechanisms for service requesters and providers that do not specify service protocol or locations; for example, an implementation of JAX-RPC with support for multiple protocols.
Service provider identity or implementa-tion	Declared or Negotiated	► Service invocation mechanisms that enable service substitution, for example JAX-RPC. ► Adapters or ESB infrastructure can perform service routing to different providers. ► Directory (for example, UDDI) or broker intermediary to decide who fulfills the service each time. ► An ESB might identify a suitable service provider based on WS-Policy, for example by selecting the cheapest or most-responsive provider available at the time.
Time	Declared or Negotiated	► As IT systems show many differing planned and unplanned availability characteristics (such as 24/7 versus working hours), service interactions will sometimes have to span systems with different characteristics. ► Declared by WSDL or negotiated through WS-Policy. ► Use of asynchronous transport protocols, for example WebSphere MQ, WS-ReliableMessaging. ► ESB or intermediary store and forward capability for asynchronous request /response, message correlation, and so forth. ► Message correlation and transaction identifiers used to associate individual service interactions with longer ongoing business process interactions.

Aspect	SOA intention	SOA intention techniques
Delivery assurance, integrity, and error handling	Declared or Negotiated	► Assured delivery communication protocols; for example WebSphere MQ, WS-ReliableMessaging. ► Error and exception handling processes, for example for SOAP faults. ► Use the features and deployment descriptors of containers, such as J2EE, in service implementations. ► Advanced WS standards, for example WS-ReliableMessaging and WS-Transaction. ► Negotiated through WS-Policy by the ESB. ► In order to provide a consistent end-to-end approach to delivery assurance, integrity, and error handling for a chain of service interactions, it will often be necessary to combine several techniques that are used for individual interactions. These techniques might include handling communication failures, the use of synchronous two-phase commit, the ability to handle duplicate messages, and compensation schemes.
Security	Declared or Negotiated	► Declared by WS-Security or negotiated through WS-Policy. ► Point-to-point or communication-based security and trust models. ► Implemented through applications or through third-party or intermediary components in the SOA architecture.
Service version	Declared or Negotiated	► Service naming standards. ► Version-based routing in the bus infrastructure. ► Service request / provider tolerance of changes in optional data attributes.
Interaction state	Declared	► Matching of messages or events to long-lived processes by explicit process or transaction IDs in semantic interface, or by application data (for example, customer ID). ► Service Choreography technology may provide some facility to use a variety of input data to associate messages with specific instances of processes. ► Primary key matching technology such as provided by WebSphere InterChange Server. ► The emerging WS-ResourceFramework provides a standard model for associating services with stateful resources. ► Enterprise Application Integration middleware support for message aggregation and correlation. ► Customized solutions involving custom message headers.

It is interesting to note in the table that the only aspect of the service that is specified as coupled is the business behavior. By specifying other aspects to be declared, transformed, negotiated, or decoupled, the intention is to build the maximum possible flexibility into the architecture, enabling other aspects of service implementation and interaction to vary as freely as possible.

In combination with the flexibility of business behavior that is achieved by encapsulating well-designed business function as services, SOA attempts to maximize the overall flexibility of integrated business systems. The next two sections discuss some aspects of what is meant by encapsulating well-designed business function as services, in order to ensure that the flexibility of behavior really is achieved.

3.3.2 Designing connectionless services

The question of whether services should be stateful or stateless has been discussed frequently in relation to SOA. However, the issue is complicated by whether it really is possible to draw a clear line between state and business data. Many service interactions must be stateful in order to play a role in ongoing business processes or interactions; the issue is how we should design such services so as to maximize the flexibility of the architecture and the processes it supports.

To answer this issue, we return to the key to SOA: defining behavior in the interface. In doing so, we do away with considering stateful and stateless services, and instead declare that services, whether they deal with stateful business behavior (for example, the renewal process for an insurance policy) or stateless business behavior (such as performing an exchange rate calculation) should be connectionless. Connectionless services are those that do not allow or require a service requester and a specific, executable instance of the service provider to maintain a relationship between service invocations. The successful implementation of connectionless services depends on two considerations:

1. The use of technology that prevents handles being retained to specific executable instances.
2. The design of service interfaces that do not depend on implicit, shared knowledge created through a sequence of interactions between a specific requester and provider.

The first consideration is relatively easy to address: When stateless protocols such as asynchronous messaging are used to invoke services, this criterion is fulfilled. When technologies that are capable of supporting stateful behavior are used, the features of the technology that manipulate state (for example, HTTP Session or cookies) should not be used. Meeting this criterion might imply the use or assessment of specific communication technologies, or the application of design and development guidelines to the implementation of systems that participate in the SOA.

The second criterion can only be fulfilled through its application as a design principle to the design of the individual service interface. Table 3-2 shows an example of both a connected and connectionless design for two services. In this example, a store system in a consumer electronics shop is trying to charge the cost of an expensive television to a card account that belongs to Bruce; the cost is high enough that the store system must explicitly authorize the transaction with the card supplier.

Table 3-2 Connected and connectionless service interactions

Connectionless	**Connected**
Service Client: Can Bill pay $1000 for a new television? Service Provider: Yes Service Client: Charge Bill $1000 for a new television. Service Provider: OK	
All of the business data is defined in the interface.	Part of the business data (the fact that we are dealing with Bill) is implied in the sequence.

The connectionless example in the table shows that the interface to each call specifies all of the data that is required to perform the service, other than business information owned by the service provider. For example, Bill's card balance and credit limit are not part of the service interface because they are business information owned by the card provider. However, the fact that Bill is the owner of the account that is relevant to this specific transaction is a part of the interface, because both the service to authorize the payment and the service to make the payment must relate to the same cardholder. If this information were not part of the service interface (as in the Connected example), then the specific, executable instances of the service client and the service provider would have to maintain a reference to each other in order to identify the correct context.

This could cause difficulty if, for example, the physical server that supports the instance of the service provider crashed; in such as case, what happens to the instance of the service that remembers that it was dealing with Bill? Was the state of that instance safely stored somewhere prior to the failure, perhaps in a database? If so, how does the service requester then connect to another executable instance of the service that somehow knows which information to read from the database? All of these issues should be familiar to anyone who has developed stateful distributed applications, such as J2EE or WebSphere applications that make use of the Java HTTP Session object.

The principle that services should be designed to be connectionless is really saying that for a shared sequence of activity, each instance of that activity should be identified uniquely (for example, through a transaction ID or, in this case through a customer ID, Bill), and that the identity should form part of all service calls. Even better would be to explicitly define and share the process definition, as the emerging Business Process Execution Language for Web Services (BPEL4WS) standard could do.

3.3.3 Service granularity and choreography

Many descriptions of SOA also refer to the use of "large-grained" services. However, some powerful counterexamples of successful, reusable, fine-grained services exist. For example, getBalance is a very useful service, but hardly large-grained.

More realistically, there will be many useful levels of service granularity in most SOAs; for example:

- Technical functions (such as logging)
- Business functions (such as getBalance)
- Business transactions (such as openAccount)
- Business processes (such as applyForMortgage)

Some degree of choreography or aggregation is required between each granularity level. It is unlikely that all organizations will share identical definitions of granularity, but each will undoubtedly find it beneficial to define their own. At each level of granularity, it is important that service definitions encapsulate function well enough that it is reusable. Figure 3-11 on page 48 shows an example of service granularities and choreographies.

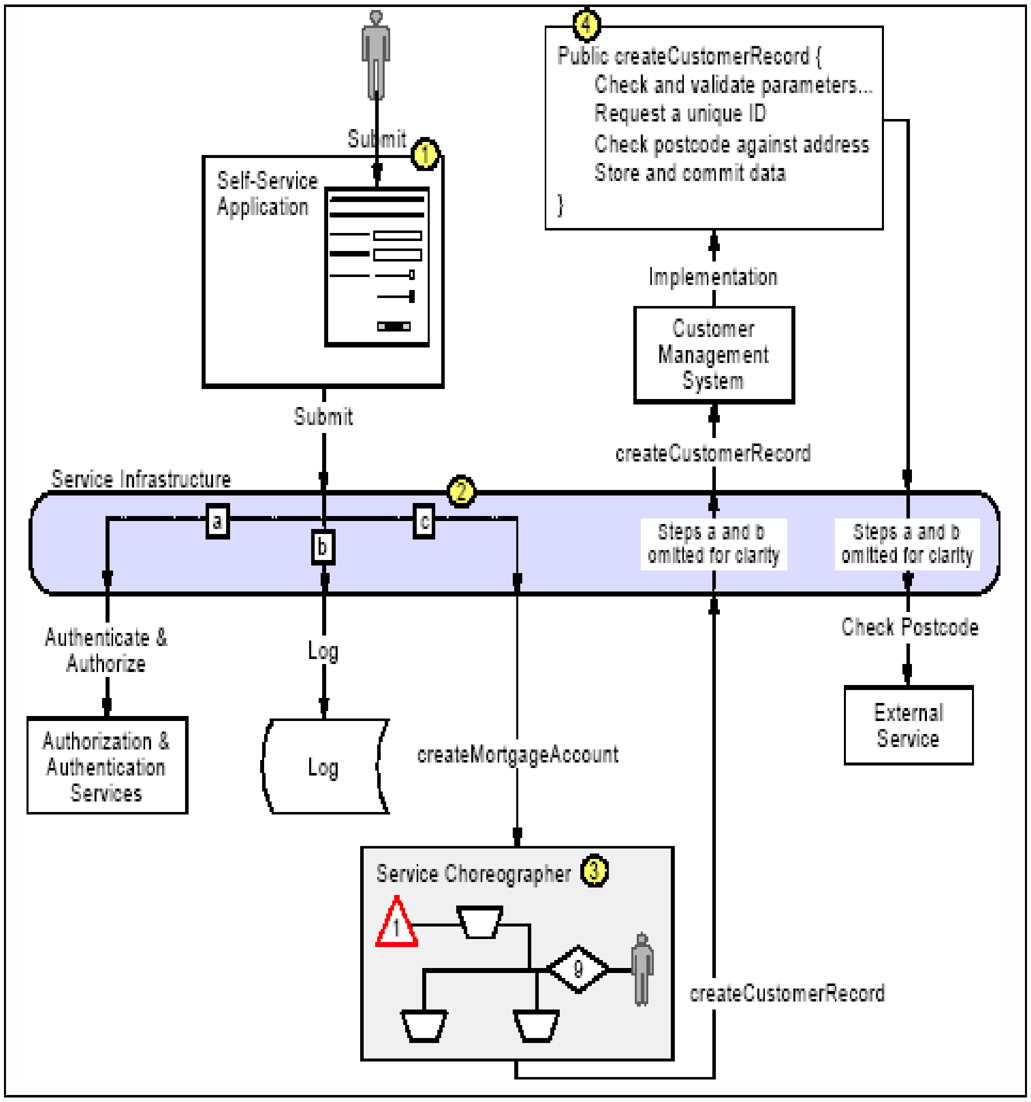


Figure 3-11 Service granularity and choreography

Figure 3-11 describes these interactions among services of various granularities:

- A user submits a request to a self-service application to create a mortgage account. The self-service application submits the business process service request *createMortgageAccount* through the service infrastructure to a service choreographer component, whose purpose is to choreograph business transaction services into business process services.
- On receiving the request for the *createMortgageAccount* business process service, the service infrastructure first invokes authentication and authorization technical function services to ensure that the request is valid, then a log technical function service before finally invoking the *createMortgageAccount* business process service in the service choreographer.
- The service choreographer executes the *createMortgageAccount* business process service. If the request is valid, then when the other process elements are complete the choreographer invokes the *createCustomerRecord* business transaction service through the service infrastructure to store the details of the new customer. (Before doing this, it may already have invoked s*toreMortgageDetails*.)

- In the implementation of the Customer Management System *createCustomerRecord* business transaction service, it is necessary to validate the information for the new customer. Part of this validation is checking whether the post code and address match. In order to do this, a *CheckPostCode* business function service is invoked through the service infrastructure.

To summarize, three aggregations or choreographies are performed by distinct components for distinct granularity levels:

- Service choreographer

  It choreographs business transaction services into higher level business process services.
- Service Infrastructure (may be an Enterprise Service Bus)

  It choreographs technical function services to control the invocation of business process services, business transaction services, and business function services.
- Individual application components

  It is responsible for invoking business function services where they are required in order to implement business transaction services.

Of course, this is just one hypothetical example. Real organizations must formulate their own definitions.

3.3.4 Implications of service-oriented architecture

The encapsulation of reusable business function, the achievement of loose coupling, the definition of appropriate levels of granularity, and so forth are analysis issues as much as a technology issues. They are difficult issues to grasp, so SOA cannot be successful without skilled architects and designers who understand and are able to articulate them. It is easy to see these concerns becoming hostage to time, skill, and cost issues, leading to another generation of isolated systems that will require integration.

Widespread implementation of an SOA and infrastructure is a long-term endeavour that involves all of the usual hard business decisions, questions of data, and process ownership. It requires serious, long-term commitment by business and by the IT organization that supports it. It may involve up front costs, centralized costs, and many other challenges:

- No specific technologies are ruled in or ruled out.
- Legacy implementations are possible (for example, CICS Transaction Server “super router” transactions with simplified, text-based interfaces)
- EAI implementations are commonplace (for example, XML over MQ WebSphere Business Integration Message Broker)
- Web services are potentially a very good fit, but are still maturing.

3.3.5 Web services interoperability

A unique feature of Web services is that it is a relatively high-level integration protocol with near-ubiquitous support in the IT industry; this alone is an important reason for its success and is behind why many individual projects have used the Web services standards to perform integrations between different platforms.

In order to facilitate the development of truly interoperable Web services standards from this widespread support, the Web Services Interoperability Organization (often referred to as the WS-I) was formed in February 2002. The WS-I aims to promote interoperability of Web services implementations by publishing profiles, which are descriptions of conventions and practices for the use of specific combinations of Web services standards through which systems can interact. Technology vendors can then produce compliant implementations and publicize that compliance, offering some level of assurance to technology customers as to the level of Web services interoperability that can be achieved with different implementations.

The WS-I published the first profile for interaction, the Basic Profile, in July 2003, and many technology vendors provide product implementations of Web services that are compliant with this profile. The WS-I is creating a Basic Security Profile to describe interoperability using the Web services security (WS-Security) standards. A draft specification of this profile was published in February 2004. Of course, interoperability can be achieved using Web services where WS-I profiles do not exist; however, it may be more limited or require additional work to achieve. Therefore, the WS-I is an important mechanism for assuring and simplifying interoperability between implementations of Web services standards as those standards mature and evolve.

The Web Services Interoperability Organization Web site contains links to published, draft, and planned interoperability profiles and information about vendor compliance:

`http://www.ws-i.org/`

3.4 Web services (WS) and SOA work together

The link between Web services and SOA is threefold:

- Web services provide an open standard and machine-readable model (WSDL) for creating explicit, implementation-independent descriptions of service interfaces.
- Web services provide communication mechanisms that are location-transparent and interoperable.
- Web services are evolving through Business Process Execution Language for Web Services (BPEL4WS), document-style SOAP, and WSDL, and emerging technologies such as WS-ResourceFramework to support the technical implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

Together, Web services and SOA have the potential to address the following technical issues that we may face:

- (WS) A multitude of technologies and platforms support your business systems.

  Web services are a set of open-standard technologies that are supported by most of the IT industry and by the Web Services Interoperability organization. Their basis in simple, text-based, and open-standard technologies such as XML and HTTP, and the fact that they can leverage more sophisticated interoperable technologies such as asynchronous messaging, means that they can be supported in the vast majority of IT environments. Increasing ubiquity and maturity of product support means that implementing and integrating Web services will become increasingly efficient.
- (SOA) Business process models are a mixture of people practices, application code, and interactions among people and systems or systems and systems. Although SOA is an approach to architecture that must be applied to systems and integrations, it specifies a set of principles and techniques that encourage the encapsulation and modeling of reusable business functions and processes. Recent and emerging trends in Web services, such as BPEL4WS and WS-Resource Framework, will increasingly support the modeling concepts of SOA.
- (SOA) Changes to one system tend to imply ripples of change at many levels to many other systems.SOA specifies several principles and techniques for achieving the encapsulation of service function and the loose coupling of service interactions. These

techniques minimize the cases where change to one part of a system implies changes to other parts to those cases where the implied changes are necessary to support the underlying changes to the way the system supports the business.

- (WS) No single, fully functional integration solution will talk to them all.

  At one level, the use of widely available and interoperable basic Web services open standards such as SOAP/HTTP with existing Internet and intranet infrastructure provide an integration solution that already has impressive reach, and it will become increasingly ubiquitous. Where increased manageability and qualities of service are required, emerging Enterprise Service Bus middleware, which combine Web services and SOA concepts with the power of traditional Enterprise Application Integration middleware technology, will provide a sophisticated and widely interoperable integration infrastructure.

- (WS) Deployment of any single, proprietary integration solution across the enterprise is complex, costly, and time consuming. Where basic Web services that utilize existing infrastructure are appropriate, deployment costs and efforts are minimal. The increasing availability of Web services support in Enterprise Application Integration middleware also enables the integration of different middleware infrastructures. Similarly, emerging Enterprise Service Bus technologies will interact with existing integration infrastructure rather than automatically replace it. So, although SOA and Web services cannot remove the cost and effort of deploying integration infrastructure, they offer several characteristics to minimize it.

- (WS) Assuming that you create the services, will your integration solution talk to your partners? Your future partners?

  The Web services technologies have proven effective in many B2B integrations, where their open standards basis and use of simple, existing infrastructure and protocols makes them particularly effective. Recent and emerging standards such as WS-Security add to the sophistication of interaction that is possible when using Web services in this model.

- (SOA) There is no single data, business, or process model across (or beyond) the enterprise.

  Although they are not a magic solution, the SOA principles define an approach that enables organizations to progressively expose functions across their business as services and to combine those services into process. Over time, businesses that take this approach will improve the consistency of their business and process models, and will leverage the use of business process modeling and automation technology to more explicitly control and monitor their execution of processes.

- (WS) Not all integration technologies work as well across a wide area network or the Internet as they do across a local area network.

  The Web services technologies support multiple protocols, so they can use the simplest protocols available, such as HTTP when that offers an advantage, or leverage other infrastructures such as WebSphere MQ when that is more appropriate.

We have discussed how the key features of SOA and Web services enable us to address those technical issues, and we have offered new opportunities for more flexible, rapid, and widespread integration, in a model that is consistent with the exposure of business function as services, and the choreography of those services into processes that can be modeled, executed, and monitored: SOA and WS. These techniques and technologies give companies the tools that are required to implement flexible SOAs and evolve toward an on demand business model. However, at the current time and for some time to come, the technologies will be evolving rather than mature and stable. Therefore, individual SOA solutions must make carefully balanced solutions among customized, proprietary, and open-standard technologies, which characteristics and components of SOA to implement, and which areas of business function and process to apply them to.

Of course, these decisions will be balanced between business benefits, technology maturity, and implementation or maintenance efforts.

3.5 SOA and Web service architecture design considerations

Previous sections in this chapter described various SOA and Web service components and motivations for Web services. They also described the various SOA technologies used for implementing Web services and showed how business might apply these technologies in an application. Where possible, the sections offered guidelines for good design and highlighted the advantages and disadvantages among the technologies.

In this sections, we illustrate how to apply these guidelines to the design and implementation of a Web service application, the adventure builder enterprise. When architecting and designing Web service applications, you are faced with the significant challenge of constructing the various application modules so that they work together smoothly. We explain the motivational factors and issues that need to be considered, and make these issues concrete by showing how we came to the decisions we eventually made as we architected. Through this examination, we hope to make it easier for you to determine how best to architect and design your own Web service applications.

Application design for e-business presents some unique challenges compared to traditional application design and development. The majority of these challenges are related to the fact that traditional applications were primarily used by a defined set of internal users, whereas e-business applications are used by a broad set of internal and external users such as employees, customers, and partners. Web applications must be developed to meet the varied needs of these end users.

3.5.1 e-business application design considerations

The following list provides key issues to consider when designing e-business applications:

- The user experience and the look and feel of the site need to be constantly enhanced to leverage emerging technologies and to attract and retain site users.
- New features have to be constantly added to the site to meet customer demands.
- Such changes and enhancements will have to be delivered at record speed to avoid losing customers to the competition.
- e-business applications in essence represent the corporate brand online. Developers have to work closely with the marketing department to ensure that the digital brand effectively represents the company image. Such intra-group interactions usually present content management challenges.
- It is hard to predict the runtime load of e-business applications. Based on the marketing of the site, the load can increase dramatically over time. If the load increases, the design must allow such applications to be deployed in various high-volume configurations. It is important to be able to move Web applications between these runtime configurations without making significant changes to the code.
- Security requirements are significantly higher for e-business applications compared to traditional applications. To execute traditional applications from the Web, a special set of security-related software might be needed to access private networks.
- The emergence of the personal digital assistant (PDA) and broadband Internet markets will require the same information to be presented in various UI formats. PDAs will require a lightweight presentation style to accommodate the low network bandwidth. Broadband users, on the other hand, will demand a highly interactive, rich, GUI.

To meet these challenges, it is critical to design Web applications to be flexible. This section helps you understand some of these design challenges and presents various design options that promote loosely coupled design to provide a maximum degree of flexibility in a Web application. We also provide application integration design guidelines and best practices for Web services, J2EE Connectors, and Java Message Service (JMS).

3.5.2 Design considerations for Web services

Web services are deployed on the Web by service providers. The functions provided by the Web service are described using the Web Services Description Language (WSDL). Deployed services are published on the Web by service providers.

A service broker helps service providers and service requestors find each other. A service requestor uses the Universal Discovery Description and Integration (UDDI) API to ask the service broker about the services it needs. When the service broker returns the search results, the service requestor can use those results to bind to a particular service.

As we can see in Figure 3-4 on page 28:

- Web service descriptions can be created and published by service providers.
- Web services can be categorized and searched by specific service brokers.
- Web services can be located and invoked by service requesters.

Building blocks for Web services

We can now look at the building blocks of Web services: SOAP, UDDI and WSDL.

The characteristics of SOAP

SOAP is a network-, transport-, and programming language-neutral protocol that allows a client to call a remote service. The message format is XML. The currently adopted standard is W3C's SOAP 1.1 specification, while SOAP 1.2 is in the review process.

SOAP has the following characteristics:

- SOAP is designed to be simple and extensible.
- All SOAP messages are encoded using XML.
- SOAP is transport protocol independent. HTTP is one of the supported transports. Hence, SOAP can be run over an existing Internet infrastructure.
- There is no distributed garbage collection. Therefore, call by reference is not supported by SOAP; a SOAP client does not hold any stateful references to remote objects.
- SOAP is operating system independent and not tied to any programming language or component technology. It is object model neutral.

Due to these characteristics, it does not matter what technology is used to implement the client, as long as the client can issue XML messages. Similarly, the service can be implemented in any language, as long as it can process XML messages.

The characteristics of WSDL

The Web Services Description Language (WSDL) is an XML-based interface and implementation description language. WSDL1.1 provides a notation to formally describe both the service invocation interface and the service location.

WSDL allows a service provider to specify the following characteristics of a Web service:

- The name of the Web service and addressing information

- The protocol and encoding style to be used when accessing the public operations of the Web service
- Type information, including operations, parameters, and data types comprising the interface of the Web service, plus a name for the interface

A WSDL specification uses XML syntax, therefore, there is an XML schema for it.

The characteristics of UDDI

UDDI is both a client-side API and a SOAP-based server implementation that can be used to store and retrieve information on service providers and Web services.

UDDI is a technical discovery layer. It defines:

- The structure for a registry of service providers and services
- The API that can be used to access registries with this structure
- The organization and project defining this registry structure and its API

UDDI is a search engine for application clients rather than human beings. However, there is a browser interface for human users as well.

Business roles

Next we look at the roles a business and its Web service-enabled applications can take. Three roles can be identified:

- Service broker
- Service provider
- Service requester

Service broker

The Web service broker is responsible for creating and publishing the UDDI registry. UDDI registries can be provided in two forms:

- Public registries, such as the IBM UDDI Business Registry and the IBM UDDI Business Test Registry:

  `http://www.ibm.com/services/uddi/protect/registry.html`
  `http://www.ibm.com/services/uddi/testregistry/protect/registry.html`
- Private registries such as the UDDI registry provided with IBM WebSphere Application Server

The service broker does not have to be a public UDDI registry. There are other alternatives, for example a direct document exchange link between the service provider and the service requester.

Service provider

The service provider creates a Web service and publishes its interface and access information to the service registry.

Figure 3-12 on page 55 Web service provider architecture shows in more detail the application architecture of a Web service provider. Using this architecture, we do not have to change the existing business logic or business objects in order to create a Web service from the existing enterprise business objects.

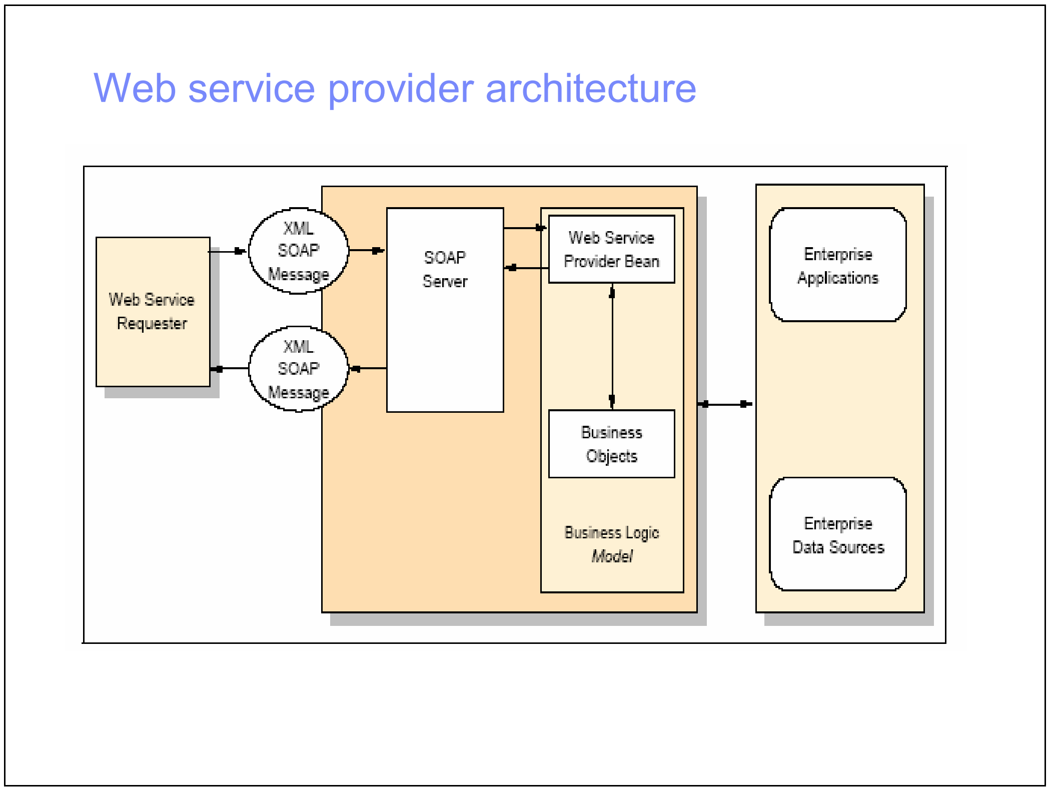


Figure 3-12 Web service provider architecture

A WSDL specification consists of two parts, the service interface and the service implementation. Hence, service interface provider and service implementation provider are the two respective subroles for the service provider. The two roles can, but do not have to, be taken by the same business.

Service requester

The service requester locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its Web services.

In Figure 3-13 on page 56 Web service requester architecture shows the architecture of a Web service requester. Note that the architectural model follows the Model-View-Controller (MVC) pattern, with the servlet as the main component of the controller; the JSP™ the main component of the View; and the commands and the Web services residing the Model layer. Web services provide a link to another system within the Model layer.

This is considered a best practice for building Web-based applications. We can see that Web services fit very easily into this model.

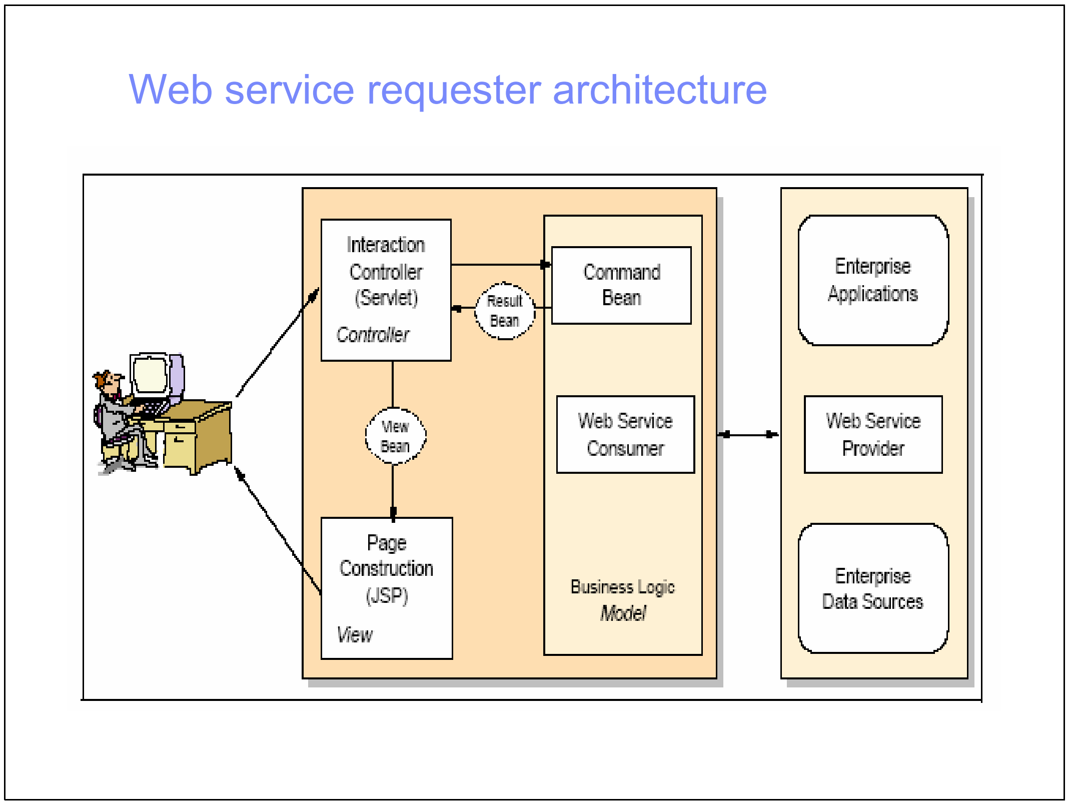


Figure 3-13 Web service requester architecture

SOAP messaging mechanisms

The next design point in architecting a Web service is to choose the SOAP messaging mechanism to use. Figure 3-14 on page 57 shows the two general categories of Web services SOAP messaging mechanisms:

- SOAP Remote Procedure Calls (RPC)-based Web services
- SOAP message-oriented Web services

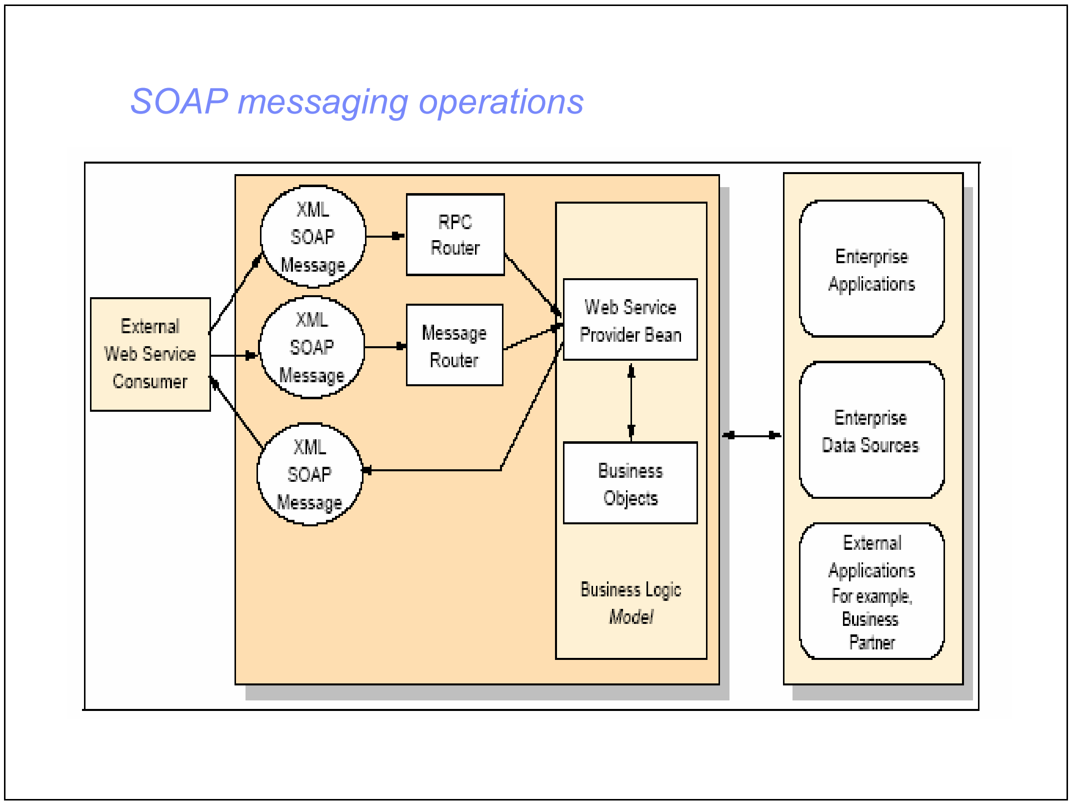


Figure 3-14 SOAP messaging operations

RPC versus message-oriented

The advantages and disadvantages of the SOAP RPC approach versus the SOAP message-oriented Web service approach can be summarized as follows:

- SOAP RPC advantage:
  – Simpler development.
- SOAP RPC disadvantages:
  – Requester is too dependent on the availability of the Web service provider.
- SOAP message-oriented advantages:
  – Less dependency on the Web service provider availability
  – Works well for exchanging large documents
  – Works well from a nonrepudiation perspective because documents can be signed digitally and stored at both ends
  – Enables extended enterprise electronic workflow and business process integration using asynchronous integration
- SOAP message-oriented disadvantage:
  – Relatively more complex development because it uses assured delivery of asynchronous messages and can require compensating transactions.

Static versus dynamic Web services discovery

Our next design point is to decide if the Web service requester will use static or dynamic discovery of available Web services. The requester has to begin with the WSDL file that describes the interface and implementation specification of the Web service to be invoked. This WSDL file can be retrieved dynamically using a service registry, or statically, as shown in Figure 3-15.

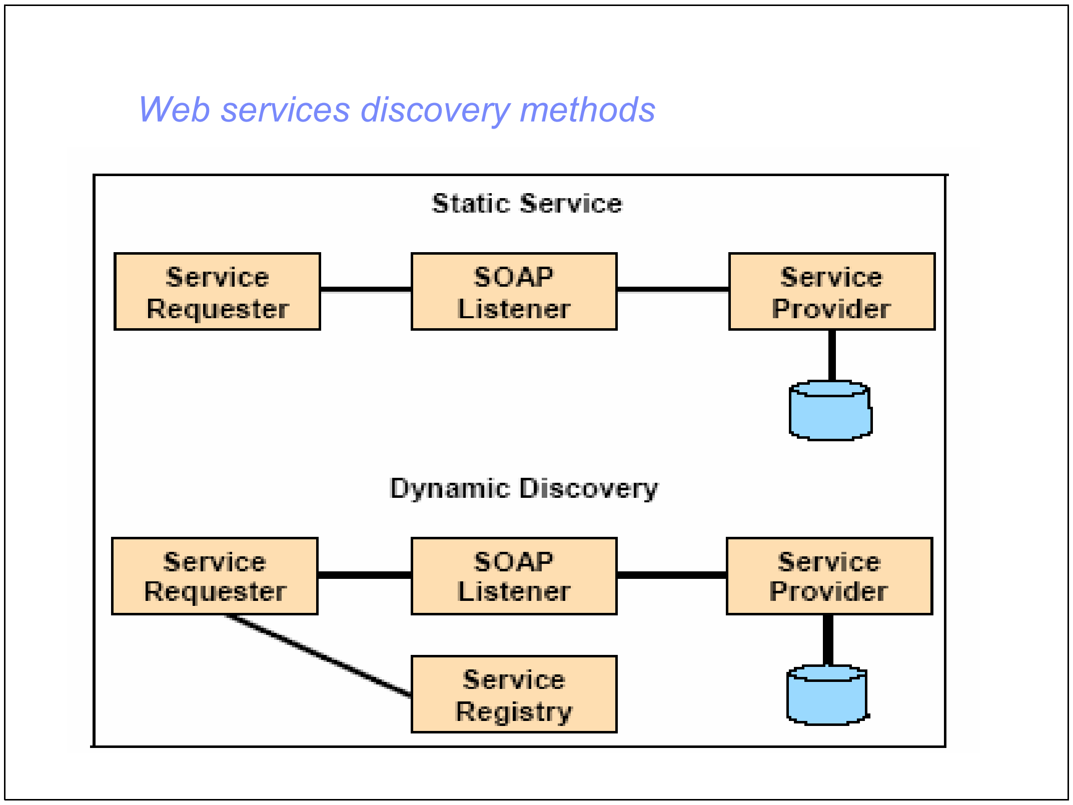


Figure 3-15 Web services discovery methods

Three types of discovery methods for requesters can be identified. They import interface and implementation information at different points in time (build time versus. runtime):

- Static service

  No public, private, or shared UDDI registry is involved. The service requester obtains a service interface and implementation description through a proprietary channel from the service provider (an e-mail, for example), and stores it in a local configuration file.

- Provider-dynamic

  The service requester obtains the service interface specification from a public, private, or shared UDDI registry at build time and generates proxy code for it. The service implementation document identifying the service provider is dynamically discovered at runtime (using the same or another UDDI registry).

- Type-dynamic

  The service requester obtains both the service interface specification and the service implementation information from a public, private, or shared UDDI registry at runtime. No proxy code is generated; the service requester directly uses the more generic SOAP APIs to bind to the service provider and invoke the Web service.

Message structure

The Web services specification does not mandate any particular message structure. The message structure is defined by the service provider. Message structures can be anything from simple strings to complex XML documents.

SOAP encoding versus literal encoding

SOAP encoding, the infamous set of rules often referred to as Section 5 encoding (W3C SOAP 1.1 Section 5, `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`) after its location in the specification, was introduced to provide standard rules for encoding data within SOAP envelopes. Simple services and clients could simply agree to follow a set of rules for encoding data to XML in order to make writing services and clients easier. But SOAP encoding is only a suggestion in the specification. Thus, when a product claims SOAP compatibility, it is not explicitly claiming SOAP encoding compatibility. This is why the available higher-level APIs cannot be correct in general when they do automatic marshalling of data types. Just as the extensibility of SOAP with regard to transports often causes implicit assumptions that create compatibility problems, so does this extensibility with regard to payload data encoding. To know whether the client API will generate SOAP envelopes that a specific Web service will understand, we must be explicitly aware of the data encoding that the Web service expects and whether that encoding is supported by the client API.

An alternative is to use literal encoding where the payload of a SOAP message is defined completely by a specific schema, often an XML Schema. Instead of having the Web service and the client agree to follow a set of rules for serializing the data, they agree on the exact format of the data. If the format is described by using an XML Schema, a development tool can read it and provide automatic marshalling of the data from the native language structures into XML. In this case, all the toolkit has to understand is the entire XML Schema specification instead of the combination of the particular encoding rules as well as the chosen type system. The only issue left with using literal encoding is how the tool finds the particular service's XML Schema. This issue is solved by WSDL.

Currently, no machine-understandable standard exists for describing data models for use with SOAP Section 5 encoding, so developers are leaning towards literal encoding with XML Schema. Development tools often provide features such as syntax assistance and data model validation with XML Schema. This may change soon, however, as the Web Services Description working group is considering creating a language to describe SOAP Section 5 data models as part of the WSDL binding for SOAP in WSDL Version 1.2.

Synchronous versus asynchronous Web services

Our next design point is selecting the kind of messaging we want to implement. Our choices are synchronous or asynchronous. The Web services specifications define synchronous operations only. However, Web services are by their very nature somewhat asynchronous. From the perspective of the Web service provider it must, in effect, be a listener and be prepared to accept requests asynchronously from the requester. From the consumer or requester side, the application can be designed for either synchronous or asynchronous operation.

Although Web services defines synchronous operations only, there is nothing in the specifications to preclude asynchronous operations. Generally, the Web services requester has no guarantee of when, or if, it will receive a response. Beyond that, there are also situations where the Web service provider needs to perform some external operation, or wait for human intervention, or call another service that would introduce a delay in the response. Synchronous Web services are suitable when the Web service provider can provide the required response instantaneously, such as when getting a stock quote. Here we are, in effect, using Web services as another RPC mechanism. Current tools are more focused on this type of Web service. Asynchronous Web services are suitable when the Web service provider is unable to provide the required response instantaneously, for a variety of reasons as mentioned above.

Asynchronous operations are usually driven by the asynchronous nature of the business transaction itself. Asynchronous Web services are suitable for document exchange between enterprises. It is important to separate this design point from the reliability of the underlying transport mechanism. We will discuss this in further detail in the next sections.

The designer of a Web services requester needs to decide how to handle asynchronous responses and how to ensure that his or her implementation is compatible with the way in which a service provider supports asynchronous operations. One option for the requester is to issue a request and then block its thread of execution waiting for a response, but for obvious reasons this is not a good alternative; among other problems, it results in resource inefficiencies and raises transactional and scalability issues. The preferred solution is to build asynchronous behavior into the Web services requester. The requester makes a request as part of one transaction and carries on with the thread of execution. The response message is then handled by a different thread within a separate transaction. In this model, the requester requires a notification mechanism and a registered listener component to receive responses. Likewise, there must be a correlator (a correlation or transaction ID) exchanged between the service requester and the service provider for associating responses with their requests.

Transports and local interfaces

The transports that can be used for Web services communications vary in their capabilities to facilitate the support of asynchronous operations. Thus, it is not only Web services behavior that can be described as either asynchronous or synchronous; the transport used for exchanging Web services messages also falls into one category or the other. Transports whose interfaces inherently support the correlation of response messages to request messages for application use and support a push and pull type of message exchange are often described as being asynchronous transports. Synchronous transports do not provide these facilities and, when used for asynchronous operations, require that the applications (the client and service provider, for the purposes of this discussion) manage the correlation of messages exchanged by not only defining how the correlator will be passed within each message, but by also matching responses with requests. Examples of transports that can be used in support of asynchronous operations are listed in Table 3-3.

Table 3-3 Web services transports type

Asynchronous transports	Synchronous transports
HTTPR JMS IBM WebSphere MQ Messaging MS Messaging	HTTP HTTPS RMI/IIOP SMTP

Typically, when business partners use Web services to integrate their business processes, they prefer to use Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol over Secure Socket Layer, or HTTP over SSL (HTTPS), and Hypertext Transfer Protocol Reliable, HTTPR as transports for communications across the Internet:

See: `http://www-128.ibm.com/developerworks/library/ws-httprspec/`

Correlation ID

Regardless of the transport being used for an asynchronous operation, because the response to a request is expected to be received at a later time, there must be a mechanism to correlate the response with the request. Web services requesters and providers must agree upon a correlation ID scheme. They also must agree upon who is responsible for generating the correlation ID.

Return address

In addition there must be an agreed-upon mechanism to identify the return address to which to send the response. You could set up a return address in a profile database or its return address could be part of every request.

The asynchronous transports enable a client to continue processing on its thread of execution immediately after requesting a service invocation. They also provide mechanisms to enable a client to determine the status of its Web service requests, and to retrieve responses to those requests.

Web service implementations that do not provide the ability to initiate the transmission of a response on a separate thread of execution cannot be used for asynchronous operations. Examples of such implementations would be those that use EJBs to front-end database applications or implementations that provide access to enterprise systems through the use of local interfaces such as Java Connector Architecture (JCA).

Asynchronous Web services approaches

When implementing an asynchronous mechanism in Web Services, the preferred solution is to build asynchronous behavior into the Web services requester. The requester makes a request as part of one transaction and carries on with the thread of execution. The response message is then handled by a different thread within a separate transaction. In this model, the requester requires a notification mechanism and a registered listener component to receive responses. Similarly, there must be a correlator (a correlation or transaction ID) exchanged between the service requester and the service provider for associating responses with their requests.

A typically asynchronous scenario would include the following:

- Production and transmission of a request message by a service requester
- Consumption of the request message by the service provider
- Production and transmission of a response message by the service provider
- Consumption of the response message by the service requester

Development strategies for Web service providers

A service provider can choose between three different development styles when defining the WSDL and the Java implementation for her Web service:

- Top-down

  When following the top-down approach, both the server-side and client-side Java code are developed from an existing WSDL specification.

- Bottom-up

  If some server-side Java code already exists, the WSDL specification can be generated from it. The client-side Java proxy is still generated from this WSDL document.

- Meet-in-the-middle

  The meet-in-the-middle (MIM) development style is a combination of the two previous ones. There are two variants:

  - MIM variant 1

    Some server-side Java code is already there. Its interface, however, is not fully suitable to be exposed as a Web service. For example, the method signatures might contain unsupported data types. A Java wrapper is developed and used as input to the WSDL generation tools in use.

– MIM variant 2

  There is an existing WSDL specification for the problem domain; however, its operations, parameters, and data types do not fully match with the envisioned solution architecture. The WSDL is adopted before server-side Java is generated from it.

In the near future, we expect most real-world projects to follow the meet-in-the-middle approach, with a strong emphasis on its bottom-up elements. This is MIM variant 1, starting from and modifying existing server-side Java and generating WSDL from it.

Level of integration between requester and provider

In a homogeneous environment, client and server (requester and provider) use the same implementation technology, possibly from the same vendor. They might even run in the same network.

In such an environment, runtime optimizations such as performance and security improvements are possible. We expect such additional vendor-specific features to become available as the Web services technology evolves.

We do not recommend enabling such features, however, because some of the main advantages of the Web service technology such as openness, language independence, and flexibility can no longer be exploited. Rather, you should design your solution to loosely couple requester and provider, allowing heterogeneous systems to communicate with each other.

3.5.3 The key challenges in Web services

Web services can potentially revolutionize application integration by providing a layer of abstraction between the technology that requests a service and the technology that provides the service. In order to achieve this, though, there are still technical challenges that have to be addressed. This section briefly describes a few key issues, such as the Extended Web Services Architecture, security, interoperability, quality of service, and distributed transactions.

In February 2004, W3C released a draft Web Service Architecture specification that identifies the functional components, the relationships among those components, and establishes a set of constraints to guide the desired properties of the overall architecture.

See the W3C Web Services Architecture specification for more detail:

http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

3.5.4 Security considerations

Web services security is one of the bigger challenges in implementing Web services-based systems. Threats to Web services involve threats to the host system, the application and the entire network infrastructure. According to W3C, “at this time, there are no broadly-adopted specifications for Web services security”. In this section we will discuss the requirements for providing security for Web service, and a few of approaches.

At high level, the requirements for security are as following:

- Identity, which enables a business or service to know who you are
- Authentication, which enables you to verify that a claimed identity is genuine
- Authorization, which lets you establish who has access to specific resources

- Data integrity, which lets you establish that data has not been tampered with
- Confidentiality, which restricts access to certain messages only to intended parties
- Nonrepudiation, which lets you prove a user performed a certain action such that the user cannot deny it
- Auditing, which helps you to keep a record of security events

With IBM WebSphere Application Server V6, you have the following options for securing Web services:

- Message-level security using Web services security (WS-Security)
- Transport-level security using TLS/SSL

Figure 3-16 provides an overview of Web services security.

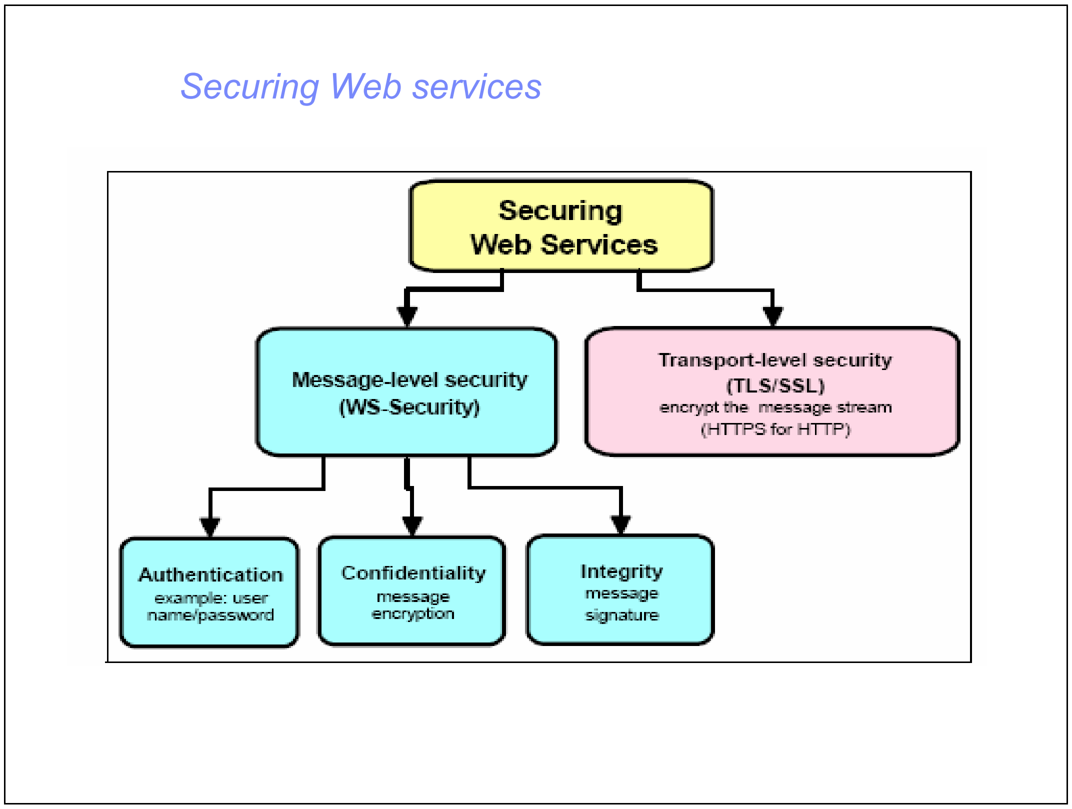

Securing Web services
Securing Web Services
Message-level security (WS-Security)
Transport-level security (TLS/SSL) encrypt the message stream (HTTPS for HTTP)
Authentication example: user name/password
Confidentiality message encryption
Integrity message signature

Figure 3-16 Securing Web services

For a full discussion of Web services security and implementation examples, see *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

WS-Security

The WS-Security specification defines message-level security that provides for message content integrity and confidentiality. Unlike SSL, WS-Security can provide end-to-end message-level security. This means that the message security can be protected even if the message goes through multiple services, called intermediaries. WS-Security is independent of the transport layer protocol and can be used for any Web service binding (for example, HTTP, SOAP, RMI).

Here are some simple guidelines as to when WS-Security should be used:

- Multiple parts of message can be secured in different ways.

  You can apply multiple security requirements, such as integrity on the security token (user ID and password) and confidentiality on the SOAP body.
- Intermediaries can be used.

  End-to-end message-level security can be provided through any number of intermediaries.
- Non-HTTP transport protocol is used.

  WS-Security works across multiple transports (also change of transport protocol) and is independent of the underlying transport protocol.
- User authentication is possible.

  Authentication of multiple party identities is possible.

WS-Security represents only one of the layers in a complex, secure Web services solution design. A more general security model is required to cover other security aspects, such as logging and non-repudiation. The definition of those requirements is defined in a common Web services security model framework.

The Web services security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security: identification, authentication, authorization, integrity, confidentiality, auditing, and non-repudiation. It is based on the WS-Security specification, co-developed by IBM, Microsoft, and VeriSign.

The Web services security model is schematically shown in Figure 3-17.

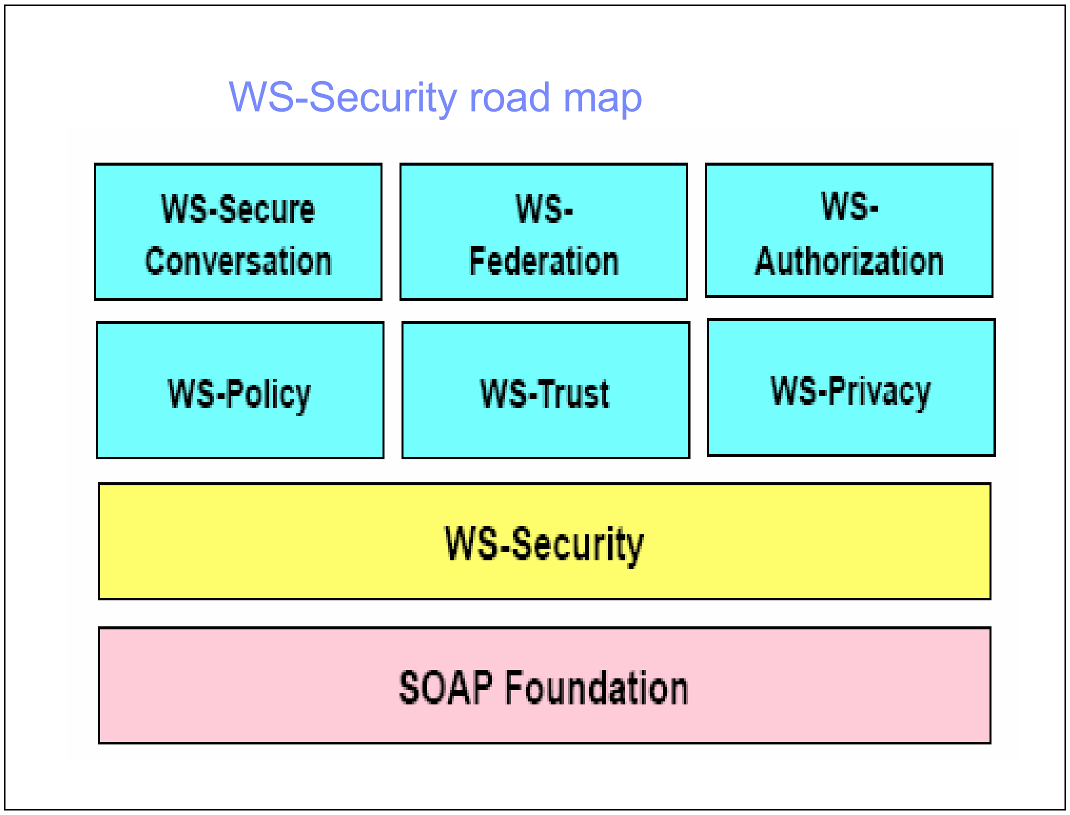


Figure 3-17 WS-Security road map

These specifications include different aspects of Web services security:

- WS-Policy - Describes the capabilities and constraints of the security policies on intermediaries and endpoints (for example, required security tokens, supported encryption algorithms, and privacy rules).
- WS-Trust - Describes a framework for trust models that enables Web services to securely interoperate, managing trusts and establishing trust relationships.
- WS-Privacy - Describes a model for how Web services and requestors state privacy preferences and organizational privacy practice statements.
- WS-Federation - Describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.
- WS-Authorization - Describes how to manage authorization data and authorization policies.
- WS-SecureConversation - Describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys.

The combination of these security specifications enables many scenarios that are difficult or impossible to implement with today's more basic security mechanisms such as transport securing or XML document encryption.

For more information, see: *Security in a Web Services World: A Proposed Architecture and Roadmap*, proposed by IBM and Microsoft.

`http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/`

Transport-level security

To secure HTTP, transport-level security can be applied. Transport-level security is a well-known and often used mechanism to secure HTTP Internet and intranet communications. Transport-level security is based on Secure Sockets Layer (SSL) or Transport Layer Security (TLS) that runs beneath HTTP.

HTTPS allows client-side and server-side authentication through certificates, which have been either self-signed or signed by a certification agency.

For Web services bound to the HTTP protocol, HTTPS/SSL can be applied in combination with message-level security (WS-Security).

Unlike message-level security, HTTPS encrypts the entire HTTP data packet. There is no option to apply security selectively on certain parts of the message. SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections.

SOAP/HTTP transport-level security

Although HTTPS does not cover all aspects of a general security framework, it provides a security level regarding party identification and authentication, message integrity, and confidentiality. It does not provide authentication, auditing, and non-repudiation. SSL cannot be applied to other protocols, such as JMS. To run HTTPS, the Web service port address must be in the form https://.

Even with the WS-Security specification, SSL should be considered when thinking about Web services security. Using SSL, a point-to-point security can be achieved.

Here are a few simple guidelines to help decide when transport-level security should be used:

- No intermediaries are used in the Web service environment.

With intermediaries, the entire message has to be decrypted to access the routing information. This would break the overall security context.

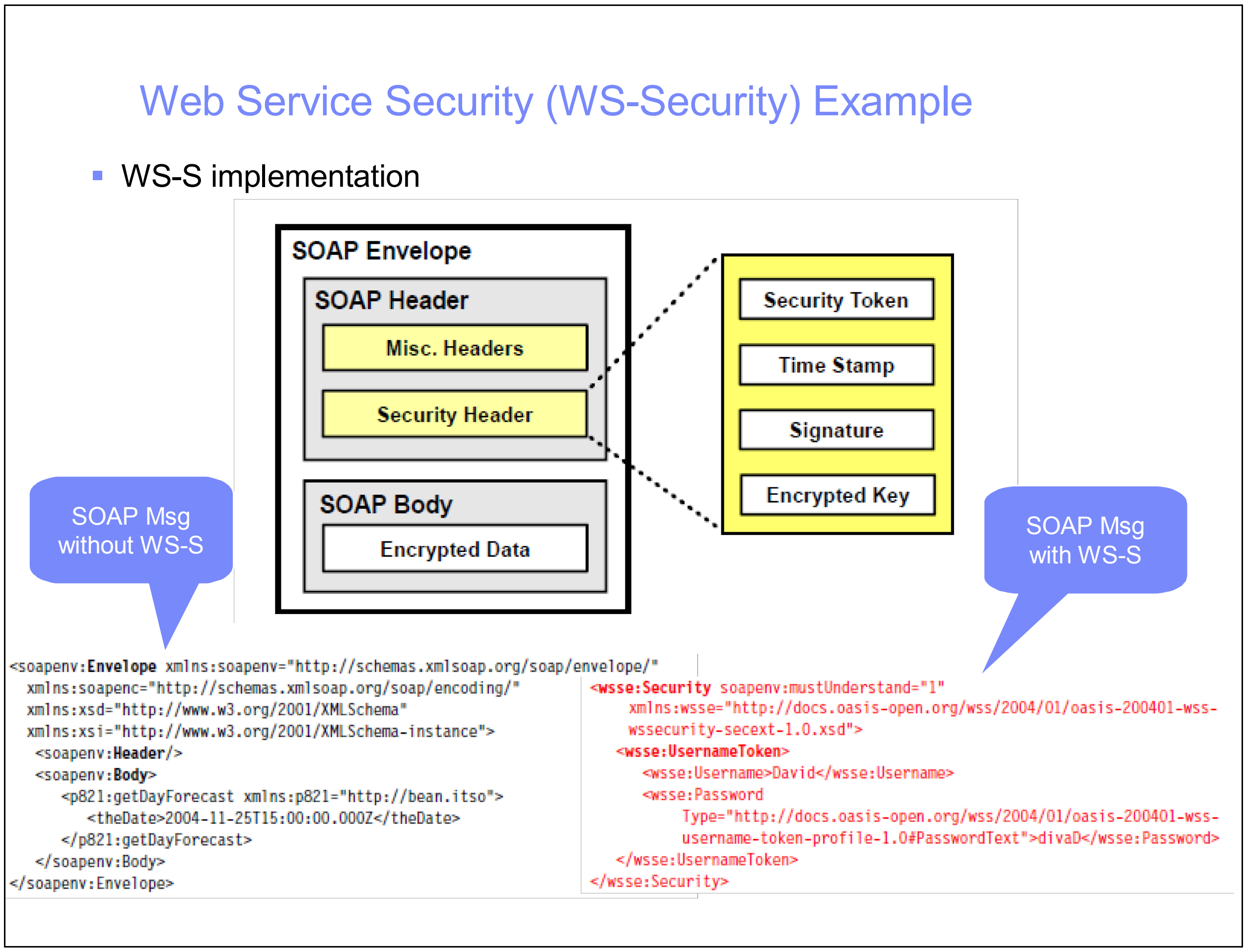
- The transport is only based on HTTP.

  No other transport protocol can be used with HTTPS.

- The Web services client is a stand-alone Java program.

  WS-Security can only be applied to clients that run in a J2EE container (EJB™ container, Web container, application client container). HTTPS is the only option available for stand-alone clients.

Figure 3-18 provides examples of how to implement WS-S. There are two code examples, one of them has WS-S, another without WS-S.



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header/>
   <soapenv:Body>
      <p821:getDayForecast xmlns:p821="http://bean.itso">
         <theDate>2004-11-25T15:00:00.000Z</theDate>
      </p821:getDayForecast>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<wsse:Security soapenv:mustUnderstand="1"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-secext-1.0.xsd">
  <wsse:UsernameToken>
     <wsse:Username>David</wsse:Username>
     <wsse:Password
          Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
          username-token-profile-1.0#PasswordText">divaD</wsse:Password>
  </wsse:UsernameToken>
</wsse:Security>
```

Figure 3-18 Web Service Security (WS-Security) Example

3.6 Additional information for SOA

For more information, see:

- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- *Patterns: Implementing Self-Service in an SOA* Environment, SG24-6680

- *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6,* SG24-6494
- *Patterns: Implementing an SOA Using an Enterprise Service Bus,* SG24- 6346
- *XML for DB2 Information Integration*, SG24-6994
- WebSphere Application Server Version 6.0 Information Center, available at:

  `http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp`
- The XML Encryption workgroup home page is available at:

  `http://www.w3.org/Encryption/`
- The WS-Security specification 1.0 is available at:

  `http://www.ibm.com/developerworks/library/ws-secure/`
- *Security in a Web Services World: A Proposed Architecture and Roadmap*, proposed by IBM and Microsoft.

  `http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/`
- OASIS WS-Security 1.0 and token profiles is available at:

  `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`
- Web Services Security: SOAP Message: Errata 1.0

  `http://www.oasis-open.org/committees/download.php/9292/oasis-200401-wsssoap-mes`
  `sage-security-1%200-errata-003.pdf`

4

SOA and user interfaces with portals

In Chapter 3, we provided an introduction to service-oriented architecture (SOA), technical standards and implementation guidelines. In this chapter we describe the relationship between enterprise portals and SOA. Many enterprises that adopted an enterprise portal are now turning their attention to the IT infrastructure and increasingly utilizing SOA as an approach and strategy. By applying SOA along with portals and settings in a standard-based, service-oriented environment, different systems are being service enabled that can be orchestrated to directly support changing business needs.

Portals are becoming the de facto, front-end, user interfaces for IBM SOA strategy - by offering the customers products that simplify development and deployment of composite applications, of applications that flexibly combine data services from multiple existing investments, to realizing the benefits of SOA. A service-oriented programming model can simplify the development of program-to-human interactions by abstracting interfaces, standardizing messages, and aggregating independent information sources at the presentation layer under the control of a user or administrator.

SOA gives organizations unprecedented flexibility to compartmentalize the business processes trapped within their application silos, and to re-purpose and integrate them into new business process applications as needed. Portals let organizations extend this new class of applications, quickly and inexpensively, to all the different users that may be involved in executing these business processes. Plus when you need to, you can quickly change those applications. In today's business world, end-user demands continue to outpace the ability of the IT organization to respond. The key is giving end-user communities a managed environment in which they can be more self-sufficient.

SOA and composite applications are very powerful concepts for enabling companies to provide more responsive and flexible business solutions. Portals extend this even further by enabling IT to give process analysts and business users additional tools for assembling IT assets, applications and processes in a way that meets their needs.

SOA is taking process applications created years ago, based on the idea that everything takes place in house or in one user community, and extending them, rather than rewriting them, so that others with a role in the process can participate. And portals can be the

© Copyright IBM Corp. 2006. All rights reserved.

interface for these extended applications, the way these new communities of users will interact with these business processes.

As Web services will become the predominant method for SOA, making information and applications available programmatically via the Internet, portals need to allow for integration of Web services as data sources and as remote application components.

We see two important options for the use of Web services in conjunction with portals:

- Portlets running on a portal server can access a Web service to obtain information or invoke remote methods provided by the Web service.
- Portals can publish portlets as remote portlet Web services to make them available to other portals in a way to easily find and integrate them.

This chapter contains the following topics:

- An introduction to portals and portlets
- The standardization of portlets (Java standardization request - JSR-168)
- Portals and SOA
- Additional information for portals and SOA

4.1 An introduction to portals and portlets

A portal offers a single point of personalized, unified access to applications, content, processes, and people. A portal delivers integrated content and applications, plus offers a unified, collaborative workplace. A portal also provides other valuable functions such as security, search, and workflow. A portal is an open, standards-based framework supporting a wide array of options across databases, directories, platforms, and security. Portals are the next-generation desktop, delivering e-business applications over the Web to many different client devices.

Portals are designed to meet the needs of all enterprises, from small and medium businesses to the largest enterprises that demand the most scalable, secure, and robust infrastructure. Portal applications have several important features. They can collect content from a variety of sources and present them to the user in a single unified format. The presentation can be personalized so that each user sees a view based on their own characteristics or role. The presentation can be customized by the user to fulfill their specific needs. They can provide collaboration tools, which allow teams to work in a virtual office. They can provide content to a range of devices, formatting and selecting the content appropriately according to the capabilities of the device.

A consistent, integrated user experience is achieved by portals that do not only aggregate components into a single view, but in addition allow integration of these components within the context. This is often called integration on the glass, because all the applications are integrated in context by the portal into one single window on the monitor of the portal end user. This is a very powerful concept that in today’s world of widely fractured IT infrastructures enables the delivery of consistent and integrated views on multiple IT services. Integration on the glass improves the user experience and productivity of the IT user; instead of dealing with different IT systems with potential different user interfaces, integration on the glass provides a single, consistent view.

4.1.1 What is a portal?

A portal is a Web based application that –commonly – provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of Information

Systems. Aggregation is the action of integrating content from different sources within a Web page. A portal may have sophisticated personalization features to provide customized content to users. Portal pages may have different sets of portlets to create content for different users.

A complete portal solution should provide users with convenient access to everything they need to get their tasks done anytime, anywhere, in a secured manner.

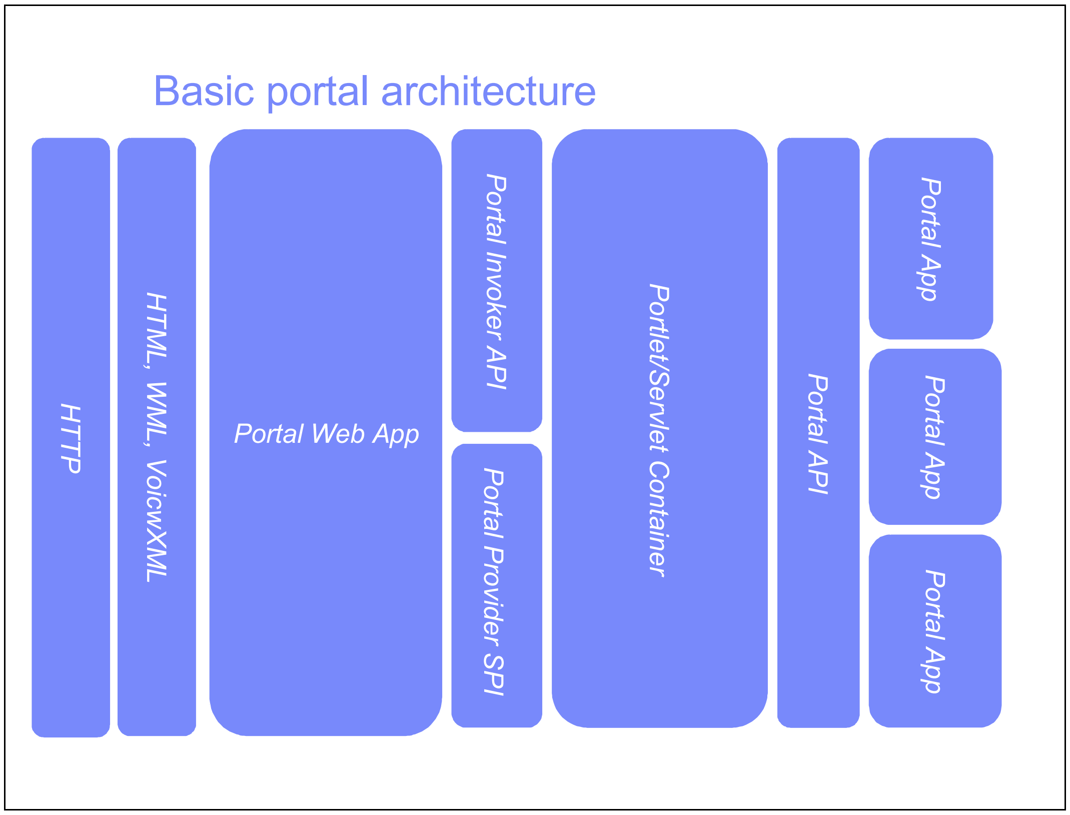


Figure 4-1 Basic portal architecture

Figure 4-1 depicts the basic architecture of a portal. The client request is processed by the portal Web application, which retrieves the portlets that appear on the current page for the current user. The portal Web application then calls the portlet container for each portlet to retrieve its content. The portlet container provides the runtime environment for the portlets and calls the portlets via the Portlet API.

4.1.2 Portal applications

An organization has many types of users that rely on its information and services. Customers, partners, and employees each have specific and often diverse needs. To address these needs, many different kinds of portals have been implemented. These can be categorized as follows:

Business-to-Consumer (B2C)

This type of extended enterprise portal (extranet) is associated with customer relationship management (CRM) and provides consumers with direct access to a variety of content, for example, product manuals and availability or price lists. Portal customers might also purchase products, check order status, and communicate with customer support. Like any other portal, a B2C portal is usually tailored to match customer needs.

Business-to-Business (B2B)

Another type of extended enterprise portal, B2B portals participate in supply chain management (SCM) by providing personalized access to business information by suppliers, resellers, and distributors. A typical B2B portal might provide a business partner with access to purchase orders, invoices, statements, and confirmations. Application integration is also required to integrate business processes in procurement, billing, manufacturing, and distribution areas.

Business-to-Employee (B2E)

B2E portals (also known as intranet portals) generally serve as a means to aggregate and disseminate corporate information and services to an organization's employees. There are two basic types of B2E portals:

- **Employee portals** provide access to relevant content such as company news, HR information, search engines, sources of expertise, reports, and other types of information generally applicable to all employees. These portals can enable employees to communicate and collaborate via chat rooms, discussion groups, and so on. Typically, an employee portal also allows for self-service, where an employee can sign up for classes or benefits, change personal information, and so on.
- **Knowledge worker portals** are aimed a particular role or set of roles such as sales. These portals often integrate content in order to support a particular process or processes. For example, an automotive technician might require resources from a number of applications such as service history, scheduling, or parts availability.

Portal applications have several important features:

- They can collect content from a variety of sources and present them to the user in a single unified format.
- The presentation can be personalized so that each user sees a view based on their own characteristics or role.
- The presentation can be customized by the user to fulfill their specific needs.
- They can provide collaboration tools, which allow teams to work in a virtual office.
- They can provide content to a range of devices, formatting and selecting the content appropriately according to the capabilities of the device.

4.1.3 Portal page

In Figure 4-2 on page 73, basic portal page components depicts the basic portal page components. The portal page itself represents a complete markup document and aggregates several portlet windows. A portlet window consists of a title bar with the title of the portlet, decorations, and the content produced by the portlet (markup fragment). The decorations may include buttons to change the window state of the portlet (for example, maximize or minimize the portlet) and buttons to change the mode of a portlet (for example, show help or edit the predefined portlet settings).

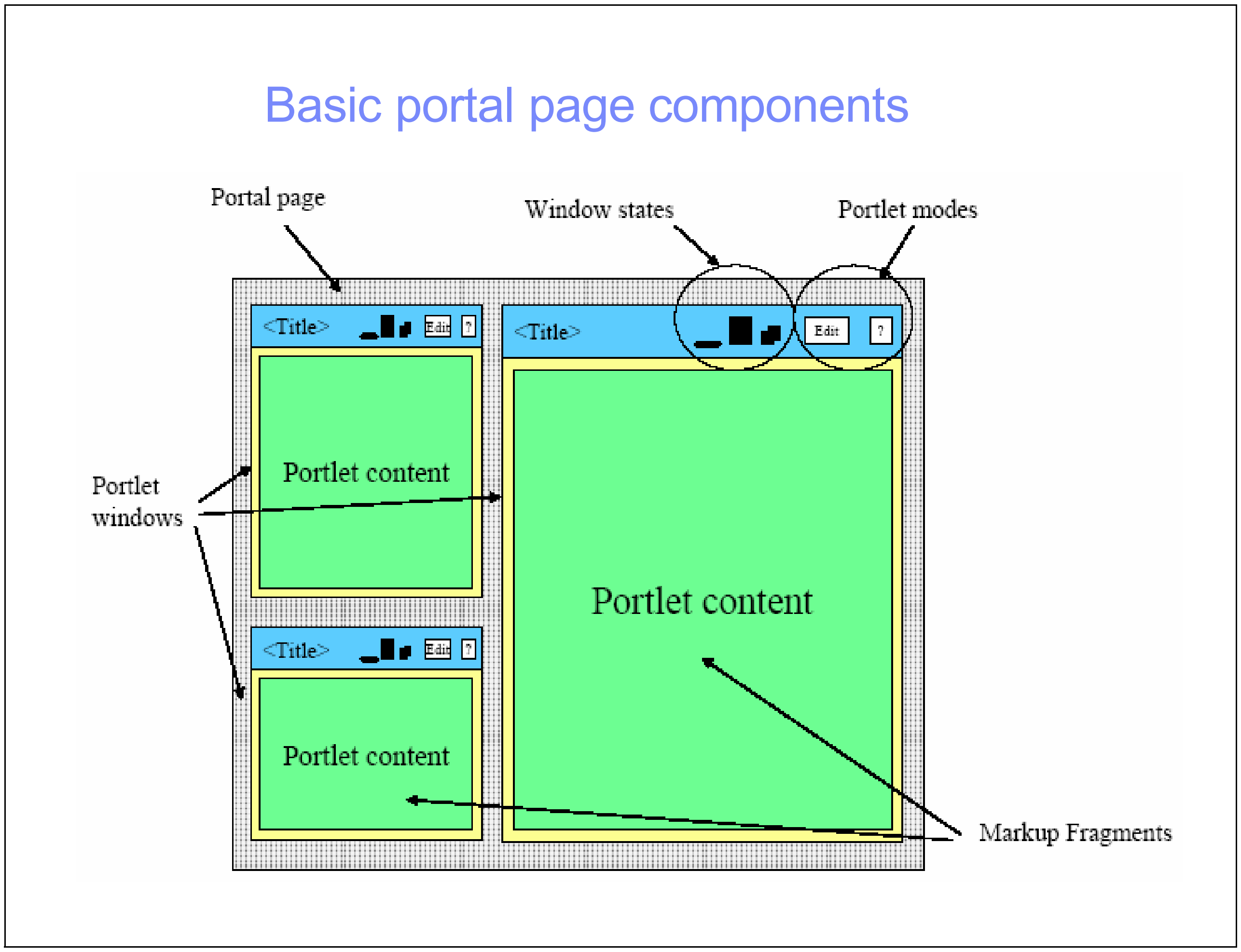


Figure 4-2 Basic portal page components

4.1.4 The portal engine

A portal provides a pure Java engine whose main responsibility is to aggregate content from different sources and serve the aggregated content to multiple devices. The portal engine also provides a framework that allows the presentation layer of the portal to be decoupled from the portlet implementation details. This allows the portlets to be maintained as discrete components.

Figure 4-3 on page 74 shows the WebSphere Portal Engine components.

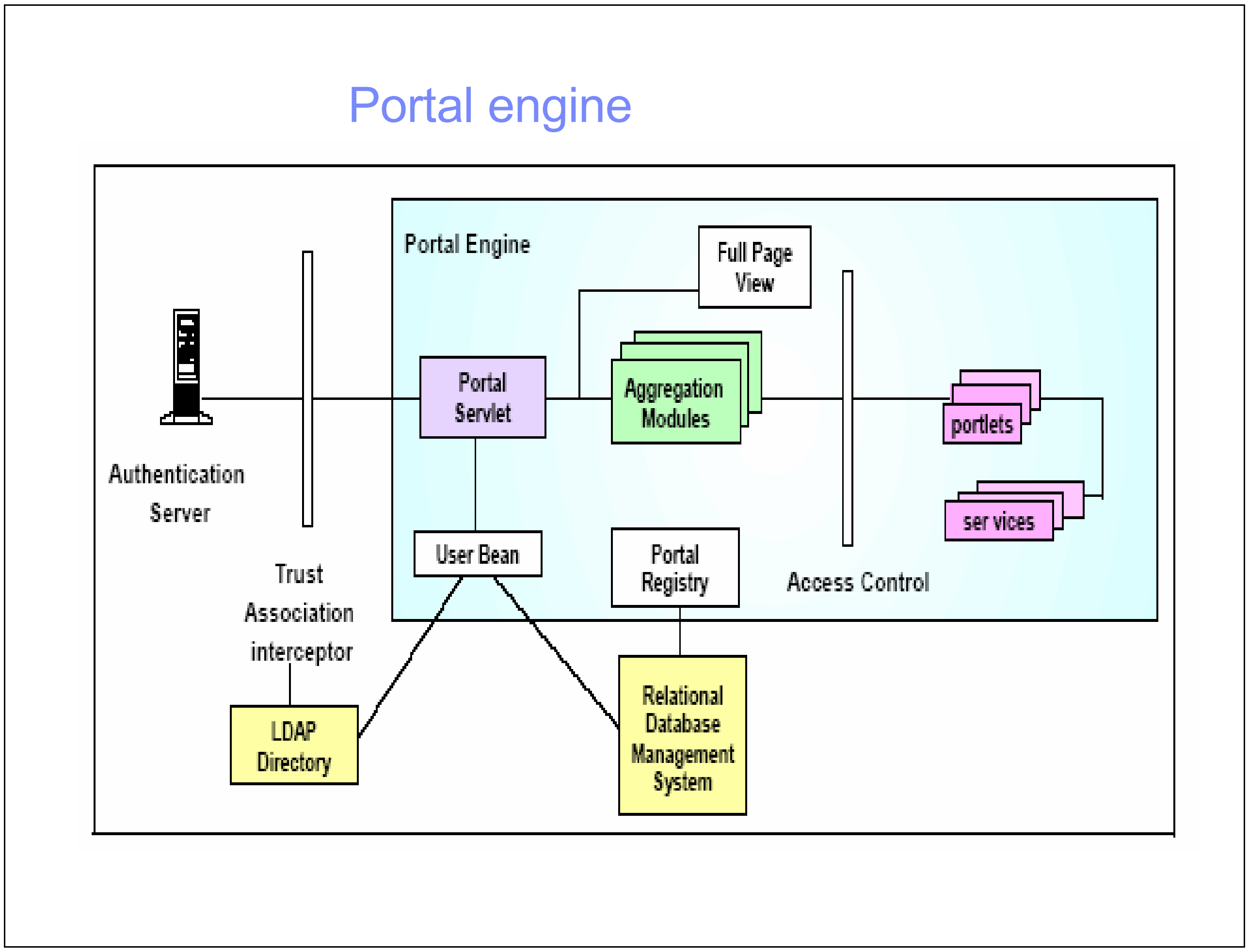


Figure 4-3 Portal engine

The *Authentication Server* is a third-party authentication proxy server that sits in front of the Portal engine. Access to portlets is controlled by checking access rights during page aggregation, page customization, and other access points.

The *Portal Servlet* is the main component of the Portal engine. The portal servlet handles the requests made to the portal. The portal requests are handled in two phases. The first phase allows portals to send event messages between themselves. In the second phase, the appropriate Aggregation Module for the requesting device renders the overall portal page by collecting information from all the portlets on the page and adding standard decorations such as title bars, edit buttons, and so on.

4.1.5 What is a portlet?

A *portlet* is a Java technology based Web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems.

The content generated by a portlet is also called a fragment. A fragment is a piece of markup (for example, HTML, XHTML, WML) adhering to certain rules and can be aggregated with other fragments to form a complete document, the portal page. The life cycle of a portlet is managed by the portlet container.

Web clients interact with portlets via a request/response paradigm implemented by the portal. Normally, users interact with content produced by portlets, for example by following links or submitting forms. This user interaction results in a portlet action being received by the portal that is forwarded to the portlet targeted by the user's interactions.

The content generated by a portlet may vary from one user to another depending on the user configuration for the portlet. Also from a user's perspective, a portlet is a window on a portal site that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable modules that are designed to run inside a portlet container of a portal server.

Tools like *WebSphere Portlet Factory* is a comprehensive portlet development environment that automates the process of creating service-oriented architecture (SOA)-based portlets. With WebSphere Portlet Factory, developers can quickly and easily leverage their company's core assets, using automation to assemble them into custom, high-value portlets. These portlets are dynamic, robust Enterprise Edition (J2EE) applications, based on SOA. In addition, portlets created with WebSphere Portlet Factory can automatically present themselves as stand-alone Web applications, Web services, or native WebSphere and Java Standardization Request 168 (JSR-168) portlets, without requiring any coding or duplicating assets. Business users can also then modify these portlets in real time to meet changing business requirements without having to go back to IT.

Portlets are pluggable components running inside a portal's portlet container, similar to servlets in many aspects. Portlets are written to a portlet API similar to the servlet API. However, portlets run in a portal environment, while servlets run stand-alone in a servlet container. While servlets communicate directly with their clients, portlets are invoked indirectly via the portal application. In order to properly run in the context of a portal, portlets must produce content that is suited for aggregation in larger pages, for example, portlets should produce markup-fragments adhering to guidelines that assure that the content generated by many different portlets can be aggregated.

When the portal receives a servlet request, it generates and dispatches events for any portlet affected by parameters in the request and then invokes all portlets that have to be displayed through the portlet invocation interface (see Figure 4-4 on page 76).

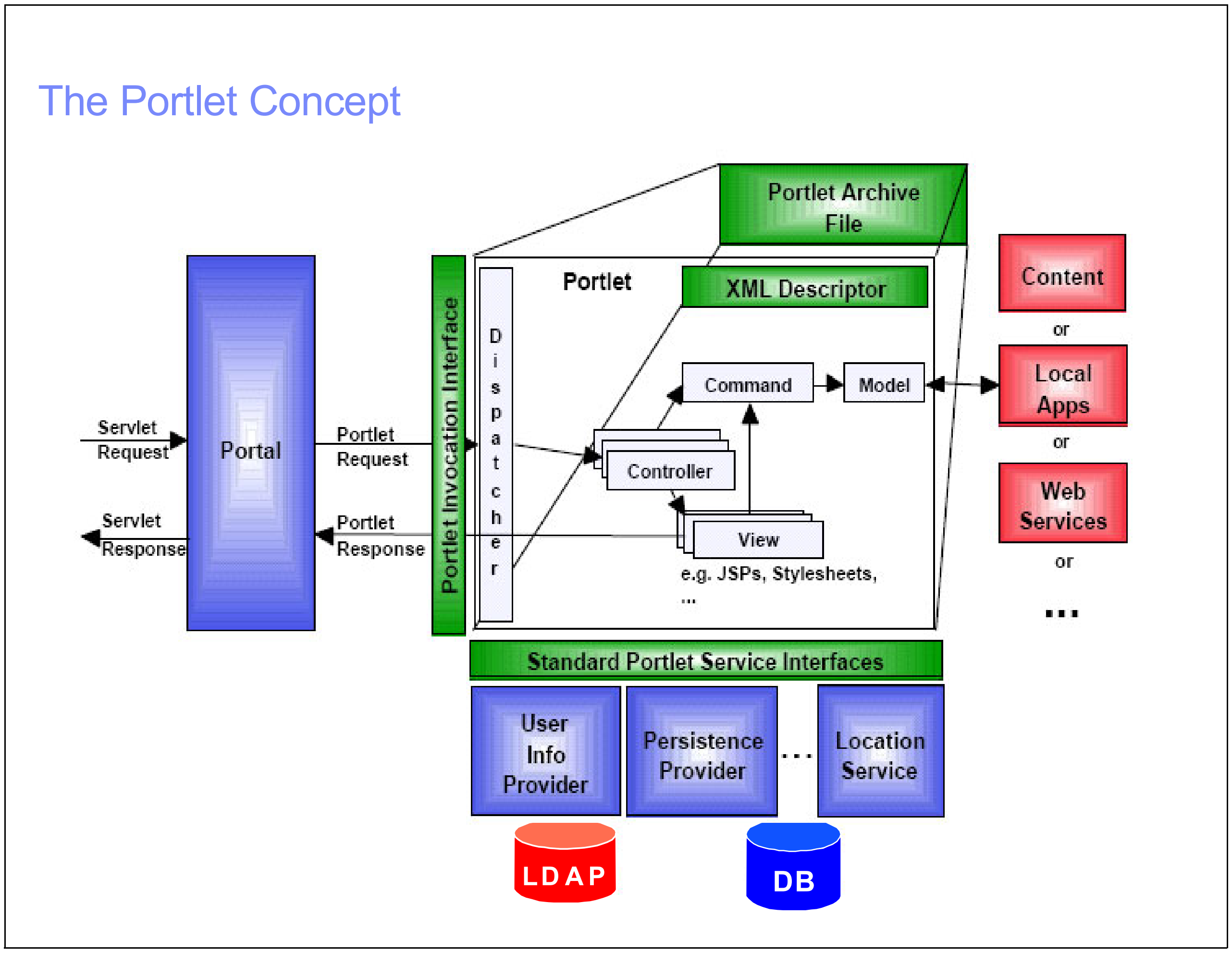


Figure 4-4 The portlet concept

While portlets must implement the invocation methods required by the Portlet API, internally they may be implemented differently. A pattern that has proven very suitable for portlet programming is the *Model-View-Controller* pattern. It separates the portlet functionality into a controller receiving incoming requests, invoking commands operating on a model that encapsulates application data and logic and finally calling views for presentation of the results.

Portlets have access to portal related functions and data through *Portlet Service Interfaces*. These interfaces provide portlets with functions including access to user profile information, persistent per-portlet instance data, action handling, and so on. Apart from portal specific functions, portlets can use all the J2EE services that are available to servlets as well as vendor-provided connectors to access back-end data and applications or even services in the Internet.

For easier deployment, portlets can be grouped in Portlet Applications packaged into Portlet Archive files containing a deployment descriptor, Java classes, jar files, and resources.

4.1.6 Portlet container

A portlet container runs portlets and provides them with the required runtime environment. A portlet container manages the portlet life cycle. It also provides persistent storage for portlet preferences. A portlet container receives requests from the portal to execute requests on the hosted portlets. A portlet container is not responsible for aggregating the content produced by the portlets. It is the responsibility of the portal to handle the aggregation. A portal and a portlet container can be built together as a single component of an application suite or as two separate components of a portal application.

4.1.7 Portlet life cycle and request processing

The basic portlet life cycle in the Portlet API is:

- *init* - to initialize the portlet and put the portlet into service.
- *handle requests* - process different kinds of action and render requests.
- *destroy* - to put portlet out of service.

The portlet receives requests based on the user interaction with the portlet or portal page. The request processing is divided into two major phases:

- *Action processing*

  A click on an action link in the markup of a portlet triggers an action call for this portlet. The action processing must be finished before any rendering of the portlets on the page is started. In the action phase the portlet has the ability to change state.

- *Rendering content*

  In the render phase the portlet produces its markup to be sent back to the client. Rendering should not change any state, allowing a page re-fresh without modifying the portlet state. The rendering of all portlets on a page can be performed in parallel.

4.2 The standardization of portlets (Java standardization request - JSR-168)

With the emergence of an increasing number of enterprise portals, a variety of different APIs for portal components (portlets), has been created by different vendors. The variety of incompatible interfaces creates problems for application providers, portal customers and portal server vendors. To overcome these problems, the Java Portlet Specification (JSR 168) standard will provide interoperability between portlets and portals. JSR 168 by the Java Community Process (JCP), provides a standard for interoperability between portlets and portals.

The JSR 168 was co-leaded between IBM and Sun and had a large expert group that helped to create the final version now available. This expert group consisted of Apache Software Foundation, Art Technology Group Inc.(ATG), BEA, Boeing, Borland, Citrix Systems, Fujitsu, Hitachi, IBM, Novell, Oracle, SAP, SAS Institute, Sun, Sybase, Tibco, Vignette. More details about this JSR can be found at:

`http://jcp.org/en/jsr/detail?id=168`

Portlets written to this portlet API can be deployed on any JSR 168 compliant portal and run out-of-the-box. The portlet architecture is an extension to the Java Servlet architecture. Therefore, many aspects of portlet development are common to typical Web application development. However, the unique aspects of a portal add complexity to the application model, such as: multiple portlets per Web page, portlet URL addressing, portlet page flow control, user interface rendering restrictions and Inter-portlet communication.

4.2.1 JSR 168 portlet modes

Portlets will perform different tasks and create different output depending on the function they are currently performing. Modes will allow the portlets to provide different function for the task that is required. JSR 168 supports three modes, *View, Edit*, and *Help*. JSR 168 also supports *custom* modes. Figure 4-5 shows a sample of JSR 168 portlet.

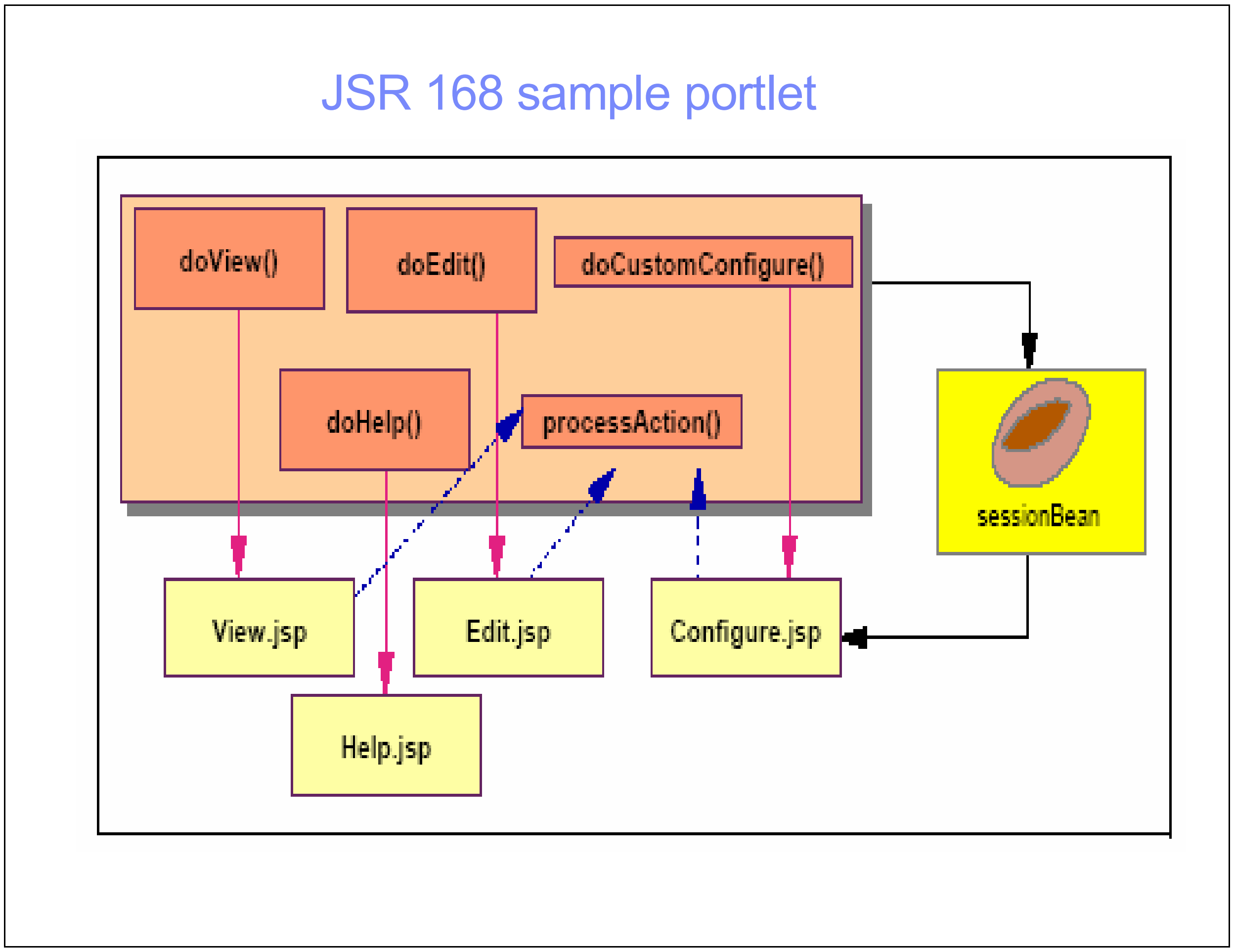


Figure 4-5 JSR 168 sample portlet

Portlet can change the mode programmatically in the *processAction* method. You can also specify the mode when creating an action or render link.

View

The *View* mode is used for displaying content reflecting the current state of the portlet. *View* mode may included multiple screens the user can navigate. The *doView()* method of the *GenericPortlet* class is invoked for this mode. All portlets are required to support the View mode.

Edit

The *Edit* mode is used for customizing the behavior of the portlet by modifying the *PortletPreferences* object. As with *View* mode the *Edit* mode may contain multiple screens for navigation. The *doEdit()* method of the *GenericPortlet* class is invoked for this mode. Portlets are not required to support the Edit mode.

Help

Help should be used to provide the user with information about the portlet. This could be generic or it could provide context-sensitive help. The *doHelp()* method of the *GenericPortlet* class is invoked for this mode. Portlets are not required to support the *Help* mode.

4.2.2 JSR 168 specific concepts

This section covers concepts like *Persistent state*, *Transient state and Navigational state*. Also we will indicate that are significantly different in the JSR 168 compared to the IBM Portlet API for portlet and servlet relationship.

Persistent state

A portlet can access two different types of persistent data: initialization parameters and portlet preferences. Initialization parameters are read-only data which you specify in the portlet deployment descriptor to define settings that are the same for all portlet entities created from this portlet description. You can use them to specify basic portlet parameters, such as the names of the JSPs that render the output.

Transient state

A portlet has access to two different kinds of transient state: session state and navigational state. The session state is available to the portlet for each user. The session concept is based on the Http Session defined for Web applications. Because portlet applications are Web applications, they use the same session as servlets. To allow portlets to store temporary data private to a portlet entity, the default session scope is the portlet scope. In portlet scope, the portlet can store information, needed across user requests, that are specific to a portlet entity. Attributes stored with portlet scope are prefixed in the session by the portlet container to avoid two portlets (or two entities of the same portlet definition) overwriting each other's settings.

The second scope is the Web application session scope, in which every component of the Web application can access the information. The information can be used to share transient state among different components of the same Web application (such as between portlets, or between a portlet and a servlet).

Navigational state

Navigational state defines how the current view of the portlet is rendered and is specific to the portlet window in which the portlet is rendered. Navigational state can consist of data such as the portlet mode, window state, a sub-screen ID, and other data. The navigational state is represented in the JSR 168 portlet API through the portlet mode, window state, and the render parameters. The portlet receives the render parameter for each render call. The data can only be changed in an action, or by clicking on a render link with new render parameters.

The different in IBM portlet API and JSR 168 for portlet and servlet

In the IBM Portlet API, portlets extend servlets and all the major interfaces (such as *Request, Response*, and *Session*) extend the corresponding servlet interfaces. In JSR 168, portlets are separate components that may be wrapped as servlets, but they do not need to be servlets.

This separation was made to enable the different behavior and capabilities of portlets. Because a portlet is not a servlet, in JSR 168 it is possible to define a clear programming interface and behavior for portlets. However, in order to reuse as much as possible of the existing servlet infrastructure, JSR 168 leverages functionality provided by the servlet specification wherever possible, including:

– Deployment

- – Classloading
- – Web applications
- – Web application life cycle management
- – Session management
- – Request dispatching

Many concepts and parts of the portlet API have been modeled similar to the servlet API.

4.2.3 JSR 168 and Web Service for Remote Portlets (WSRP)

JSR 168 establishes a standard API for creating portlets, the integration component between applications and portals that enables delivery of an application through a portal. Without this standard, each version of an application will need its own portlet API, and each of the various portals required that these portlets be specifically tailored for implementation through that portal. This has increased portlet developer time, effort, and costs with the net effect that fewer applications have been made available through fewer portals to the detriment of the end-users, ISVs, developers, and portal vendors. Now, when portal developers and portal vendors adhere to this standard, applications can be delivered through any portal almost immediately.

WSRP specification is a product of the Organization for the Advancement of Structured Information Standards (OASIS), which is a consortium that facilitates the adoption of technical standards. WSRP defines a Web service interface for accessing and interacting with interactive presentation-oriented Web services.

WSRP is built upon existing Web services standards like SOAP, WSDL, and UDDI. Figure 4-6 shows how WSRP fits into the technology stack.

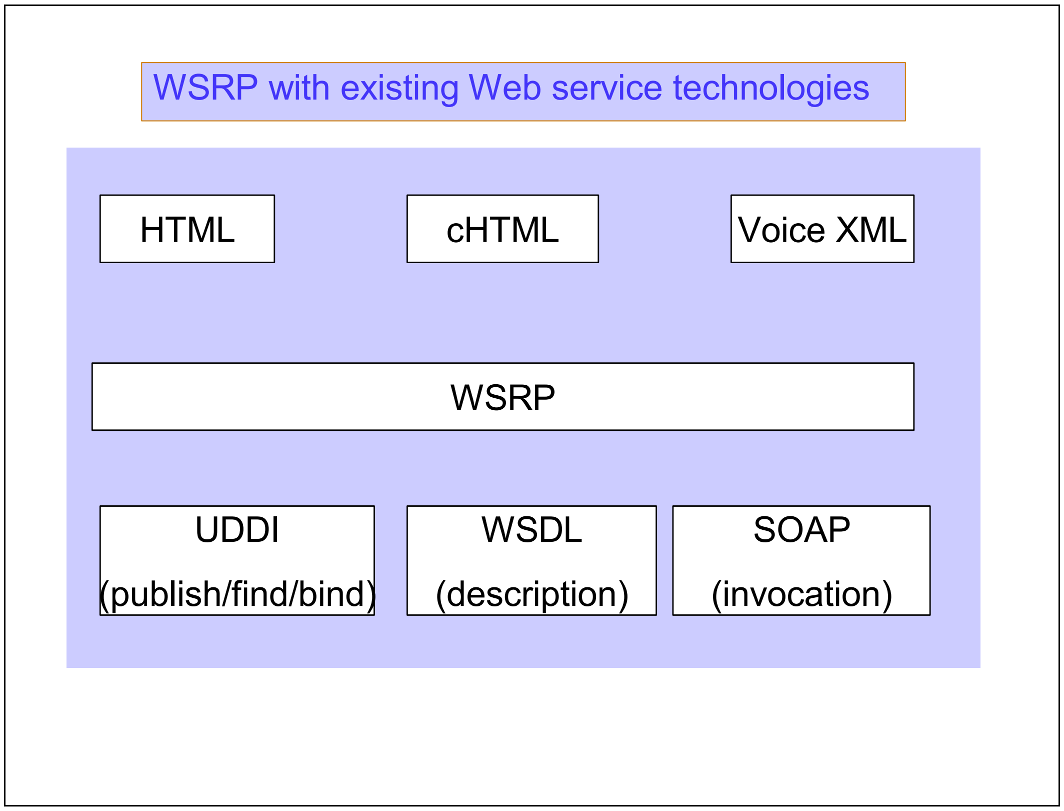


Figure 4-6 WSRP with existing Web service technologies

There is a mapping of all concepts between JSR and WSRP. This allows implementing JSR 168 portlet containers that can be accessed via WSRP and therefore expose JSR 168 portlets as WSRP services. Figure 4-7 shows how those two work together.

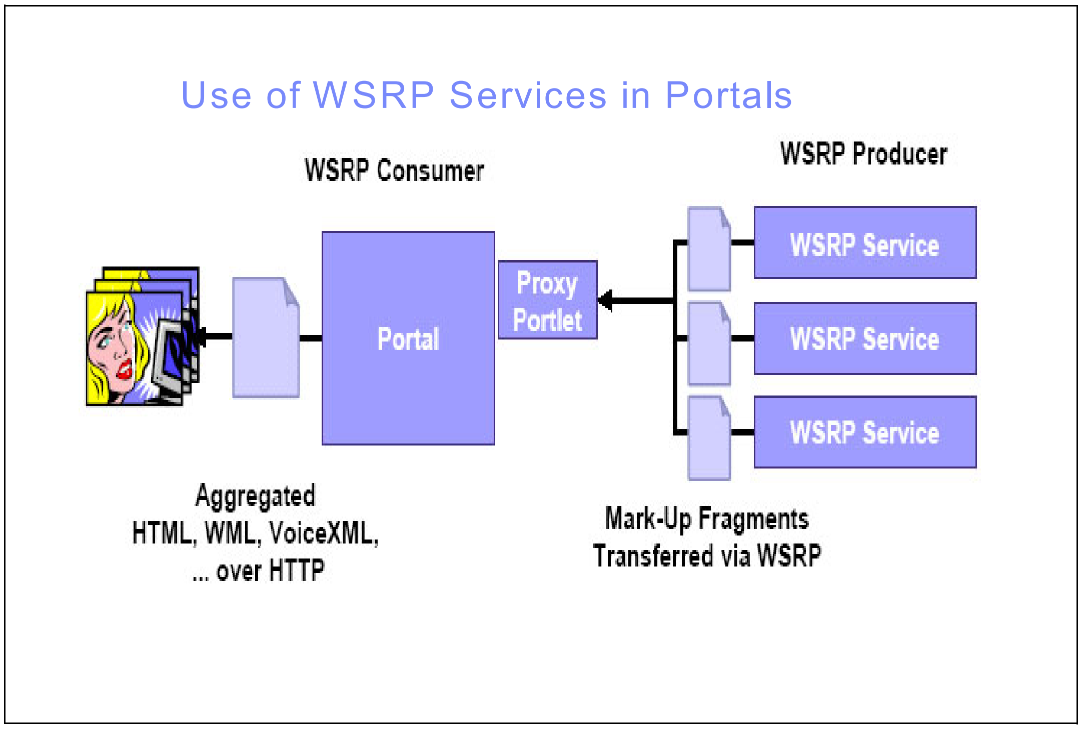

Use of WSRP Services in Portals
WSRP Consumer
WSRP Producer
Portal
Proxy Portlet
WSRP Service
WSRP Service
WSRP Service
Aggregated HTML, WML, VoiceXML, ... over HTTP
Mark-Up Fragments Transferred via WSRP

Figure 4-7 Use of WSRP Services in Portals

Figure 4-8 shows how the portal shares portlets as WSRP services.

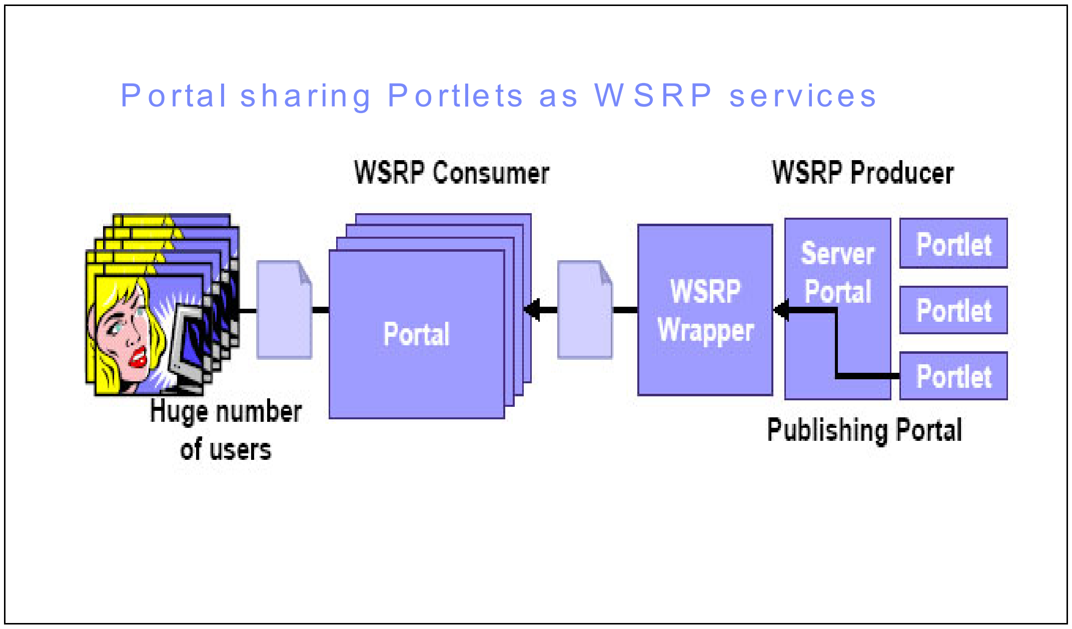


Figure 4-8 Portal sharing Portlets as WSRP services

4.2.4 Portlet development guidelines with JSR 168

As with any coding practice, there are exceptions to every rule. These guidelines are intended to help you produce best-of-breed JSR 168 portlets for the WebSphere Portal environment. The guidelines must ultimately be adapted to your development environment and product architecture.

- Do not use instance variables.

  Portlets, like servlets, exist as a singleton instance within the server's JVM™. Therefore, a single memory image of the portlet services all requests, and it must be thread-safe. Data stored in instance variables will be accessible across requests and across users and can collide with other requests. Data must be passed to internal methods as parameters.

- Pass data to the view JSP as a bean in the request object.

  Use the *RenderRequest* object to pass data to the view for rendering, so that when the request is complete, the data falls out of scope and is cleaned up. Passing it as a bean lets the JSP simply refer to the data as properties on the bean using intrinsic functions in the JSP syntax.

- Adopt good code documentation habits.

  While commenting of code is not required for the functionality of the portlet, it is essential for its maintenance. A portlet's maintenance can change hands over time, and well-written portlets serve as models for other portlets. Therefore, someone else must understand what you wrote. Well documented code implies more than simply inserting comments; it also implies good naming practices for Java resources.

- Follow *Struts* design guidelines for *Struts* portlets.

  Struts is an emerging standard for *Model-View-Controlle*r Web application design and implementation. The *Struts* framework has been adapted to the portlet development environment and is available as a plug-in (stand-alone WAR file) for WebSphere Portal. This support lets you incorporate *Struts* into your portlet application without having to work through the details of including Struts support in WebSphere Portal.

- Categorize the portlet state early in the design phase.

  As mentioned in the introduction, the JSR 168 supports different kind of states. The portlet programmer should very carefully and early on decide the category of information for each state.

  The categories are: *navigational state, session state, persistent state*.

  – Use *navigational state* for all view-related data that will let the user navigate forwards and backwards using the browser buttons. The scope of navigational state information is the current request. Examples of navigational state information include the current selected article in a news portlet, and the current selected stock quote for which the portlet should render more details.

  – Use *session state* for all information which is relevant for the duration of the user session. Do not use session state as caching store. An example of session state information is the content of a shopping cart.

  – *Persistent state* has a life-time beyond the current user session. Use it to store customization data and user-specific personalization data. Examples include the server to retrieve the stock quotes from, the default list of stock quotes to display, or the news topics of interest for a specific user.

- Internationalize the portlets using resource bundles.

  Use a resource bundle per portlet to internationalize the portlet output, and declare this resource bundle in the portlet deployment descriptor.

4.2.5 Building JSR 168 portlets with Rational Application Developer (RAD)

Creating a JSR 168 portlet project and generating a portlet are the first steps to build a JSR 168 portlet. RAD provides a wizard to generate both a JSR 168 portlet project and portlet. To create a sample JSR 168 portlet project and portlet:

1. Start RAD and open Web Perspective.
2. Select File > New > Project > Portlet Project (JSR 168).
3. On the Portlet Project (JSR 168) panel, specify a name and location for the new portlet project. Be sure to check the Create a portlet box so the following steps display. Click Next.
4. On the Portlet Type panel, select Basic portlet (JSR 168). Click Next.
5. On the Features panel, click Next.
6. On the Portlet Settings panel, enter values for Define a Portlet Name and the Portlet Display Name.
7. Check the Change code generation options box to add a package prefix and a class prefix. Click Next.
8. On the Action and Preferences panel, make sure the box for Portlet action handling is checked. Click Finish

When portlet generation has completed, a JavaServer Pages (JSP) file, several portlet class files, a Web.xml file, and a portlet.xml file will be created. Selecting the action handling indicates generation of an additional portlet method, public void processAction(ActionRequest request, ActionResponse response), a portlet JSP tag <portlet:actionURL/>, a portlet session bean class, and a getter method to retrieve this class within the portlet.

You must manually create three more regular Java class files, using the following information.

A back-end bean

Contains two methods:

public TaskManagerDelegate getTaskManager() retrieves the *TaskManagerDelegate object*

public SampleTaskBean getInputMsg(TaskManagerDelegate taskManager, String taskID) first retrieves the input message of the task identified by the *taskID*, then obtains the input message fields and stores them into the data bean.

A data bean

Contains the attributes includes the getter and setter methods for these attributes. This bean provides the input message fields to the view bean.

A view bean

Contains the above attributes and a *String variable imageURL*, *boolean isApproved* and *String comments* variables. The *imageURL* stores the fully qualified name of the image file, which consists of location and name of the image file. The view bean contains *getter* and *setter* methods of its attributes. This bean is used to display input message fields, *boolean isApproved* and *String comments*, and the image file on the browser.

4.3 Portals and SOA

This section provides an overview and some samples for creating a sample portlet that will work as a Web Service client to interact with a Web Service. The Web Service client portlet is created using the wizard provided by the WebSphere Portal Toolkit. The samples in this section will allow you to understand the techniques used to develop portlets that retrieve information using Web Services.

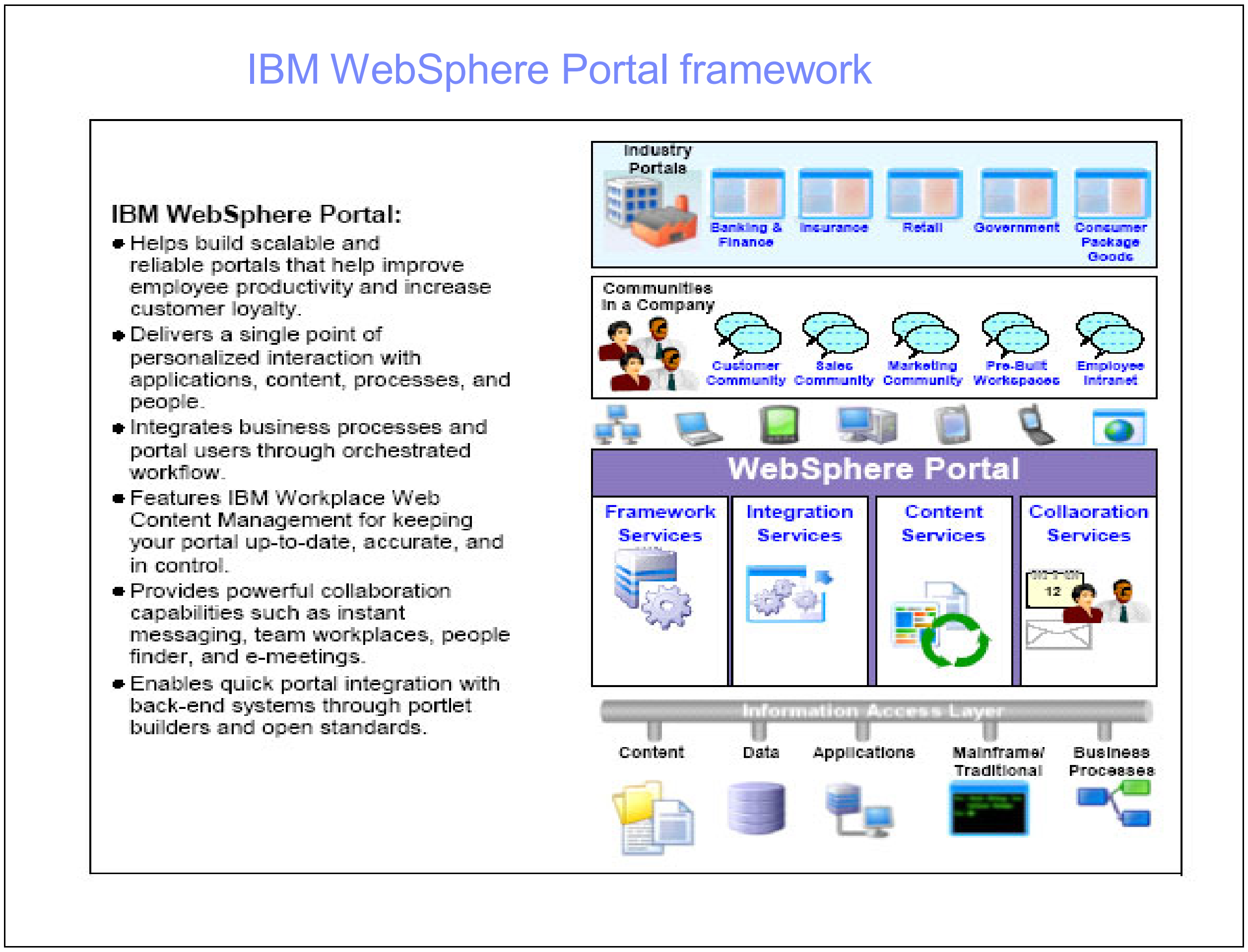
In addition to contextual integration capabilities, portals can provide rich programming frameworks for building user interfaces for component-oriented applications in SOA is an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services. SOA provides means to integrate and manage these different services. For more information, refer to the following Web page:

http://www.ibm.com/SOA

Portals provide first-class user interface (UI) support in service-oriented architectures. Portlets, their basic building block, let developers focus on unique aspects of their application, while the middleware handles common functions for life cycle, per-user customization, aggregation, and integration with other components. In addition, portals might provide valuable service functions such as security, search, collaboration, and workflow. Portals provide the ability to aggregate and integrate the UI in a similar way SOA run times can combine and integrate services. Component UIs are aggregated into larger, higher value UIs, giving users a single view of IT services with a single UI to master. Applications originally designed separately can be integrated (aggregation and context) together to enable new function. The portal model allows for improved agility for on-demand businesses. Portal administrators become application integrators who create new applications for their users without programming: by defining new pages, adding portlets to them, connecting the portlets together in context, and setting entitlements. With portal technologies, end users can become their own application assemblers by customizing their portal-based workspaces. There are many reasons why a portal would benefit your organization, for example:

- Control information glut
- Improve cycle times
- Empower knowledge workers
- Reduce it complexity
- Enhance partner and supplier communication
- Streamline™ processes

WebSphere Portal (Portal for short) supports multiple industry portals and various communities within a company. Portal consists of four basic services: Framework, Integration, Content, and Collaboration. Figure 4-9 on page 85 illustrates the IBM WebSphere Portal framework.


IBM WebSphere Portal framework

IBM WebSphere Portal:

- Helps build scalable and reliable portals that help improve employee productivity and increase customer loyalty.
- Delivers a single point of personalized interaction with applications, content, processes, and people.
- Integrates business processes and portal users through orchestrated workflow.
- Features IBM Workplace Web Content Management for keeping your portal up-to-date, accurate, and in control.
- Provides powerful collaboration capabilities such as instant messaging, team workplaces, people finder, and e-meetings.
- Enables quick portal integration with back-end systems through portlet builders and open standards.

Figure 4-9 IBM WebSphere Portal framework

4.3.1 User access to service

Service-oriented architecture specifies the use of interfaces to define encapsulated, reusable business function: in part, those interfaces identify a business function and specify the data required to interact with it. This is precisely the purpose of many application user interfaces: to enable users to identify a function, collect the data required to invoke it, and return the outcome to the user.

This correspondence has led to several interesting patterns emerging in providing user access to services and Web services:

Portal technologies, such as WebSphere Portal Server, offer the capability to automatically present some Web services as portlets; however, this is often dependent on the addition of specific display-related information to the Web services description.

The (Organization for the Advancement of Structured Information Standards) OASIS Remote Portlet Web Services specification provides an open standards means to exposed Web services in a manner that is suitable for display by portal technology, but the standard is relatively recent so full product support may take some time to emerge.

The World Wide Web Consortium (W3C) recently published the xForms specification for device-independent description of the data model for user interfaces. The content of xForms descriptions bears several similarities with that of Web Services Description Language

(WSDL) descriptions of the data that is required by and returned from a service. If a service interface can be transformed manually or programmatically into an xForms definition, then xForms UI generators can be used to generate a variety of Web-based, desktop, or other UIs. The xForms specification can be found at:

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

In the SOA approach, the environments that host components are abstracted as containers, which provide well-known sets of infrastructure services. From a UI perspective, the three major containers for hosting client-side UI components are the:

- Basic Web browser.
- Web browser augmented with JavaScript™ and dynamic HTML.
- IBM Workplace Client Technology™ - the Eclipse-rich client plus native IBM WebSphere Application Server client support.

These containers can be augmented by supporting technologies such as servlets, JavaServer Pages (JSP), and JSP Tags; Struts for page sequencing; JavaServer Faces (JSF) for advanced page composition; and portlets to combine views of multiple applications on the same page.

Frameworks for UI development

UI development frameworks can simplify the creation of complex user-facing applications. The following UI frameworks are often used to create UI components:

- Struts, with the largest developer community and exceptional tools support, is an Apache open source project predating the Java Portlet Specification, JSR 168 (see Resources for a link to the Struts Web site). Struts is a multi-page MVC framework for server-based UI development using the servlet/JSP paradigm. A special version of the Struts, Version 1.1 library supports JSR 168 portlets on IBM WebSphere Portal.
- JavaServer Faces (JSF) is an MVC realization for Java Web applications and builds incrementally on prior technologies. It is well suited for portlet development, offering portlets and servlets, state handling, validation, and events. A JSF page has one or more local models that interact with UI controls on the page. These controls render UI properties into outputs, and sophisticated logic ensures their presentation is at the "right" place. The client-side model can be wired into the Enterprise Service Bus to send and receive events.
- Java Widget Library (JWL), an extended widget set usable by portal and portlet programmers, adds JavaScript client-side processing to JSF and will be supported by IBM Rational Suite® Development Studio. Updating the view locally on the client saves round trips to the server, shortens response time by orders of magnitude, and dramatically improves the user experience.

Portals provide first-class UI support. In the portal architecture, a portlet (typically developed using one of the above-mentioned UI frameworks) is the basic building block. This architecture lets developers focus on unique aspects of their application and delegate common functions for life cycle, per-user customization, aggregation, and integration with other components to the middleware.

The following sections describe portlet components for individual services and portals as a service aggregation mechanism.

Portlets for the service-oriented UI

The portlet component implements a standardized service interface and protocol. The Java Portlet Specification and Web Services for Remote Portlet (WSRP) standard define this interface for Java and Web services, respectively. The two standards are similar enough that portlets written to either interface are interchangeable if the proper containers or proxies are present.

Java portlet example

Every Java portlet implements the portlet interface or extends a class that implements it. This interface defines the service contract between the portlet and its container and defines the portlet life cycle:

- Initializing the portlet and putting it into service (*init* method)
- Request handling (*processAction* and *render* methods)
- Taking the portlet out of service (*destroy* method)

During request handling, the portlet container calls the portlet's:

- *processAction* method to notify the portlet of a user action. Only one user-based action per client request is triggered. The portlet can issue a redirect, change its portlet mode or window state, or modify its state.
- *render* method to request a markup fragment.

Portlets can also invoke further services to perform desired functions. Example 4-1 uses a Web service to retrieve and display stock quotes for a specific user.

Example 4-1 Stock quote portlet code sample

```
public class StockQuotePortlet extends GenericPortlet {
private ServiceManager serviceManager;

public void init(PortletConfig config) throws PortletException {
   serviceManager = new ServiceManager();
}

public void doView(RenderRequest request, RenderResponse response) throws
PortletException, IOException {
   response.setContentType("text/html");

// invoke autogenerated wrapper to locate service
   NetXmethodsServicesStockquoteStockQuoteServiceLocator loc =
      new NetXmethodsServicesStockquoteStockQuoteServiceLocator();
   NetXmethodsServicesStockquoteStockQuotePortType port =
      loc.getNetXmethodsServicesStockquoteStockQuotePort();

   // loop through all stock quotes the user is interested in
      PortletPreferences prefs = request.getPreferences();
      Iterator quoteKeys = prefs.getMap().keys().iterator();
      String key;
      Float quote;
      StockBean quoteBean = new StockBean();
      while ( quoteKeys.hasNext() ) {
         key = quoteKeys.next();
         quote =  port.getQuote (key);
         quoteBean.add(key, quote);
      }

      request.setAttribute("StockQuoteBean", quoteBean);
      // render stock quotes using a JSP
PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher("jsp/View.jsp");
        rd.include(request,response);

   }
}
```

This section illustrated how you can implement a UI service using the Java Portlet Specification, and how your portlet can invoke additional Web services. The next section shows how to publish your UI as a Web service using WSRP.

Web Services for Remote Portlets

WSRP is the standard for remote rendering of portlets, enabling a portal to aggregate content from multiple sources. WSRP extends the integration capabilities of Web services to presentation-oriented components, and exposes the view layer to sharing across platforms, implementation languages, and vendors. Content and application provider services can be discovered and plugged into standards-compliant applications without any extra programming effort

Typical Web services employ a remote presentation paradigm, meaning all view logic executes on the client, whereas the application logic and the data layer (controller and model) reside on the server. By contrast, WSRP splits presentation between the client and the server in a distributed paradigm.

In order to allow for dynamic integration of portlets in portals without installing a portlet archive file with the entire portlet code locally, portlets themselves have to be provided as Web services. This requires a Remote Portlet Web Service Interface description in WSDL.

The WSDL description defines a common set of methods for all remote portlets and the required parameters as well as the return values, corresponding to the Portlet API. This means that remote portlet services do not have to be implemented in Java, they could as well be implemented in other languages.

Web service providers who want to publish remote portlet Web services must publish appropriate entries to a UDDI directory, referencing the Remote Portlet Web Services Interface WSDL description.

Once a remote portlet has been published, portal administrators can use their portal administration tools to search the UDDI directory for Web services that implement the Remote Portlet Web Services Interface and pre-select some of the matching portlet Web services for use in their portal by adding them to the portal's portlet registry (see Figure 4-10 on page 89).

Once the portlets are in the registry, users can select them to be displayed on their personal pages. Alternatively, portals may be set up in a way that allows portal users themselves to browse the directory for portlet Web services and add references to remote portlets to their personal pages.

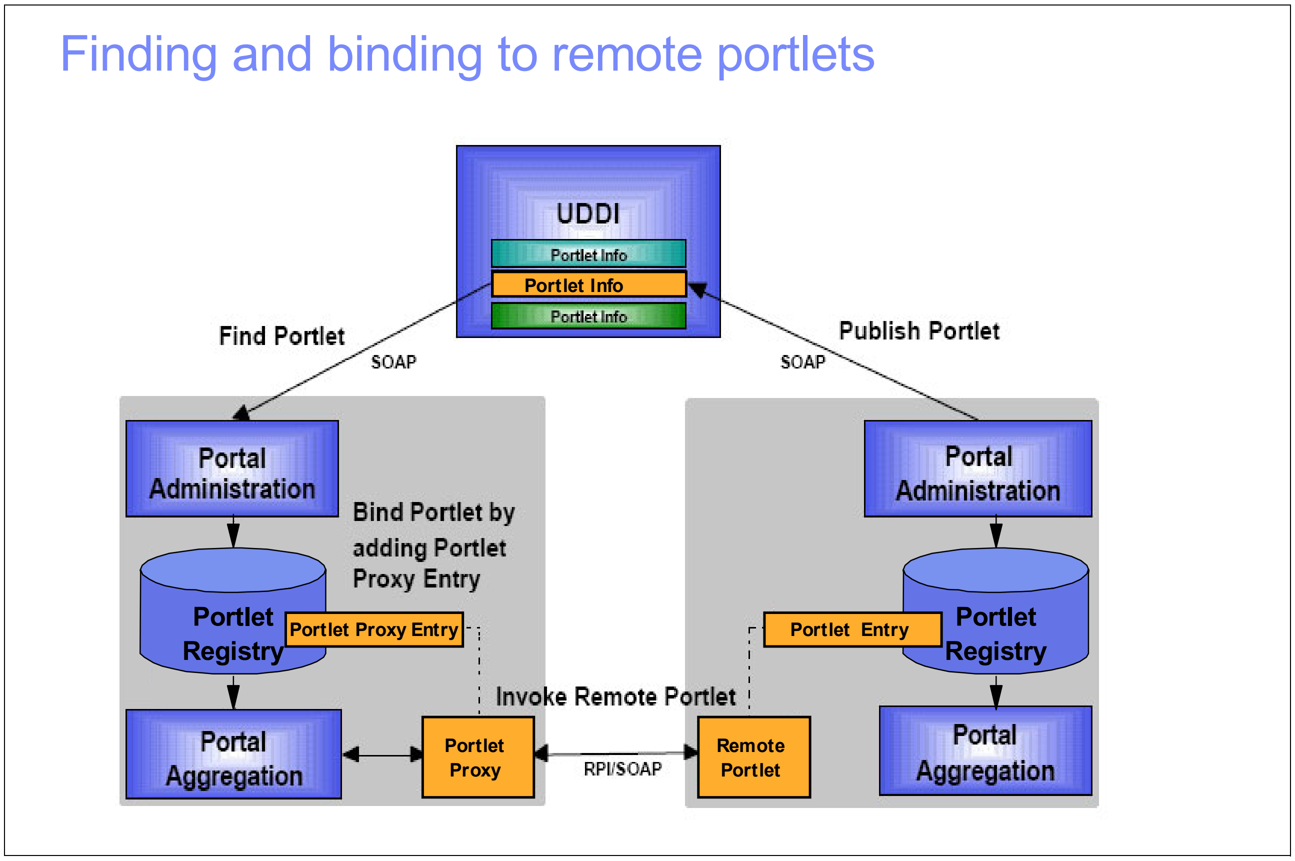


Figure 4-10 Finding and binding to remote portlets

When a page that references a remote portlet gets rendered, the portal uses a portlet proxy to invoke the remote portlet Web service through the Remote Portlet Invocation (RPI) protocol (see Figure 4-11 on page 90). The portlet invokes the portlet proxy exactly like it would invoke a local portlet, passing *PortletRequest* and *PortletResponse* objects. The portlet proxy internally invokes a SOAP proxy to marshals all parameters into a SOAP request and sends it to the remote server hosting the portlet Web service. The SOAP wrapper on the Web service side unmarshals all information in the incoming request and calls on the remote portlet.

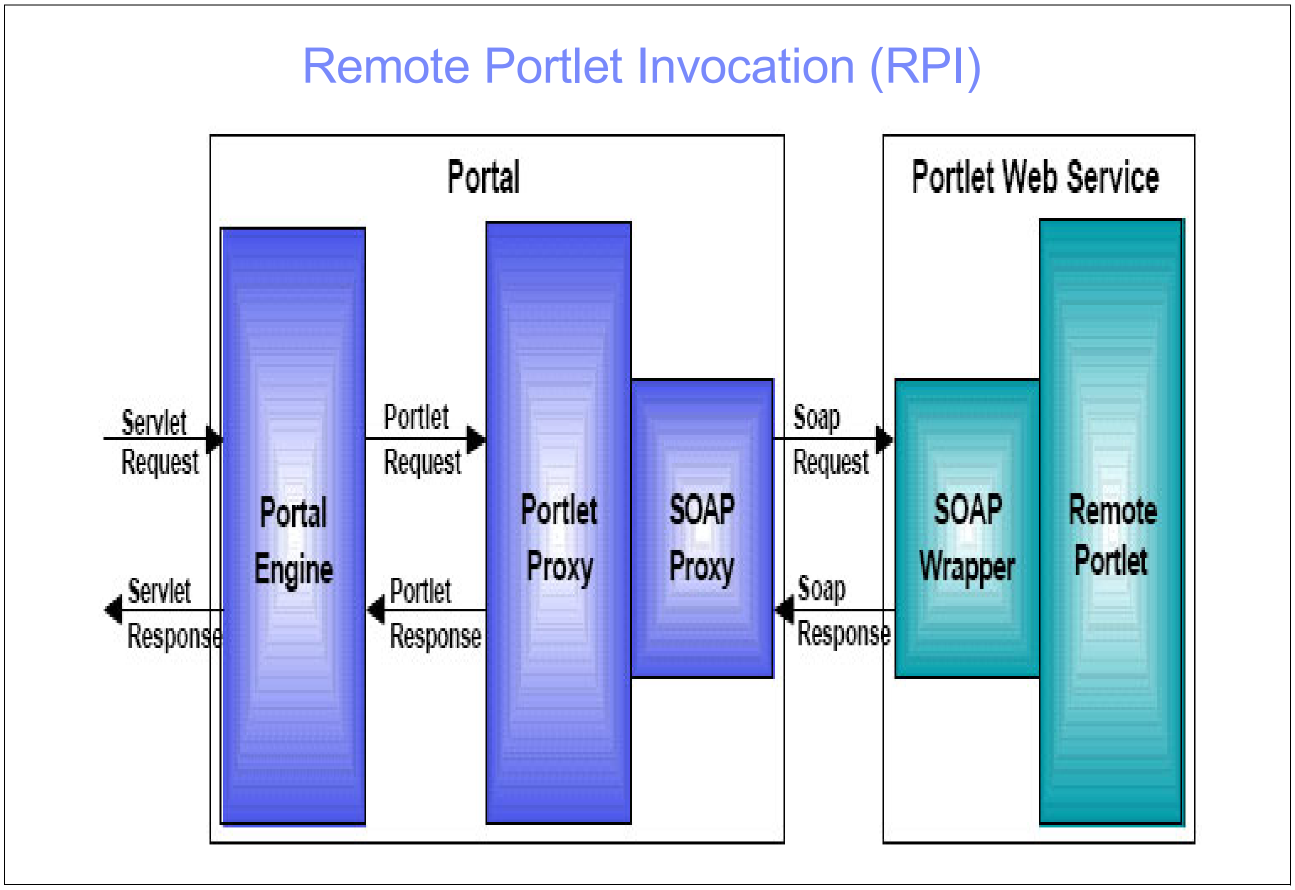


Figure 4-11 Remote Portlet Invocation (RPI)

For the remote portlet, it is transparent whether it is invoked directly by a portal engine or indirectly through the Web service interface. In each case, it processes the input parameters and returns a *PortletResponse* object.

The SOAP wrapper marshals the response into a SOAP response and sends it back as the reply to the SOAP proxy that in turn unmarshals the response for the portlet proxy that finally returns a *PortletResponse* object to the portal engine that initiated the request.

WSRP and Web service example

Example 4-2 shows a WSRP *getMarkup* request issued through Simple Object Access Protocol (SOAP) from a WSRP consumer.

Example 4-2 WSRP getMarkup request issued through SOAP

```
<?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>
   <getMarkup xmlns="urn:oasis:names:tc:wsrp:v1:types">
      <registrationContext>
         <registrationHandle>192.168.66.57_1096235652731_0</registrationHandle>
      </registrationContext>
      <portletContext>
         <portletHandle>0.1</portletHandle>
      </portletContext>
      <runtimeContext>
         <userAuthentication>wsrp:none</userAuthentication>
         <portletInstanceKey>ProxyTest_row1_col1_p1</portletInstanceKey>
         <namespacePrefix>Pluto_ProxyTest_row1_col1_p1_</namespacePrefix>
      </runtimeContext>
      <userContext>
         <userContextKey>dummyUserContextKey</userContextKey>
      </userContext>
      <markupParams>
         <secureClientCommunication>false</secureClientCommunication>
         <locales>en</locales>
         <locales>de</locales>
         <mimeTypes>text/html</mimeTypes>
         <mode>wsrp:view</mode>
         <windowState>wsrp:normal</windowState>
         <clientData>
         <userAgent>WSRP4J Proxy Portlet</userAgent>
         </clientData>
         <markupCharacterSets>UTF-8</markupCharacterSets>
         <validNewModeswsrp:view</validNewModes>
         <validNewModes>wsrp:help</validNewModes>
         <validNewModes>wsrp:edit</validNewModes>
         <validNewWindowStates>wsrp:normal</validNewWindowStates>>
         <validNewWindowStates>wsrp:maximized</validNewWindowStates>
         <validNewWindowStates>wsrp:minimized</validNewWindowStates>
      </markupParams>
   </getMarkup>
</soapenv:Body>
</soapenv:Envelope>
```

The response of the WSRP producer to this request is an HTML fragment that the consumer (typically a portal) can aggregate into a full document, such as a portal page. Instead of deploying each application or portlet on every server that intends to use it, there are obvious advantages to sharing applications across network boundaries. WSRP enables:

- Easier administration - Instead of managing local deployments of pluggable components, portal administrators can browse a registry for WSRP services to offer. Users benefit from timely availability of new services and content integration on demand.
- Load distribution - Loads are distributed across multiple servers.
- Reduced infrastructure cost - Applications can share hosting infrastructure. For example, distributing just the presentation layer (through WSRP) of a back-end banking application preserves the application provider's secured computing environment, yet users can interact with the shared UI.
- Control over content presentation - Content and application providers can vastly expand their reach to new users as portals redistribute content.

Portals: dynamic aggregation for the service-oriented UI

A portal's view-layer integration of several backend services' UIs into a centrally-managed UI can unify the fractured IT infrastructure and give users a single view of IT services with a single UI to master. Applications originally designed separately can be wired together into composite applications, enabling new functions. For example, an e-mail portlet wired to a collaboration portlet could filter the inbox to display received e-mail only when the sender is online and available for a chat - a capability absent from both original applications.

A surprising consequence of the portal model is improved agility for On Demand Businesses. Administrators become application integrators who create new compound applications - without programming - by defining new pages, adding portlets to them, wiring the portlets together, and setting entitlements (access controls). A self-service portal lets users adapt their work environment to their unique needs. The portal architecture frees application developers to concentrate on building new business value.

In the future, portals may even be able to aggregate compound services and, thus, be able to aggregate UIs on a higher level. Portals may seamlessly integrate content from other portals, providing a horizontal, enterprise-wide integration of content.

4.3.2 Web service and portals

WebSphere Portal provides services for exposing and integrating portlets as remote portlets hosted on another portal platform via Web Services technology. The entire process of packaging and responding to a SOAP request is hidden from the developer and the administrator.

As Web services will become the predominant method for making information and applications programmatically available via the Internet, portals will need to allow for integration of Web services as data sources and as remote application components very soon. A typical example is a news portlet that allows the user to configure the news categories to track and then gets the news for these categories live from a Web service whenever it is displayed. In this case, the portlet code runs locally on the portal and uses the Web service to access information. Rendering is done by the local portlet while the Web service only provides the information to be rendered, for example as an XML document (see Figure 4-12 on page 93, search portlet and news portlet).

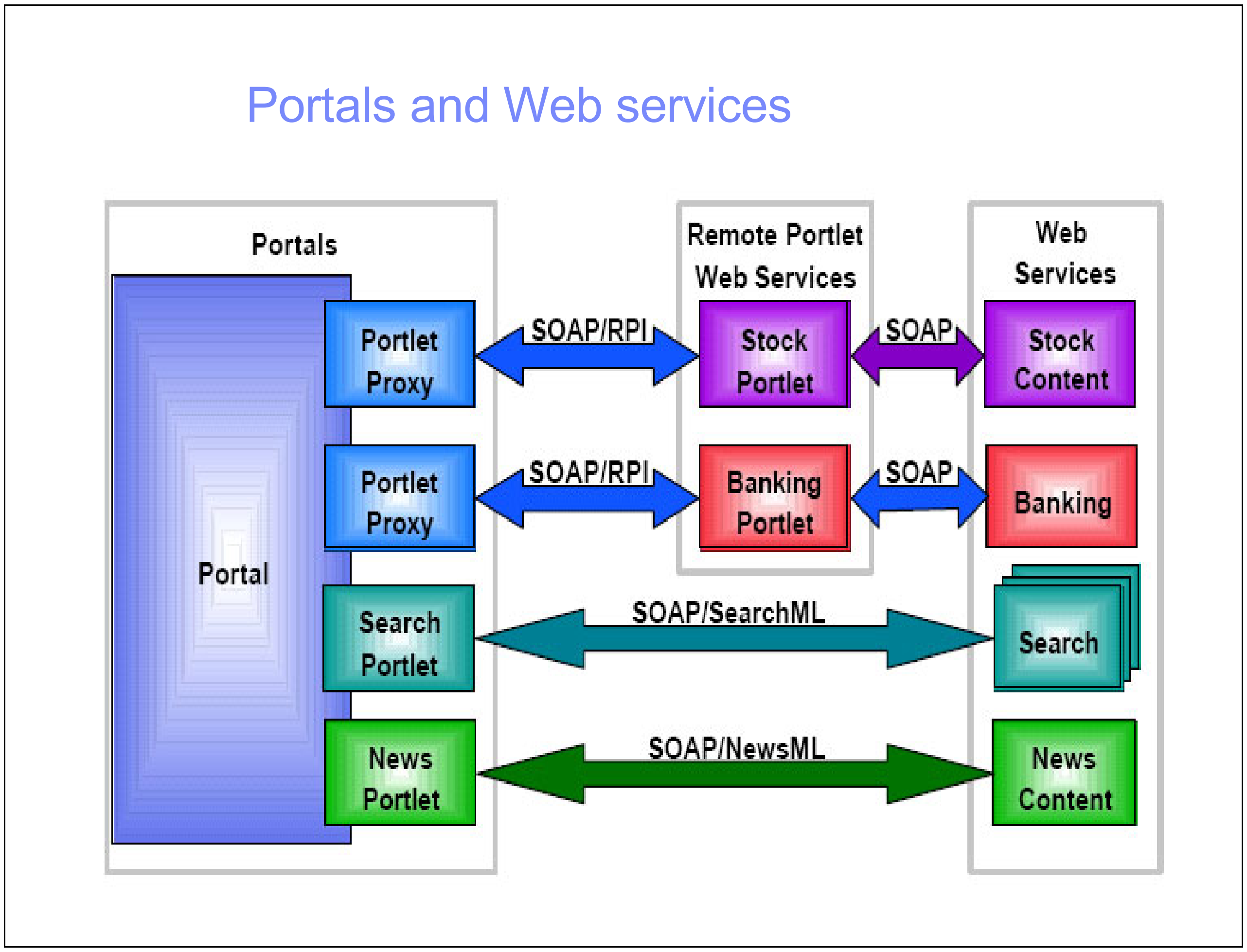


Figure 4-12 Portals and Web services

Another scenario for use of Web services by portals is sharing of portlets with other portals. In this scenario, a remote server, for example, another portal publishes portlets as remote portlet Web services in a UDDI directory. The portal can now find the remote portlet services in the directory and bind to them. As a result, the remote portlets become available for portal users without requiring local installation of portlet code on the portal itself (see Figure 4-12, portlet proxies for stocks and banking).

Figure 4-13 on page 94 shows how portlet applications can be easily integrated with existent Web Services without the need to write extra code with WebSphere Studio Site Developer tool. See more details in the redbook: *IBM WebSphere Portal V5 A Guide for Portlet Application Development,* SG24-6076.

Web services client portlet scenario

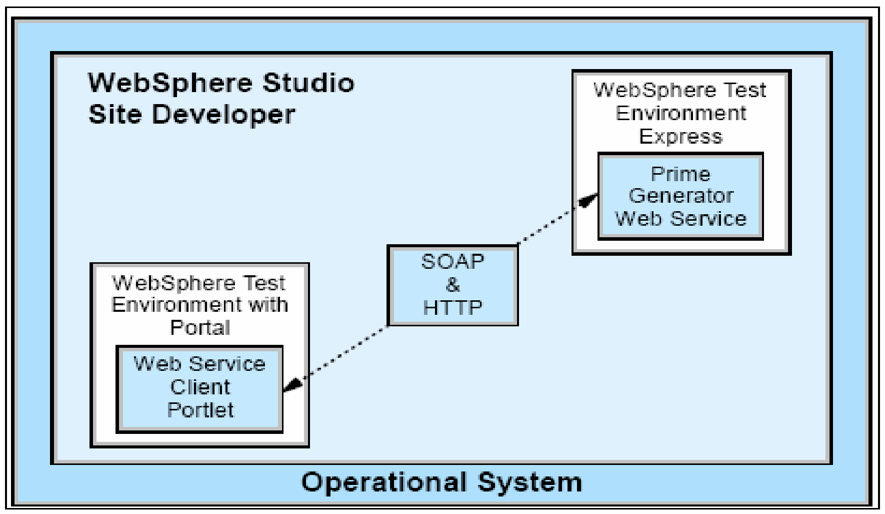

WebSphere Studio Site Developer
WebSphere Test Environment Express
Prime Generator Web Service
SOAP & HTTP
WebSphere Test Environment with Portal
Web Service Client Portlet
Operational System

Figure 4-13 Web Services client portlet scenario

For a portlet to be a source of data, programmers can use a custom JSP tag library to flag sharable data on their output pages. The tags require a data type to be specified as well as a specific value corresponding to an instance of this type. If you want to use wires source portlets, you must register properties by using a declarative or programmatic approach. Target portlets associate their actions with an input property which has been declared as an XML type. The actions are declared using WSDL, with a custom binding extension which specifies the mapping from the abstract action declaration to the actual action implementation. Associated with each action is a single input parameter described by an XML type and zero or more output parameters, each described by an XML type. Each input or output parameter encapsulates exactly one property. Example 4-3 on page 96 the target cooperative portlet Employee Details Portlet displays a list of employees working in the same department.

In the intranet it typically happens using database connections, LDAP connections, Java RMI, DCOM, CORBA, and so on. Over the Internet, in most cases the HTTP protocol is used to send requests to remote applications and receive results. Within short time, SOAP will be the primary communication mechanism for invocation of remote services by portlets and will incrementally replace the mechanisms listed above.

With SOAP and UDDI, communication between Web services and their clients and management of Web services in global and corporate directories is unified. This allows programmatic finding, binding and usage of Web services. Web services can be formally described using WSDL descriptions that can be used by appropriate tools to generate SOAP proxies for specific programming languages. Also, there are tools that can create Web services and WSDL descriptions from existing code.

Figure 4-14 on page 95 shows how a portlet that uses a Web service. When a portlet receives a request that requires invocation of a remote service, the portlet makes calls on a SOAP proxy object. The proxy takes the parameters, marshals them into a programming language-independent SOAP request, and sends this request to the remote Web service. The Web service has a SOAP wrapper that receives the SOAP request, unmarshals the parameters and invokes the local service implementation with these parameters. When the service returns the result, the SOAP wrapper marshals the result data into a programming-language independent SOAP response and sends it back to the SOAP proxy. The SOAP proxy finally unmarshals the result data and returns it to the calling portlet in the form of an appropriate object.

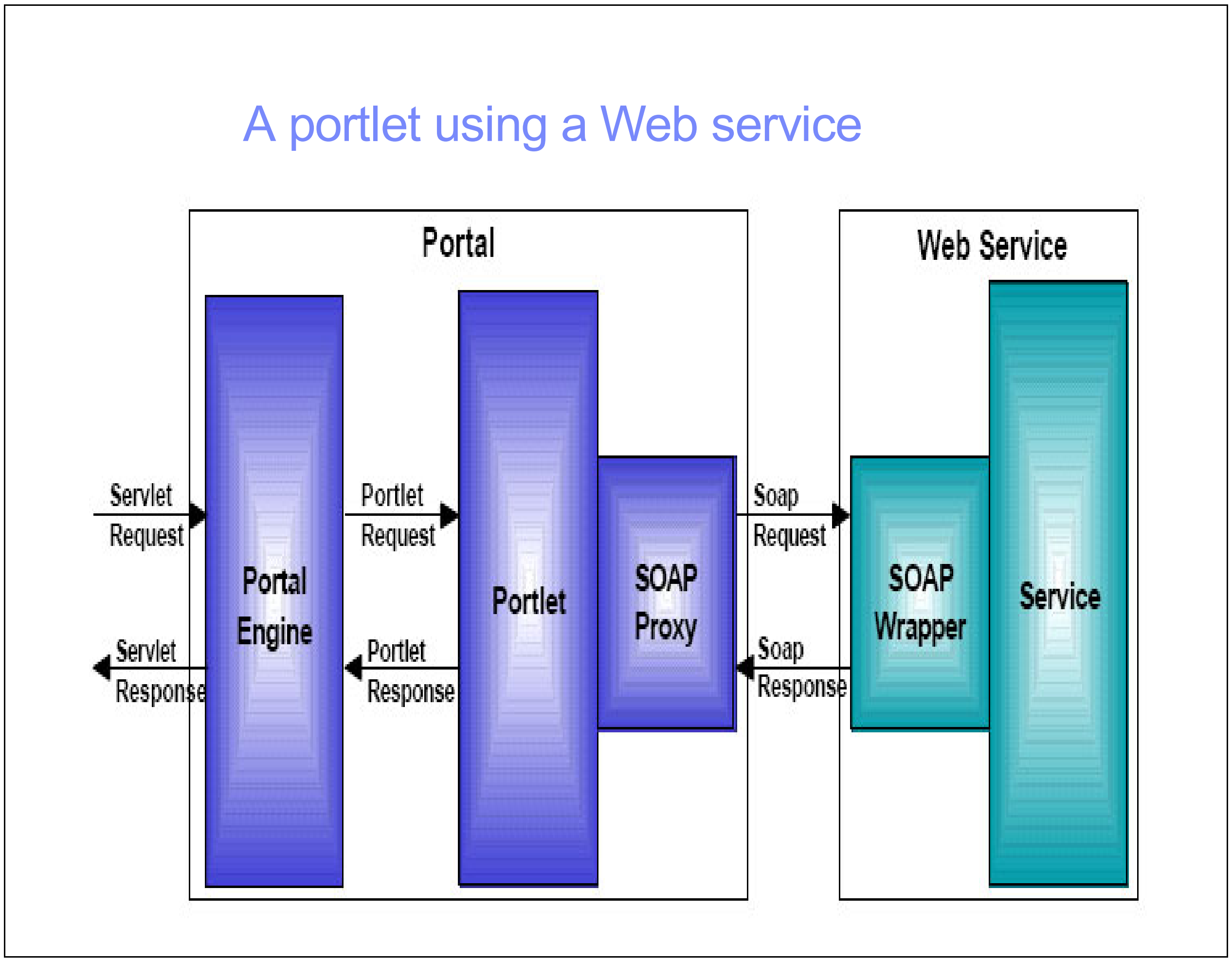


Figure 4-14 A portlet using a Web service

To simplify writing portlets using Web services, IBM provides a service proxy generator tool that automatically produces client code from a WSDL interface document, and optionally a service implementation document. If only a service interface document is used, the service proxy generator tool generates a generic service proxy which can be used to access any implementation of the given service interface. If both a service interface and a service implementation are used, the service proxy generator tool generates a service proxy that will only access the specified service implementation. The service proxy contains code that is specific to a binding within the service interface. For example, if the binding is a SOAP binding, then the service proxy will contain SOAP client code that is used to invoke the service.

Example 4-3 EmployeeDetailsPortlet.wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GetResults_Service"
   targetNamespace="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:portlet="http://www.ibm.com/wps/c2a"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ibm.com/wps/c2a/examples/hrdetails"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/

http://schemas.xmlsoap.org/wsdl/ http://www.w3.org/2001/XMLSchema
http://www.w3.org/2001/XMLSchema.xsd">

<types>
   <xsd:simpleType name="DEPT_NO">
      <xsd:restriction base="xsd:string"></xsd:restriction>
   </xsd:simpleType>
</types>

<message name="GetResultsMessageNameRequest">
   <part name="get_ResultsPartName" type="tns:DEPT_NO" />
</message>

<portType name="GetResults_Service">
   <operation name="get_ResultsOperation">
      <input message="tns:GetResultsMessageNameRequest" />
   </operation>
</portType>

<binding name="GetResultsBinding" type="tns:GetResults_Service">
   <portlet:binding />
   <operation name="get_ResultsOperation">
      <portlet:action name="hrportlet.HRPortletDetailsAction" type="simple"
         caption="Show all employees from this department."
         description="Get.Results.for.specified.sql.string" />
      <input>
      <portlet:param name="DEPT_NOParam" partname="get_ResultsPartName"
         caption="Show all employees from this department." />
      </input>
   </operation>
</binding>
</definitions>
```

4.3.3 Development tools

There are a lot of Portal development tools in the marketplace. We introduce some of IBM tools in this section:

- IBM Rational Application Developer V6
- Workplace Collaboration Services
- IBM WebSphere Portal Server

IBM Rational Application Developer V6

Rational Application Developer is an integrated development environment with full support for the J2EE programming model including EJB development, Web services, Web applications and Java. In previous releases this product was known as WebSphere Studio Application Developer. This tool includes integrated portal development, UML editing, code analysis, automated test and deployment tools, built in version control, and team tools. Everything you need to be productive and to make sure written code is well designed, scalable, and ready for production is included in Rational Application Developer. Additionally, everything is provided for version control and protection when developers work in large teams or on complex projects. Rational Application Developer is optimized for IBM WebSphere software.

Rational Application Developer V6.0 is part of the Rational Software Development Platform used to develop applications to be deployed to IBM WebSphere Application Server V6.0, V5.0.x, and IBM WebSphere Portal V5.0.2.2 and V5.1. The Rational Software Development Platform provides an integrated development environment (IDE) and tooling used to design, develop, test, debug, and deploy applications in support of the application development life cycle.

The IBM Rational Software Development Platform is built upon the IBM Eclipse SDK 3.0, which is an IBM supported version of the Eclipse V3.0 Workbench containing many new features, and a new look and feel. When used with the IBM Software Development Platform, you can access a broad range of requirements directly from Rational Application Developer for WebSphere Software with features such as the following:

Rational Web Developer tools allow accelerated use of portal, SOA, and J2EE.

- You can shorten the Java learning curve by using drag-and-drop components and point-and-click database connectivity.
- You can improve code quality by using automated tools for applying coding standard reviews, component, and Web service unit testing and multi-tier runtime analysis.
- Business applications can be integrated with Web services and SOA.

You can find more information about IBM Rational Application Developer at:

http://www.ibm.com/software/awdtools/developer/application

Workplace Collaboration Services

Workplace Collaboration Services is built on anarchitecture of components that is adaptable to enable people and teams to react quickly to changing business needs. IBM Workplace products provide the front-end to the IBM service-oriented architecture (SOA) strategy. The following services are provided by Workplace Collaboration Services:

- Team Collaboration Services
- Document Services
- Messaging Services
- Web Content Management Services
- Learning Services
- Workplace Managed Client

Team Collaboration Services

The IBM Workplace Team Collaboration™ Services component provides users with the capability to participate in online meetings, create libraries, and interact with team members through online chats, threaded discussion forums, and document sharing. It includes the following features:

- Applications provide users with access to Workplace applications, HTML-enabled Domino applications, and custom applications. Workplace applications include: Team Spaces, where members can participate in discussions/chats, share documents and a team calendar, and search for information; and Documents for managing online document libraries. Users can create and maintain a list of favorite document libraries.

- Web Conferences are online meetings in which moderators make presentations to conference participants.
- Templates provide tools for creating, customizing, and managing application templates, which are used in applications.

Document Services

The IBM Workplace Document Services component provides systematic, controlled access to critical documents and provides a fundamental document-management capability that is standards-based and has integrated collaborative capabilities. Workplace Documents supports both the IBM Workplace browser and the IBM Workplace rich client.

The IBM Workplace Document Services component provides access to the following:

- Document library capabilities provide document check-in and check-out, document locking, and version control.
- Structured access provides an easy method for setting up library access so that information needed organization-wide can be easily viewed, but selective information can be viewed only by a limited audience.
- Document editors provide the power to modify popular document types even when native editors are unavailable.
- Document author/owner/editor awareness through integrated instant messaging and chat capabilities.
- Security lets users store documents outside the file system to increase protection from viruses and other risks.

The IBM Workplace Messaging® Services component for the rich client provides the following:

- Offline support provides a secure method for users to create, import, edit, and save documents, presentations, and spreadsheets by supporting offline use and synchronization between local and server stores.
- Productivity tools provide the power to modify popular document types even when native editors are unavailable.

Messaging Services

IBM Workplace Messaging Services component is a cost-effective, standards-based messaging product that is security-rich, scalable, and easily deployed. It integrates with an organization's existing corporate infrastructure and uses the organization's LDAP directory to automatically create, delete, and authenticate user accounts; resolve addresses; and route mail. Workplace Messaging supports both the IBM Workplace browser and the IBM Workplace rich client.

The IBM Workplace Messaging Services component provides access to the following:

- Mail lets users send and receive e-mail messages.
- Calendar and Scheduling lets users maintain and manage calendar events and schedule meetings.
- Personal Address Book lets users maintain and manage contact information for people and for group mailing lists.

- Offline support that allows users to read, edit, and create mail while disconnected from the network.
- Integrated Instant Messaging and chat, including the ability to save chats. (To make this available to users, you must have a license for IBM Workplace Collaboration Services or a license for IBM Workplace Team Collaboration, and you must configure instant messaging.)

Web Content Management Services

The IBM Workplace Web Content Management™ Services component delivers powerful end-to-end Web content management through multiple Internet, intranet, extranet, and portal sites.

IBM Workplace Web Content Management is not installed as part of IBM Workplace Collaboration Services (installed separately):

- Content authoring is template-based, with a WYSIWYG rich text editor providing a guided process that does not require technical skills.
- Versioning and rollback provides a method for creating multiple content versions that can be used at different times or restored to previous versions, as needed.
- Automatic workflow processing ensures that the right people approve Web content before it is published and assures accuracy and relevancy of content.
- Integration of information from various sources allows reuse of information from back-end systems, improving transactional performance.
- Personalized delivery lets authors create content once and reuse it in different sites for users with different roles or preferences.
- Multiple database support allows use of IBM Lotus Domino, IBM DB2, Oracle, or IBM DB2 Content Manager as repositories.

Learning Services

IBM Workplace Learning Services component (Collaborative Learning) provides access to a scalable, flexible product for managing classroom-based and online learning activities, resources, curricula, and courseware catalogs.

IBM Workplace Collaborative Learning™ provides access to these features:

- The learning student experience provides an easy-to-use interface where students access courses. Students can search for courses and organize them in personalized folders, as well as preview, enroll in, and participate in courses online. Information about courses is stored on the Learning Server, while the courses themselves are presented on the Delivery Server.
- The learning management and delivery system components provide a Web-based administrative interface that course developers and instructors access to manage resources, learning programs, and skills development.
- The Authoring Tool is used by course developers on their own workstations to create course structure and content, assemble course packages, and import courses to Learning servers.

Workplace Managed Client

The IBM Workplace Managed Client provides rich client capabilities to the IBM Workplace Collaboration Services product. Users can access collaboration capabilities from their desktop, rather than from a browser. The IBM Workplace Managed Client is a separate component, which provides users with access to offline use and replication, as well as to these features:

- IBM Activity Explorer is used to track and manage activities related to a project or process. Users can capture and manage real-time communications and leverage the collaboration features of shared workspaces.
- The IBM data access tool is used to create relational database applications. Users can create forms, grids, and reports in order to add, edit, delete, and view summaries of database records stored in IBM Cloudscape™ databases.
- On Embedded browser is what users use to open and navigate Web pages directly from within the rich client.

IBM WebSphere Portal Server (WPS)

In order to be easy to use, the mechanisms for publishing portlets as remote portlet Web services, finding remote portlet Web services, binding to them and using remote portlets must be integrated seamlessly into portal products. We can identify four different dialog flows that need to be provided (see Figure 4-15 on page 101).

- Publishing portlets: Administrators can publish portlets to make them available for use by other portals as remote portlet Web services.
- Finding and binding portlets: Administrators can find remote portlet Web services and bind to these portlets.
- Using remote portlets: Users must be able to select and use remote portlets transparently, just as easily as local portlets.
- Finding and using remote portlets: “Power users” should be able to find remote portlet Web services by browsing the directory themselves.

Publishing portlets may either include two or three steps, depending on whether the portal has already been associated with a UDDI business. If this is not the case, WebSphere Portal Server prompts the administrator to enter the required business descriptions and publishes a business entry to the UDDI directory and associates the portal with that entry. Once the portal is associated with a business entry in UDDI, publishing portlets only requires two steps – in the first step, the administrator selects the portlet to be published and in the second step he provides a description for the new UDDI service entry to be created for the portlet.

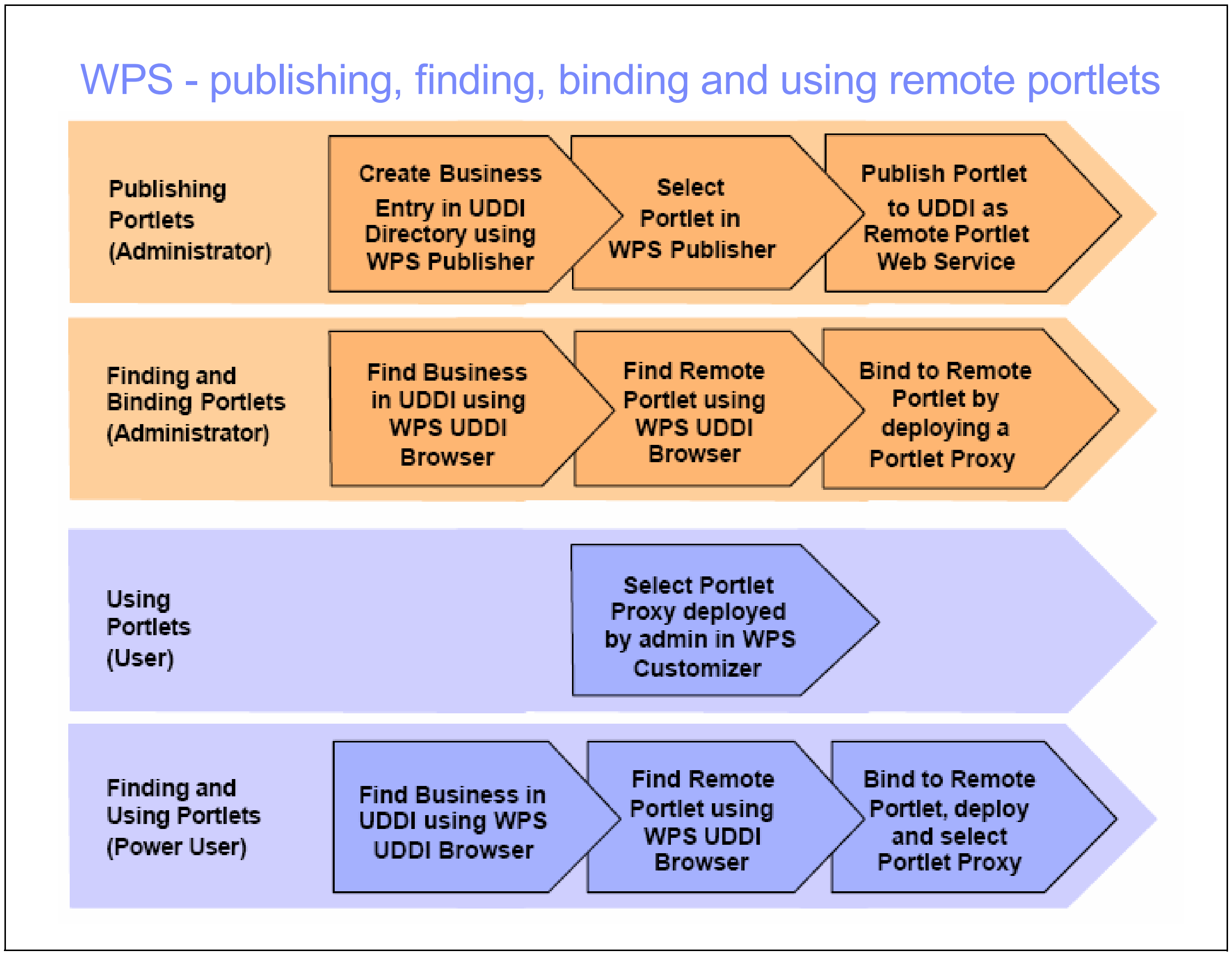

WPS - publishing, finding, binding and using remote portlets
Publishing Portlets (Administrator)
Create Business Entry in UDDI Directory using WPS Publisher
Select Portlet in WPS Publisher
Publish Portlet to UDDI as Remote Portlet Web Service
Finding and Binding Portlets (Administrator)
Find Business in UDDI using WPS UDDI Browser
Find Remote Portlet using WPS UDDI Browser
Bind to Remote Portlet by deploying a Portlet Proxy
Using Portlets (User)
Select Portlet Proxy deployed by admin in WPS Customizer
Finding and Using Portlets (Power User)
Find Business in UDDI using WPS UDDI Browser
Find Remote Portlet using WPS UDDI Browser
Bind to Remote Portlet, deploy and select Portlet Proxy

Figure 4-15 WPS - publishing, finding, binding and using remote portlets

Finding portlets requires three steps. First, the administrator uses the built-in UDDI browser to find remote portlet Web services and selects one of them. Second, he finds the desired portlet provided by the selected business and selects it. Finally, he lets WPS add the remote portlet to its portlet registry to make it available for portal users.

Using a remote portlet is as simple as using a locally installed portlet – users can select remote portlets in the customizer. To allow more sophisticated users to find and bind to remote portlets themselves, a portal may be configured to allow access to these functions for users. In this case, the dialog flow for the user would be identical to the workflow for administrators to find/bind portlets described above.

4.3.4 Conclusion

There is only one standard Web Services for Remote Portlets (WSRP) in this category, because it is a relatively new trend and might also lead to the creation of other standards and specifications.

WSRP is a Web services standard that allows for the plug-and-play of portals, other intermediary Web applications that aggregate content, and applications from disparate sources. WSRP Version 1.0 is an approved OASIS standard.

More information can be found at:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

4.4 Additional information for portals and SOA

- *WebSphere Version 6 Web Services Handbook Development and Deployment,* SG24-6461
- *Best practices: Developing portlets using JSR 168 and WebSphere Portal V5.02,* by Stefan Hepper and Marshall Lamb:

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html
- “Comparing the JSR 168 Java Portlet Specification with the IBM Portlet API”, by Stefan Hepper:

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0312_hepper/hepper.html
- “Why you should take an early look at the Java Portlet Specification V2.0”, by Stephan Hepper:

  http://www-128.ibm.com/developerworks/websphere/techjournal/0608_col_hepper/0608_col_hepper.html
- *Patterns: Extended Enterprise SOA and Web Services,* SG24-7135
- *XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture,* SG24-6826
- *DB2 for z/OS and WebSphere: The Perfect Couple,* SG24-6319
- “Developing business process portal applications using WebSphere tooling”, by Andreas Nauerz , Juergen Schaeck and Thomas Schaeck

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0505_nauerz/0505_nauerz.html
- *Web Services for Remote Portlets Specification,* found at:

  http://www.oasis-open.org/committees/wsrp
- *Java Portlet Specification, Version 1.0*, found at:

  http://jcp.org/aboutJava/communityprocess/final/jsr168
- *Architecting Portal Solutions,* SG24-7011
- *Develop and Deploy a Secure Portal Solution Using WebSphere Portal V5 and Tivoli Access Manager V5.1,* SG24-6325
- *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools,* SG24-6681
- *IBM Workplace Web Content Management for Portal 5.1 and IBM Workplace Web Content Management 2.5,* SG24-6792

5

Development tools

In this chapter, we introduce various IBM development tools that you can use in the course of creating database objects, and then later on expose these database objects as Web services.

- DB2 Developer Workbench
- Rational Application Developer
- WebSphere Developer for zSeries
- WebSphere Integration Developer
- Enterprise Generation Language (EGL) and SOA

© Copyright IBM Corp. 2006. All rights reserved.

5.1 DB2 Developer Workbench

The Development Center from DB2 for Linux, UNIX and Windows Version 8 is replaced in DB2 V9.1 by an Eclipse-based tool called Developer Workbench. The Developer Workbench is a graphical application that supports rapid development. Using the Developer Workbench, you can create new stored procedures, build stored procedures on local and remote DB2 servers, test and debug stored procedures. Developer Workbench allows you to manage your work in projects. Each Developer Workbench project saves your connections to specific databases, such as DB2 for z/OS servers, or DB2 servers on Linux, UNIX or Windows platforms.

DB2 Developer Workbench is supported on:

- Microsoft Windows 2000 Server and Professional, Microsoft Windows 2003 Server, and Microsoft XP Professional
- Red Hat Enterprise Linux 3 and SUSE Linux Enterprise Server 9

Note: At the time of writing this redbook, Developer Workbench is only supported on 32-bit platforms on Windows and Linux.

When you first open DB2 Developer Workbench Eclipse, the Welcome view gives you access to an overview, tutorials, samples and a description of what's new for this release. In addition, the icon at the upper right of the home page of the Welcome view opens the workbench.

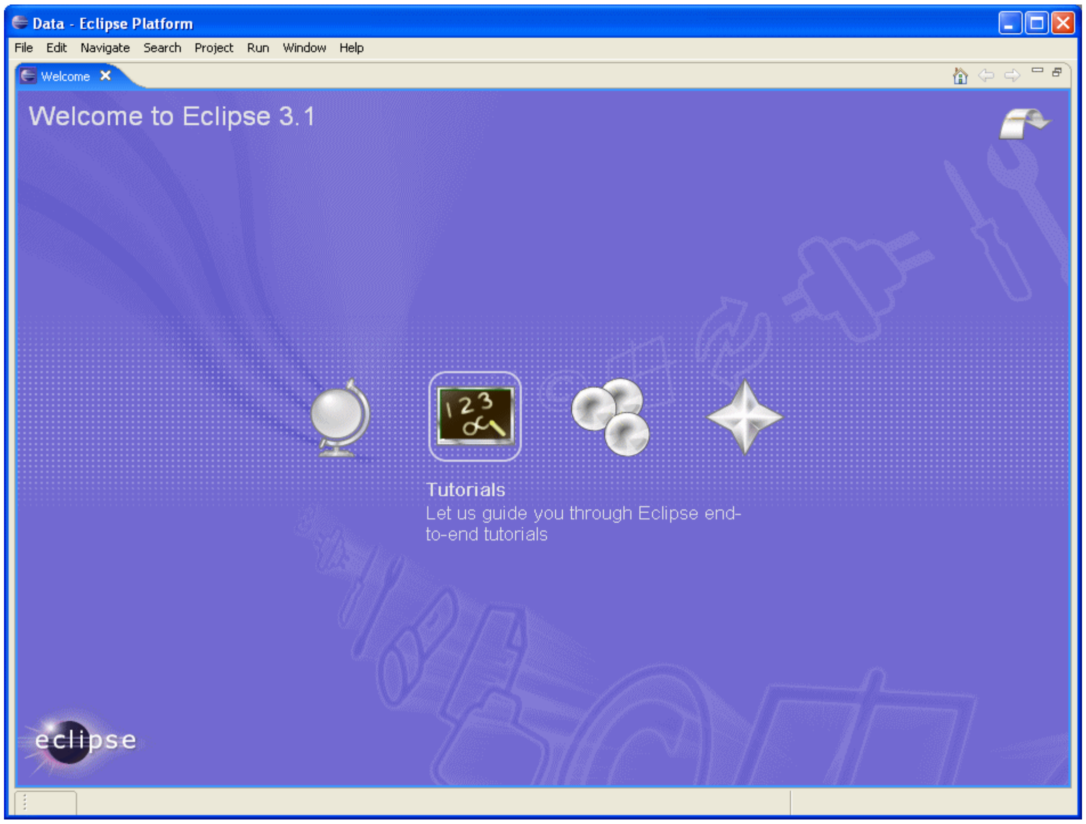

Data - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse 3.1
Tutorials
Let us guide you through Eclipse end-to-end tutorials
eclipse

Figure 5-1 Welcome view in Developer Workbench

Developer Workbench is a comprehensive development environment for creating, editing, debugging, deploying and testing DB2 stored procedures and user-defined functions (UDFs).

You can then later expose these stored procedures and UDFs that you have created in Developer Workbench as Web services. You can also use Developer Workbench to develop SQLJ applications, and create, edit, and run SQL statements and XML queries. Figure 5-2 shows the Developer Workbench user interface which looks very similar to the rest of the IBM Eclipse-based development tools.

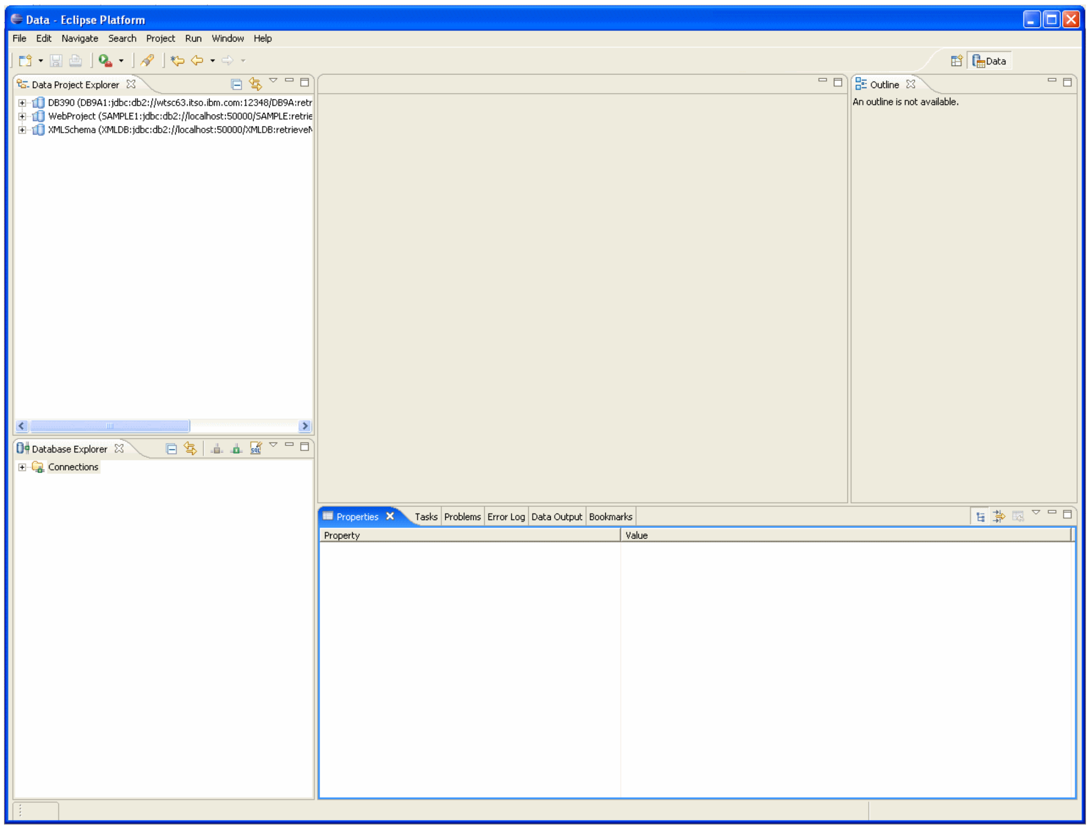

Data - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Data
Data Project Explorer
DB390 (DB9A1:jdbc:db2://wtsc63.itso.ibm.com:12348/DB9A:retr
WebProject (SAMPLE1:jdbc:db2://localhost:50000/SAMPLE:retrie
XMLSchema (XMLDB:jdbc:db2://localhost:50000/XMLDB:retrieveN
Outline
An outline is not available.
Database Explorer
Connections
Properties
Tasks
Problems
Error Log
Data Output
Bookmarks
Property
Value

Figure 5-2 Developer Workbench User Interface

In addition to existing Development Center functionality, Developer Workbench provides additional new features as described below:

- Developer Workbench information center and tutorials

  Developer Workbench help and tutorials are available in an information center that is installed with Developer Workbench. This information is for Developer Workbench only, and it is not installed with the DB2 information center CD. To access the Developer Workbench help and tutorials, click **Help** → **Help Contents** from the main menu in the product. You can also link directly to important getting started information from the Welcome page in Developer Workbench by clicking **Help** → **Welcome**.

- Migrate existing Development Center projects

  You can use a wizard to migrate existing Development Center projects into Developer Workbench.

- Compare routines

  You can compare and make changes between two routines that are contained within a data development project in Developer Workbench. You can also compare routine attributes for routines that are stored on a server.

- Deploy routines to unlike servers

  You can deploy routines that were created for one DB2 database to a DB2 database on a different platform. For example, you can create a routine for a DB2 for Linux, UNIX, and Windows database and then deploy it to a DB2 for z/OS database. (Note that not all server combinations are supported)

- Binary deploy

  For SQL or Java stored procedures targeting DB2 for z/OS Version 8 or higher, you can deploy without going through a full rebuild. The binaries for a SQL procedure or the JAR for a Java procedure are copied from the source to the target system.

- Launch Visual Explain

  You can launch Visual Explain for DB2 for z/OS or DB2 for Linux, UNIX, and Windows SQL statements from either the routine editor for SQL routines or from the wizard that is used to create a routine.

- Develop SQLJ applications

You can develop SQLJ applications by using the following features:

  - Generate an SQLJ template file by using a wizard
  - Translate and compile automatically
  - Customize by using a wizard
  - Print the profile file
  - Edit SQLJ applications by using code assist and templates
  - Debug SQLJ files

- Team support

  You can share your Developer Workbench data development project by using either CVS or ClearCase®. After you share your project, you can manage all changes and update history, and you can synchronize your files with the repository.

- Multiple JAR support

  You can create Java stored procedures that depend on code that is contained in multiple JAR files that are installed on the server. You can also package multiple Java stored procedures within the same JAR file on the server.

- SQL procedure versioning for z/OS

  There is support for native SQL stored procedures and versioning of native SQL stored procedures targeting DB2 for z/OS servers.

- Package variation support for SQL and SQLJ Java stored procedures

  You can create package variations from the Database explorer for SQL and SQLJ stored procedures targeting DB2 for z/OS. These package variations are used for creating copies of existing packages with different bind options.

- Table data editing

  You can use an editor to edit the data that is contained in a table. You can edit existing values, delete an existing row, or insert a new row.

- Data extract and load

  You can extract the data from a table or view into a file on the local file system. You can use this file to load the data into a table.

- Stored procedure debugger

  Developer Workbench includes integrated stored procedure debugging capabilities. You can debug SQL or Java stored procedures that target supported DB2 servers, or Java stored procedures that target supported Derby servers.

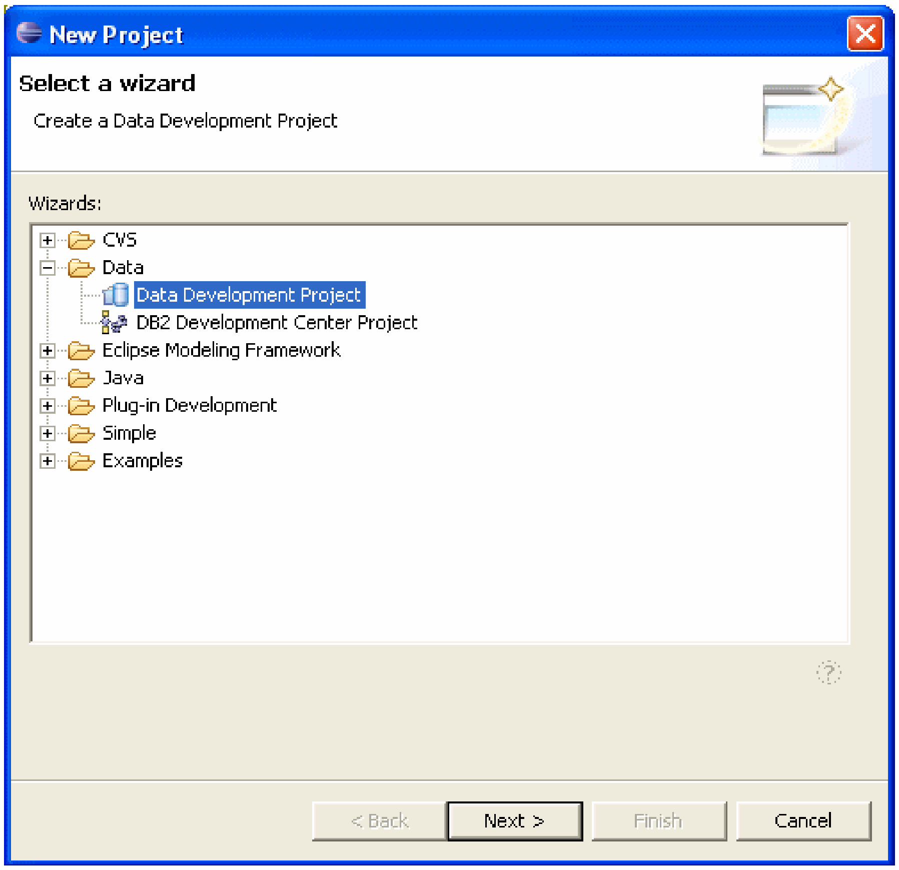
- XML support

  Developer Workbench contains support for XML functions, the XML data type, and XML schema registration. You can also create XQueries with the XQuery builder. We discuss XML support further in 5.1.2, “New XML support in DB2 Developer Workbench” on page 111, and Appendix C.1, “New features of native XML data store in DB2 V9.1 for Linux, UNIX and Windows” on page 594.

5.1.1 Creating a stored procedure using Developer Workbench

Since DB2 Developer Workbench is a new tool, we provide an example that shows how to create a Java stored procedure with Developer Workbench so you can familiarize with the new user interface. You will see that Developer Workbench is very similar to the DB2 V8 Development Center and provides much more functionalities.

You can create new stored procedures and UDF in a new project or in existing projects. You can also migrate your existing DB2 V8 Development Center project to DB2 V9 Developer Workbench. To create new project, select **Data Development Project** in the New Project wizard. To migrate an existing Development Center project from DB2 V8 to DB2 V9 Developer Workbench, select **DB2 Development Center Project** instead. Figure 5-3 shows the New Project wizard.


Select a wizard

Create a Data Development Project

Figure 5-3 New project wizard in Developer Workbench

Suppose you already have a `WebProject` created in Developer Workbench, Figure 5-4 on page 108 shows how you create a simple Java stored procedure that queries DB2’s SYSCAT.PROCEDURE catalog to list the stored procedures we currently have in the SAMPLE database.

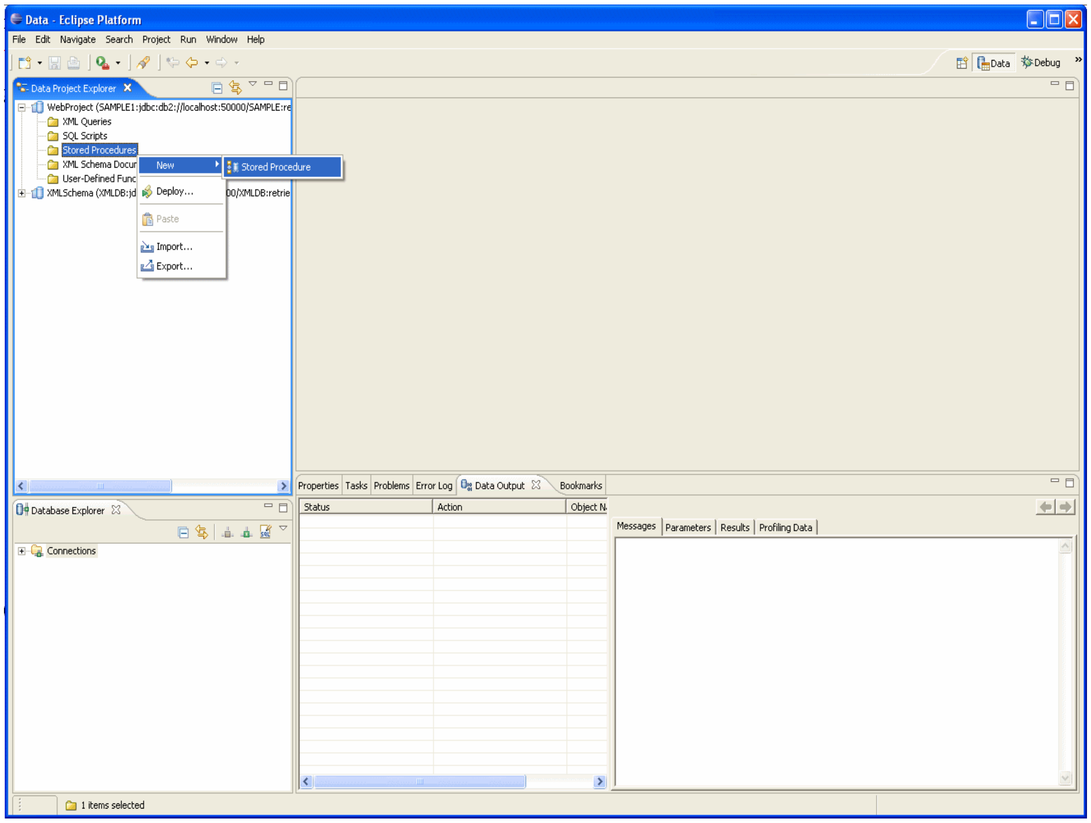
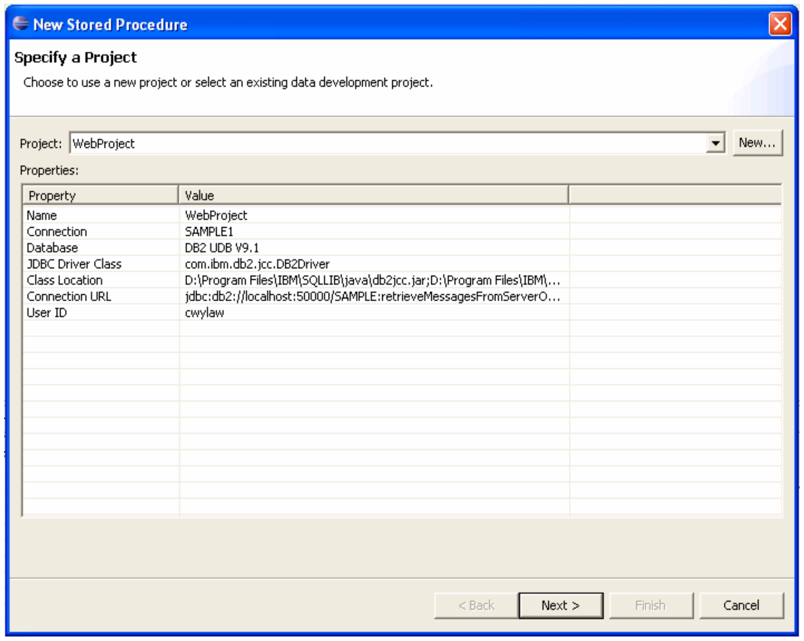


Figure 5-4   New Stored Procedure wizard

In Figure 5-5 we specify that a new Java stored procedure is to be created in our `WebProject`.


New Stored Procedure
Specify a Project
Choose to use a new project or select an existing data development project.
New...
< Back
Next >
Finish
Cancel

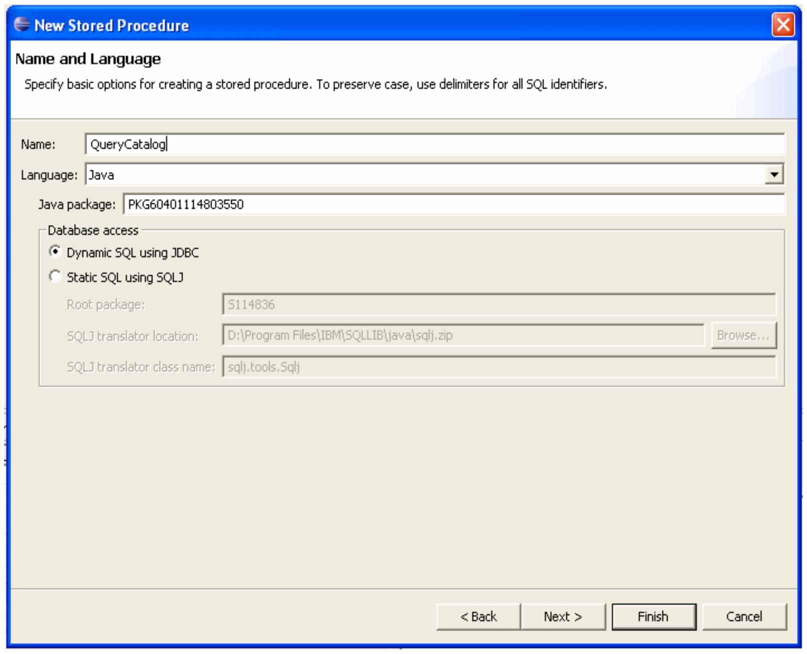
Project: WebProject

Properties:

Property	Value
Name	WebProject
Connection	SAMPLE1
Database	DB2 UDB V9.1
JDBC Driver Class	com.ibm.db2.jcc.DB2Driver
Class Location	D:\Program Files\IBM\SQLLIB\java\db2jcc.jar;D:\Program Files\IBM\...
Connection URL	jdbc:db2://localhost:50000/SAMPLE:retrieveMessagesFromServerO...
User ID	cwylaw

Figure 5-5   Create Java stored procedure in our WebProject

Click **Next**. Then enter `QueryCatalog` as the stored procedure name. Choose **Java** in the **Language** drop down list. Keep the defaults for the rest of the fields in this dialog as shown in Figure 5-6.


New Stored Procedure

Name and Language

Specify basic options for creating a stored procedure. To preserve case, use delimiters for all SQL identifiers.

Name: QueryCatalog

Language: Java

Java package: PKG60401114803550

Database access

- (●) Dynamic SQL using JDBC
- ( ) Static SQL using SQLJ

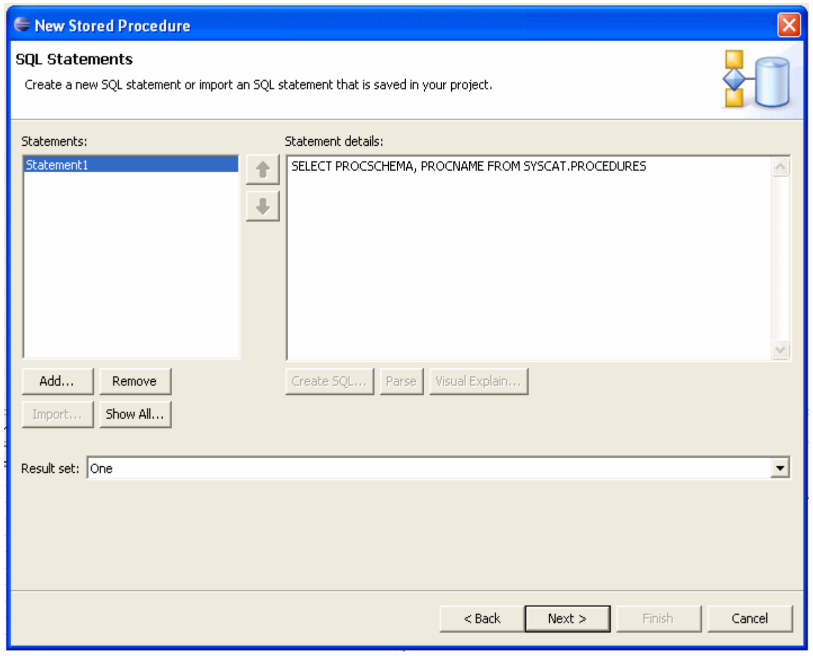
Root package: S114836

SQLJ translator location: D:\Program Files\IBM\SQLLIB\java\sqlj.zip

SQLJ translator class name: sqlj.tools.Sqlj

Figure 5-6   Specifying stored procedure name

Click **Next**. We keep the default query and return only one result set as shown in Figure 5-7.


New Stored Procedure

SQL Statements

Create a new SQL statement or import an SQL statement that is saved in your project.

Statements: Statement1

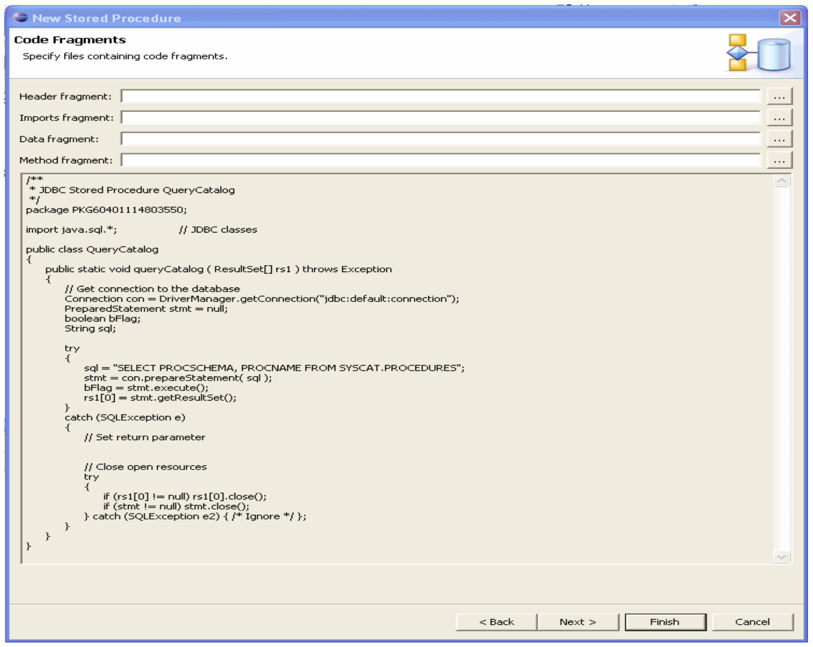
Statement details:

```
SELECT PROCSCHEMA, PROCNAME FROM SYSCAT.PROCEDURES
```

Result set: One

Figure 5-7   Specifying your SQL statement in the New Stored Procedure wizard

Click **Next** → **Next** → **Next** to take the default settings for the next three screens. Then you see the stored procedure code that is generated for you, as shown in Figure 5-8 on page 110.

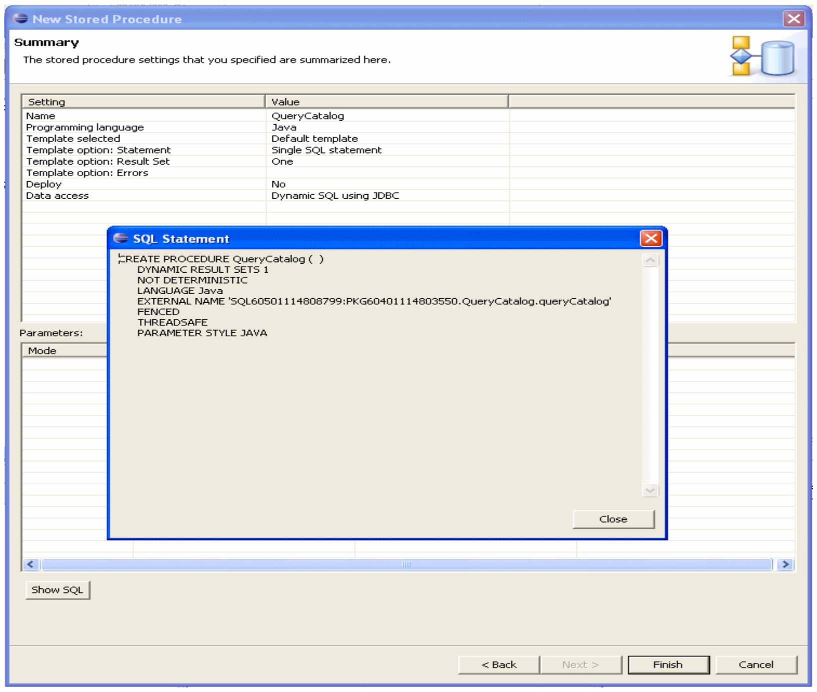


Header fragment:
Imports fragment:
Data fragment:
Method fragment:

```
/**
 * JDBC Stored Procedure QueryCatalog
 */
package PKG60401114803550;

import java.sql.*;                  // JDBC classes

public class QueryCatalog
{
    public static void queryCatalog ( ResultSet[] rs1 ) throws Exception
    {
        // Get connection to the database
        Connection con = DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement stmt = null;
        boolean bFlag;
        String sql;

        try
        {
            sql = "SELECT PROCSCHEMA, PROCNAME FROM SYSCAT.PROCEDURES";
            stmt = con.prepareStatement( sql );
            bFlag = stmt.execute();
            rs1[0] = stmt.getResultSet();
        }
        catch (SQLException e)
        {
            // Set return parameter


            // Close open resources
            try
            {
                if (rs1[0] != null) rs1[0].close();
                if (stmt != null) stmt.close();
            } catch (SQLException e2) { /* Ignore */ };
        }
    }
}
```

Figure 5-8 Stored procedure code

Click **Next** to show a summary of the stored procedure settings. You can also click **Show SQL** to see the `CREATE PROCEDURE` statement being issued on your behalf. See Figure 5-9.



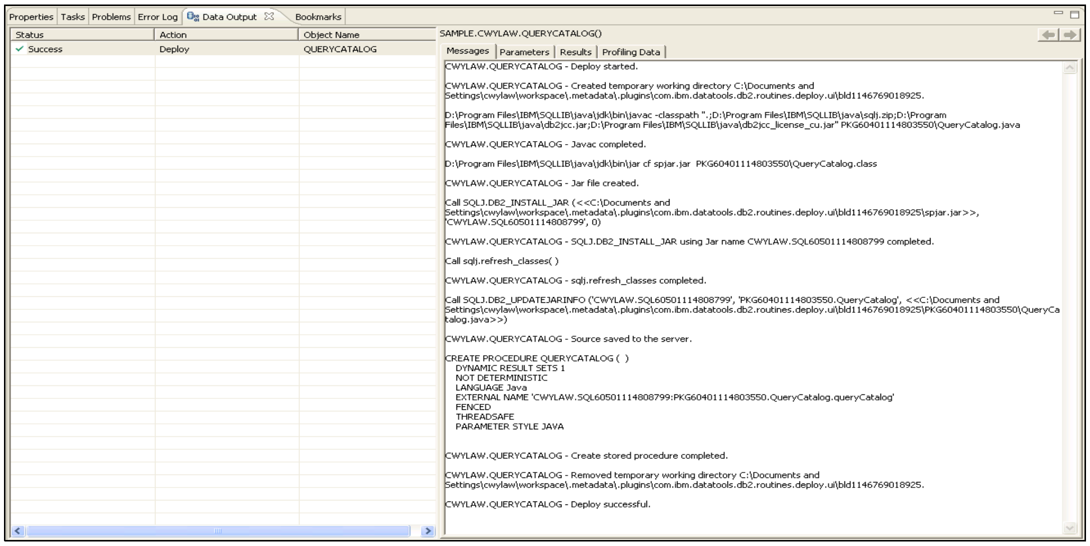
Setting	Value
Name	QueryCatalog
Programming language	Java
Template selected	Default template
Template option: Statement	Single SQL statement
Template option: Result Set	One
Template option: Errors	
Deploy	No
Data access	Dynamic SQL using JDBC

```
CREATE PROCEDURE QueryCatalog ( )
    DYNAMIC RESULT SETS 1
    NOT DETERMINISTIC
    LANGUAGE Java
    EXTERNAL NAME 'SQL60501114808799:PKG60401114803550.QueryCatalog.queryCatalog'
    FENCED
    THREADSAFE
    PARAMETER STYLE JAVA
```

Figure 5-9 Stored procedure settings summary and CREATE PROCEDURE statement

Click **Close** to close the SQL Statement if you have the SQL statement dialog opened. Then click **Finish** to complete creating the stored procedure. At this point the Java stored procedure source will be displayed as shown in Figure 5-10 on page 111.

In Data Project Explorer, locate the QueryCatalog stored procedure then right-click **QUERYCATALOG** → **Deploy** to deploy the stored procedure. You shall see output messages in the Data Output view similar to what is shown in Figure 5-10 at the bottom of the Developer Workbench.

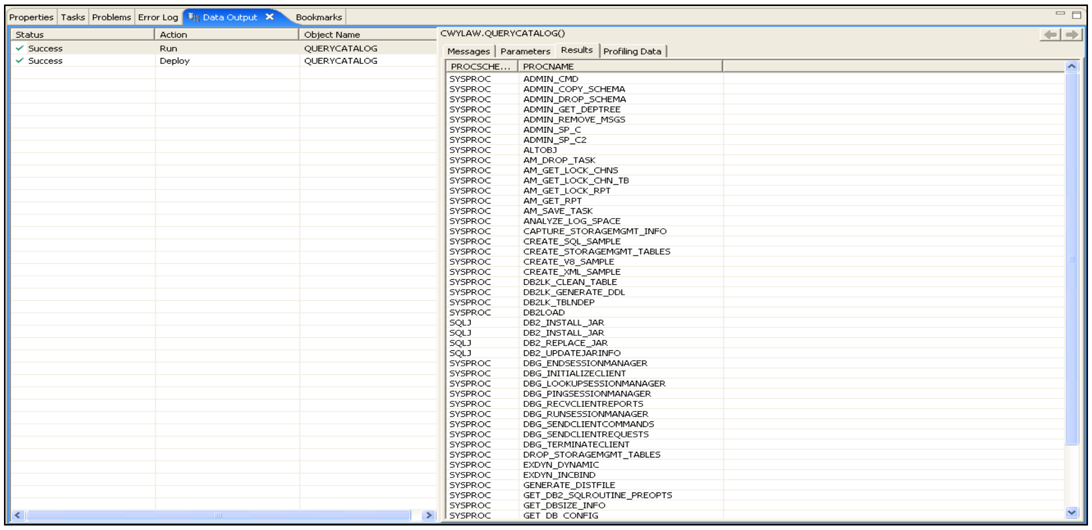


Status	Action	Object Name
✓ Success	Deploy	QUERYCATALOG

```
SAMPLE.CWYLAW.QUERYCATALOG()

CWYLAW.QUERYCATALOG - Deploy started.

CWYLAW.QUERYCATALOG - Created temporary working directory C:\Documents and
Settings\cwylaw\workspace\.metadata\.plugins\com.ibm.datatools.db2.routines.deploy.ui\bld1146769018925.

D:\Program Files\IBM\SQLLIB\java\jdk\bin\javac -classpath ".;D:\Program Files\IBM\SQLLIB\java\sqlj.zip;D:\Program
Files\IBM\SQLLIB\java\db2jcc.jar;D:\Program Files\IBM\SQLLIB\java\db2jcc_license_cu.jar" PKG60401114803550\QueryCatalog.java

CWYLAW.QUERYCATALOG - Javac completed.

D:\Program Files\IBM\SQLLIB\java\jdk\bin\jar cf spjar.jar  PKG60401114803550\QueryCatalog.class

CWYLAW.QUERYCATALOG - Jar file created.

Call SQLJ.DB2_INSTALL_JAR (<<C:\Documents and
Settings\cwylaw\workspace\.metadata\.plugins\com.ibm.datatools.db2.routines.deploy.ui\bld1146769018925\spjar.jar>>,
'CWYLAW.SQL60501114808799', 0)

CWYLAW.QUERYCATALOG - SQLJ.DB2_INSTALL_JAR using Jar name CWYLAW.SQL60501114808799 completed.

Call sqlj.refresh_classes( )

CWYLAW.QUERYCATALOG - sqlj.refresh_classes completed.

Call SQLJ.DB2_UPDATEJARINFO ('CWYLAW.SQL60501114808799', 'PKG60401114803550.QueryCatalog', <<C:\Documents and
Settings\cwylaw\workspace\.metadata\.plugins\com.ibm.datatools.db2.routines.deploy.ui\bld1146769018925\PKG60401114803550\QueryCa
talog.java>>)

CWYLAW.QUERYCATALOG - Source saved to the server.

CREATE PROCEDURE QUERYCATALOG (  )
   DYNAMIC RESULT SETS 1
   NOT DETERMINISTIC
   LANGUAGE Java
   EXTERNAL NAME 'CWYLAW.SQL60501114808799:PKG60401114803550.QueryCatalog.queryCatalog'
   FENCED
   THREADSAFE
   PARAMETER STYLE JAVA


CWYLAW.QUERYCATALOG - Create stored procedure completed.

CWYLAW.QUERYCATALOG - Removed temporary working directory C:\Documents and
Settings\cwylaw\workspace\.metadata\.plugins\com.ibm.datatools.db2.routines.deploy.ui\bld1146769018925.

CWYLAW.QUERYCATALOG - Deploy successful.
```

Figure 5-10 Messages in Data Output view when deploying the Java stored procedure

In Data Project Explorer, right-click **QUERYCATALOG** → **Run** to run the `QueryCatalog` stored procedure, test the procedure and confirm it returns the result set as expected.



Status	Action	Object Name
✓ Success	Run	QUERYCATALOG
✓ Success	Deploy	QUERYCATALOG

PROCSCHE...	PROCNAME
SYSPROC	ADMIN_CMD
SYSPROC	ADMIN_COPY_SCHEMA
SYSPROC	ADMIN_DROP_SCHEMA
SYSPROC	ADMIN_GET_DEPTREE
SYSPROC	ADMIN_REMOVE_MSGS
SYSPROC	ADMIN_SP_C
SYSPROC	ADMIN_SP_C2
SYSPROC	ALTOBJ
SYSPROC	AM_DROP_TASK
SYSPROC	AM_GET_LOCK_CHNS
SYSPROC	AM_GET_LOCK_CHN_TB
SYSPROC	AM_GET_LOCK_RPT
SYSPROC	AM_GET_RPT
SYSPROC	AM_SAVE_TASK
SYSPROC	ANALYZE_LOG_SPACE
SYSPROC	CAPTURE_STORAGEMGMT_INFO
SYSPROC	CREATE_SQL_SAMPLE
SYSPROC	CREATE_STORAGEMGMT_TABLES
SYSPROC	CREATE_V8_SAMPLE
SYSPROC	CREATE_XML_SAMPLE
SYSPROC	DB2LK_CLEAN_TABLE
SYSPROC	DB2LK_GENERATE_DDL
SYSPROC	DB2LK_TBLNDEP
SYSPROC	DB2LOAD
SQLJ	DB2_INSTALL_JAR
SQLJ	DB2_INSTALL_JAR
SQLJ	DB2_REPLACE_JAR
SQLJ	DB2_UPDATEJARINFO
SYSPROC	DBG_ENDSESSIONMANAGER
SYSPROC	DBG_INITIALIZECLIENT
SYSPROC	DBG_LOOKUPSESSIONMANAGER
SYSPROC	DBG_PINGSESSIONMANAGER
SYSPROC	DBG_RECVCLIENTREPORTS
SYSPROC	DBG_RUNSESSIONMANAGER
SYSPROC	DBG_SENDCLIENTCOMMANDS
SYSPROC	DBG_SENDCLIENTREQUESTS
SYSPROC	DBG_TERMINATECLIENT
SYSPROC	DROP_STORAGEMGMT_TABLES
SYSPROC	EXDYN_DYNAMIC
SYSPROC	EXDYN_INCBIND
SYSPROC	GENERATE_DISTFILE
SYSPROC	GET_DB2_SQLROUTINE_PREOPTS
SYSPROC	GET_DBSIZE_INFO
SYSPROC	GET_DB_CONFIG

Figure 5-11 Messages in the Data Output view when running the stored procedure

As you can see, DB2 Developer Workbench allows you to create, deploy, test and run your stored procedures quickly and efficiently.

5.1.2 New XML support in DB2 Developer Workbench

The upcoming DB2 V9.1 for Linux, UNIX and Windows, and DB2 V9 for z/OS, now features the new support for XML as a first class data type just like any other SQL type. The XML data type can be used in a **`CREATE TABLE`** statement to define a new column of type XML. You can now seamlessly integrate XML with their existing relational data, and exploit both tabular and hierarchical data models, and enjoy the flexibility of using both SQL and XQuery (on DB2 for Linux, UNIX and Windows only) in your applications. The new DB2 Developer Workbench contains the following types of XML support:

- Support for the XML data type
- Support for XML schemas
- XML document validation
- XQuery builder

Stored procedure support

- You can create stored procedures that contain XML data type parameters or return XML data types.
- You can run stored procedures that contain XML data types as input or output parameters.
- You can import XML queries that were generated by the XQuery builder into the procedure body when you are creating a stored procedure.

Data Output view support

- You can view XML data type columns on the Results page.
- For any column that can contain XML documents, you can view the content as a tree or the document text.

SQL builder support

- The XML data type is displayed anywhere that other data types are displayed.
- You can select XML functions in the Expression® builder.
- You can run SQL statements that contain host variables where the column associated with the host variable is an XML data type.
- You can insert or update column values when the column value is an XML data type.

XML schema support

- From the Database Explorer in Developer Workbench, you can load existing XML schemas and XML schema documents from the XML schema repository in the database and view properties such as target namespace or schema location.
- You can register a new XML schema with its corresponding XML schema documents from the file system.
- You can drop XML schemas and XML schema documents from the XML schema repository in the database.
- You can view and edit the source for XML schema documents that make up an XML schema.

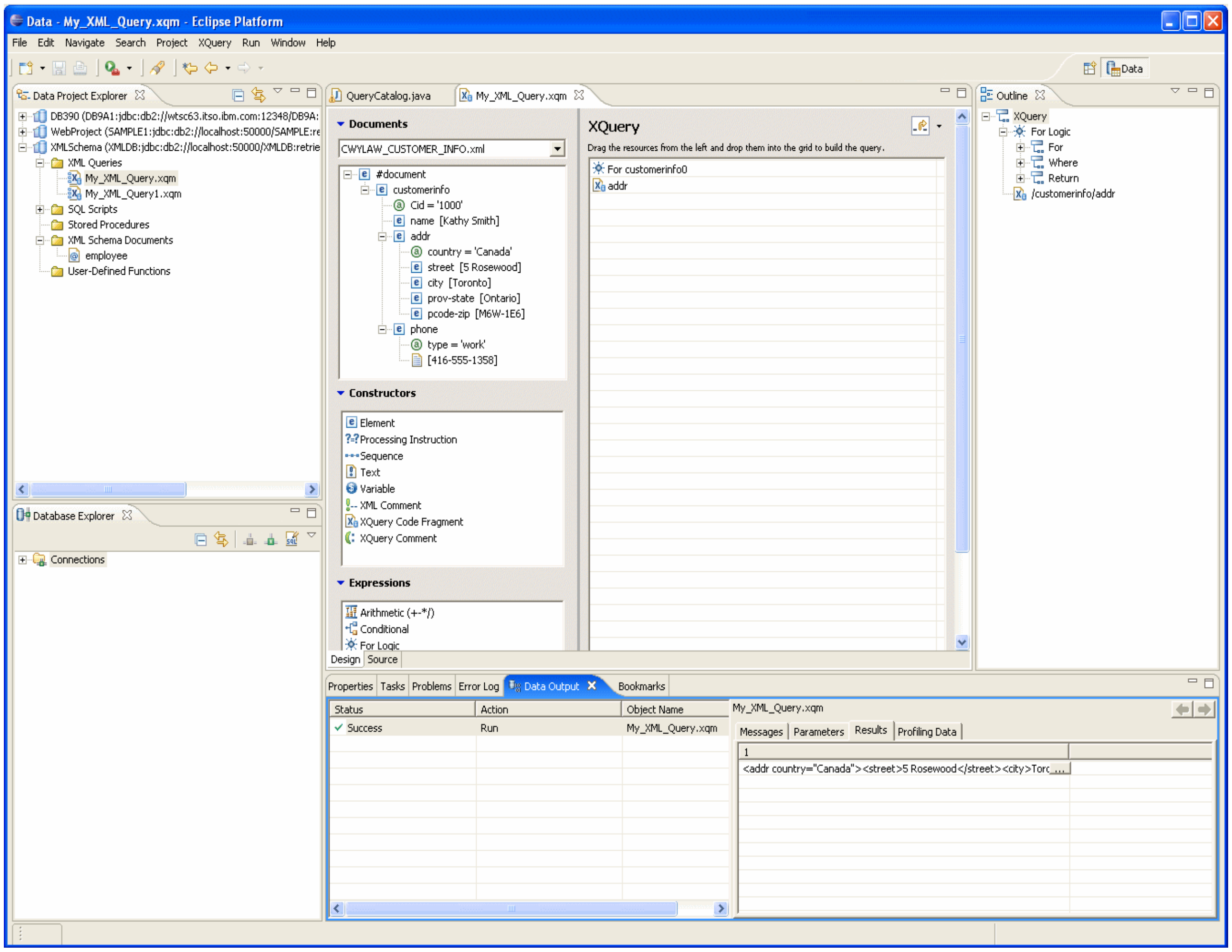
XML document validation

- You can edit and update an XML data type column.
- You can perform XML value validation for the XML document in the column against a registered XML schema.

XQuery builder

The Developer Workbench also provides an XQuery builder for creating queries without having to understand the details of XQuery semantics. The XQuery builder is an Eclipse-based tool to help you create queries against XML data that is in DB2 databases. The XQuery builder is part of the DB2 Developer Workbench. With the XQuery builder, you can create complete queries without needing to understand XQuery semantics. You can build an

XML query visually by selecting sample resultant nodes from a tree representation of a schema or XML document, and dragging the nodes onto a return grid. After a node is listed on the return grid, you can drill down into the query to add predicates and sorting preferences. You can drill down multiple levels in a query to specify nested predicates, clauses, and expressions. After you build your query, you can run it directly from Developer Workbench to test the query. Figure 5-12 shows the XQuery builder user interface.



Status	Action	Object Name
✓ Success	Run	My_XML_Query.xqm

```
1
<addr country="Canada"><street>5 Rosewood</street><city>Torc...
```

Figure 5-12 XQuery builder user interface

In C.9, “Create and register an XML schema using Developer Workbench” on page 629, we provide a step-by-step example of how to create XML schema, and later on use the same XML schema to validate XML column data in Developer Workbench.

5.2 Rational Application Developer

The creation of high-quality service-oriented systems requires a development environment which brings all parts of SOA (access to data servers, integration of Web Services, composition of portal solutions) together. In this section we describe IBM Rational Application Developer (RAD) V6.0 which is a component of the IBM Rational Software Development Platform and helps developers to quickly create, test and deploy SOA and J2EE applications.

5.2.1 The IBM Rational Software Development Platform

The IBM Rational Software Development Platform is an integrated set of products which cover the management of the entire software development process. Software modelers, architects, developers, and testers can use the same tooling to be more efficient in exchanging assets, following common processes, managing change and requirements, maintaining status, and improving quality.

The products of this platform are built upon the Eclipse 3.0 framework, thus allowing a seamless integration of the different modules. It provides a team-based environment with capabilities that are optimized for the key roles of a development team and enables a high degree of team cohesion through shared access to common requirements, test results, software assets, workflow, and process guidance. Combined, these capabilities improve both individual and team productivity.

Figure 5-13 shows the platform tooling covering the different roles of the software development process.

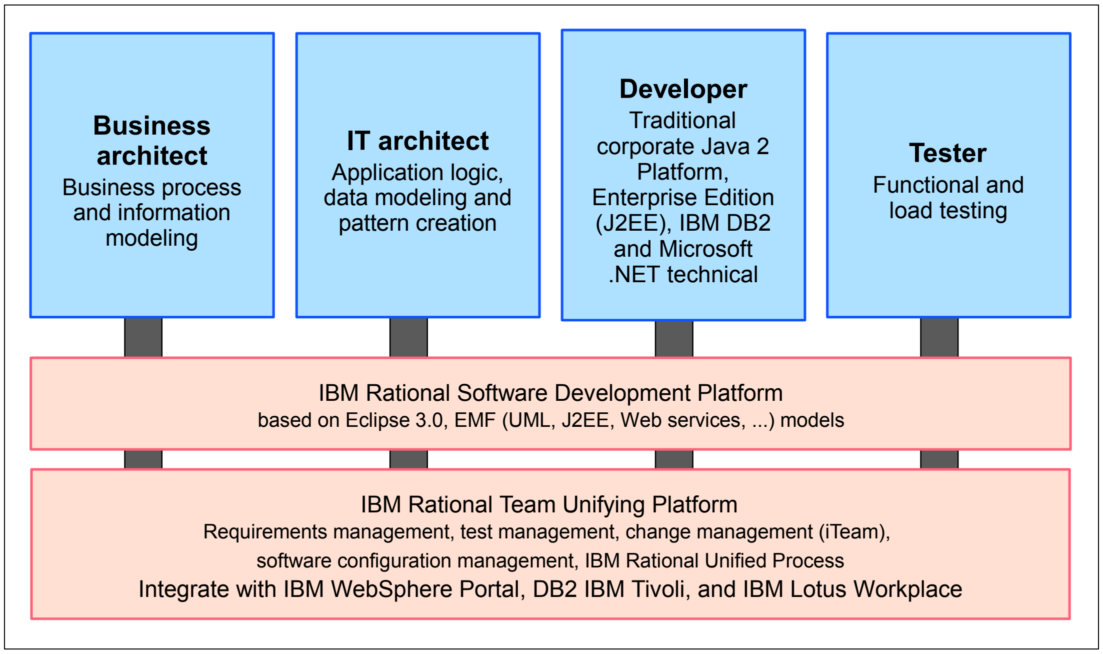

Business architect
Business process and information modeling
IT architect
Application logic, data modeling and pattern creation
Developer
Traditional corporate Java 2 Platform, Enterprise Edition (J2EE), IBM DB2 and Microsoft .NET technical
Tester
Functional and load testing
IBM Rational Software Development Platform
based on Eclipse 3.0, EMF (UML, J2EE, Web services, ...) models
IBM Rational Team Unifying Platform
Requirements management, test management, change management (iTeam),
software configuration management, IBM Rational Unified Process
Integrate with IBM WebSphere Portal, DB2 IBM Tivoli, and IBM Lotus Workplace

Figure 5-13 IBM Rational Software Development Platform tooling

A number of selected tools of the Rational Software Development Platform is listed in Table 5-1 on page 114.

Table 5-1 Rational Software Development Platform tools

Key areas	Rational Software Development Platform products
Requirements and architecture	IBM Rational Software Modeler IBM Rational Software Architect IBM WebSphere Business Integration Modeler IBM Rational RequisitePro®

Key areas	Rational Software Development Platform products
Design and construction	IBM DB2 Information Integrator IBM Rational Application Developer IBM Rational Web Developer IBM Rational Suite for Technical Developers IBM WebSphere Development Studio for iSeries™ IBM Workplace Designer
Testing tools	IBM Rational Functional Tester IBM Rational Manual Tester IBM Rational Performance Tester IBM Rational PurifyPlus™
Software Configuration Management	IBM Rational ClearCase family IBM Rational ClearQuest® family
Deployment management	IBM Tivoli Access Manager IBM Tivoli Configuration Manager IBM Tivoli Enterprise™ Console
Process and Portfolio Management	IBM Rational Portfolio Manager IBM Rational Team Unifying Platform™ IBM Rational Unified Process®

For more information about the platform visit the IBM DeveloperWorks site at:

`http://www.ibm.com/developerworks/platform/`

5.2.2 IBM Rational Application Developer

This section introduces the data server- and SOA-related features of the IBM Rational Application Developer (RAD) V6.0.

Overview

RAD helps developers to quickly design, develop, analyze, test, profile, and deploy service-oriented architecture (SOA), Java, J2EE and portal applications. It includes full support for the J2EE programming model, integrated portal development features, Unified Modeling Language (UML) visual editing capabilities, code analysis functions and automated test and deployment tools. RAD is optimized for IBM WebSphere software and provides capabilities for deploying to other runtime platforms as well.

Some of the features RAD provides are:

- Support of SOA development

  RAD includes visual construction tools for developing the services needed in SOA applications and provides the tools you need to discover, create, build, test, deploy and publish the services that comprise SOA applications. You can build new applications from scratch or enable existing applications for the SOA architecture.

- Easy to build, test and deploy J2EE applications

  The RAD environment includes full support for the J2EE programming model, including Web, Java, Web services and EJB development. A visual editor enables drag-and-drop creation of interfaces, as well as binding data to interface components. You can visualize and graphically edit the code through a UML-based visual editor and automate many functions.

- Database connectivity tools

  RAD provides a number of tools to ease the integration and creation of heterogeneous data sources. Wizards allow you to connect to data sources, edit and manipulate the data source metadata, support the access of data using APIs with different abstraction levels (JDBC, API, data beans, EJB) and include connectors for a number of relational databases from vendors like IBM, Microsoft, Oracle and Sybase.

- Automated deployment tools

  RAD provides support for testing and debugging of local and server-side code on IBM WebSphere Application Server, WebSphere Application Server Express, WebSphere Portal Server and Apache Tomcat. You can create and configure many elements of your unit test including server instances and breakpoints. In addition you have the ability to step through the code and even modify it without restarting the unit test server.

RAD V6.0 supports the following operating systems:

- Microsoft Windows:
  - Windows XP with Service Packs 1 and 2
  - Windows 2000 Professional with Service Packs 3 and 4
  - Windows 2000 Server with Service Packs 3 and 4
  - Windows 2000 Advanced Server with Service Packs 3 and 4
  - Windows Server® 2003 Standard Edition
  - Windows Server 2003 Enterprise Edition
- Linux on Intel®:
  - Red Hat Enterprise Linux Workstation V3 (all service packs)
  - SuSE Linux Enterprise Server (SLES) V9 (all service packs)

The Rational Application Developer includes a vast range of features targeting on J2EE development and Web development. In the following section we focus on the features related to the backing of data servers and SOA technologies.

Note: The *Rational Application Developer V6 Programming Guide*, SG24-6449, redbook contains a comprehensive description of the features provided by RAD.

Basic RAD concepts

When you start RAD, you see the *Workbench*, which refers to the desktop development environment of this application. The Workbench window includes the toolbar, the window menu, and a set of views and editors. The built-in views and editors allow you to perform a variety of tasks, like browsing through your project, editing different resources (for example, Java source files, dynamic Web pages, SQL stored procedures, J2EE Web deployment descriptors, EGL scripts, and many more), review your project tasks, debugging your code, configuring your data sources and much more.

The appearance of the Workbench window (the views which are shown, their position in the Workbench, and the toolbar and menu options which are available) is defined by the *Perspective*. RAD has a number of built-in perspectives, but you can also define your own customized perspectives. You switch between different perspectives during your work (for example, if you finish developing a Java class and start debugging the application, you switch from the Java perspective to the debug perspective). The perspectives we use when going through our examples are:

- Data perspective: Shows the Database Explorer and Data Definition views which allow you to connect to existing databases and work locally with their relational data objects.

Also includes the DB Output view which displays messages and results that are related to the database objects you work with.

- J2EE perspective: Contains Workbench views that you can use when developing resources for J2EE enterprise applications, EJB modules, Web modules, application client modules, and connector projects or modules. One of these views is the Project Explorer that provides an integrated view of your projects, grouped by type, and their artifacts related to J2EE. When you perform Web Services development, you will usually work with this perspective.
- Web perspective: Combines views and editors that assist you with Web application and Web services development. This is the perspective in which you typically edit Web project resources, such as HTML and JSP files, and deployment descriptors.
- Java perspective: Is designed for working with Java code. It contains the Package Explorer view which lets you browse easily through your Java code, the Outline view providing a structured view of your Java class, and a Java type hierarchy browser.
- Debug perspective: Provides convenient access to debugging information. You can define breakpoints in the source editor, manage threads in the Debug view, view console output and the run-time content of variables and manage Application Server configurations in the Server view.

The RAD *workspace* is a directory on your local file system which is used by RAD to store the following information:

- RAD environment metadata, such as configuration information and temporary files.
- All projects that you've created as part of the development process, including the source code, project definitions, config files, and generated files such as Java class files.

Resources that are modified and saved are reflected on the local file system. You can have many workspaces on your local file system to contain different projects that you are working on or differing versions. Each of these workspaces may be configured differently, since they will have their own copy of metadata that will have configuration data for that workspace.

You put your development output into different projects. The number and type of projects you use depends on the work items you create as part of your development activities. We use the following project types in our examples:

- Simple project: Intended for projects which do not follow a specific structure. We use this project type to store our local database definitions (for example, when we create a DB2 SQL stored procedure).
- Dynamic Web project: Used for all types of J2EE-based Web applications, that is, applications which result in a Web application archive (WAR) structure; this kind of Web application usually contains Java Servlets, JSPs, Web Services, Java libraries, static content and supporting metadata. We create such projects when we write Web Services.
- Portlet project: A portlet project, like a dynamic Web project, is a J2EE Web application. The main difference between a portlet project and a Web project is that the portlet project has an extra deployment descriptor, portlet.xml, in the WEB-INF directory.
- Enterprise Application project: An enterprise application project contains the hierarchy of resources that are required to deploy a J2EE enterprise application, often referred to as an EAR file. Our Web projects are bundled together with enterprise application projects so that we can deploy the projects as EAR files.

RAD database connectivity

RAD includes facilities to manage a number of aspects of the integration of relational databases into the development process. The database products supported are IBM DB2

Universal Database™, IBM Cloudscape, IBM Informix Dynamic Server, Microsoft SQL Server, Oracle, and Sybase.

Database connections

RAD allows to explore the structure of an existing external database (schemas, tables, views, stored procedures, user-defined functions). A wizard helps to create a connection to a local or remote database which is stored permanently in the user's workspace. Figure 5-14 shows the Database Explorer view which lists all stored database connections and allows to browse the database structure, down to the table column level. RAD imports the structure of databases utilizing the JDBC database metadata API. You can copy the complete database structure to a local project to modify and update it.

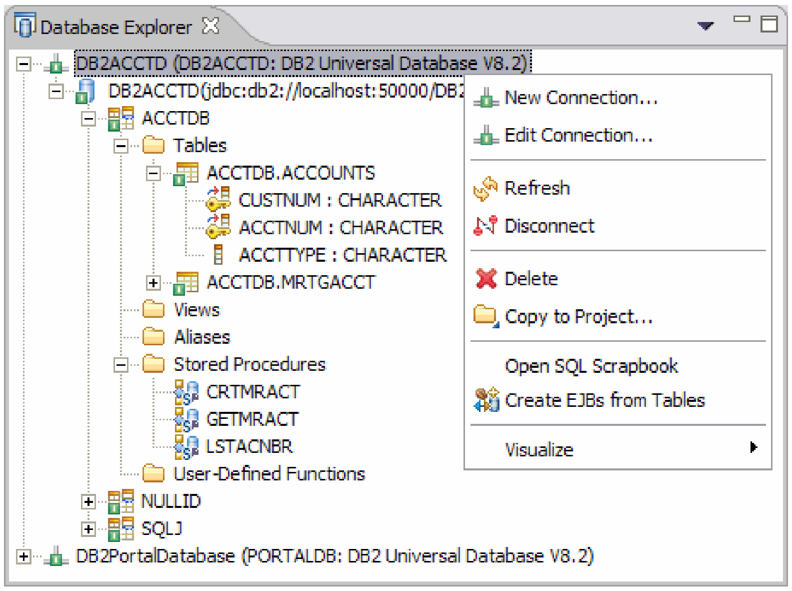


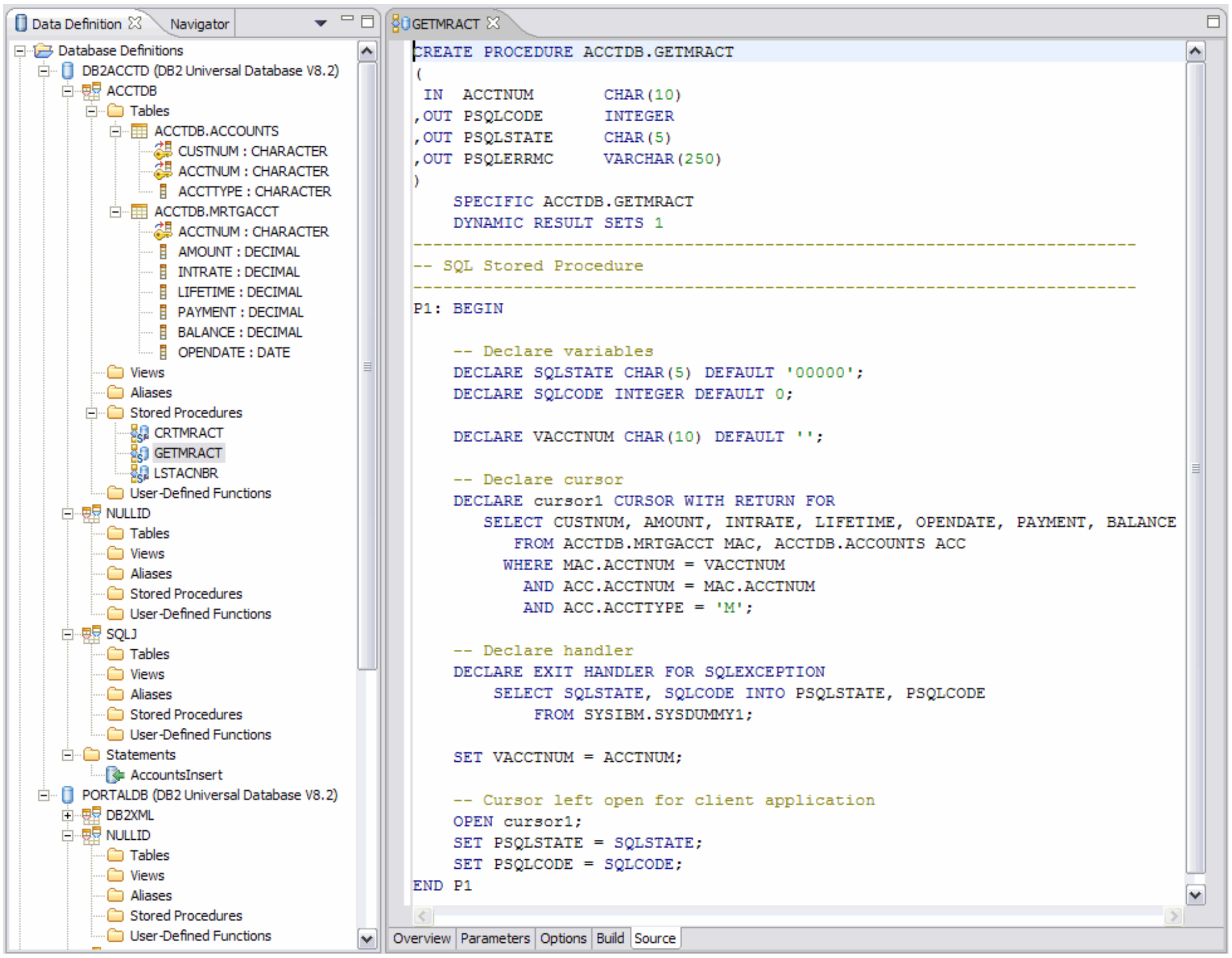
Figure 5-14 RAD Database Explorer view

Structural database changes

RAD contains wizards and editors to create or modify database tables, columns, relationships and constraints of an existing database. The modifications are first stored in the local workspace and can be propagated to the database server. You can also create and edit stored procedures and user-defined functions.

Figure 5-15 on page 119 shows the Data Definition view which provides access to database objects and allows to add new database objects to existing databases. You can edit the database objects by using special editors, for example, the stored procedure editor which is also shown in Figure 5-15 on page 119. In addition, you can create and test SQL statements (to select or modify data within tables).

RAD stores the database metadata in the XML Metadata Interchange (XMI) format, and provides conversions between the XMI format and the SQL format, which is used when communicating with the database server.



```
CREATE PROCEDURE ACCTDB.GETMRACT
(
 IN  ACCTNUM       CHAR(10)
,OUT PSQLCODE      INTEGER
,OUT PSQLSTATE     CHAR(5)
,OUT PSQLERRMC     VARCHAR(250)
)
    SPECIFIC ACCTDB.GETMRACT
    DYNAMIC RESULT SETS 1
------------------------------------------------------------------------
-- SQL Stored Procedure
------------------------------------------------------------------------
P1: BEGIN

    -- Declare variables
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE SQLCODE INTEGER DEFAULT 0;

    DECLARE VACCTNUM CHAR(10) DEFAULT '';

    -- Declare cursor
    DECLARE cursor1 CURSOR WITH RETURN FOR
       SELECT CUSTNUM, AMOUNT, INTRATE, LIFETIME, OPENDATE, PAYMENT, BALANCE
          FROM ACCTDB.MRTGACCT MAC, ACCTDB.ACCOUNTS ACC
         WHERE MAC.ACCTNUM = VACCTNUM
           AND ACC.ACCTNUM = MAC.ACCTNUM
           AND ACC.ACCTTYPE = 'M';

    -- Declare handler
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        SELECT SQLSTATE, SQLCODE INTO PSQLSTATE, PSQLCODE
            FROM SYSIBM.SYSDUMMY1;

    SET VACCTNUM = ACCTNUM;

    -- Cursor left open for client application
    OPEN cursor1;
    SET PSQLSTATE = SQLSTATE;
    SET PSQLCODE = SQLCODE;
END P1
```

Figure 5-15 RAD stored procedure editor

These features allow developers to develop test databases and work with production databases as part of the overall development process. They could also be used by database administrators to manage database systems, although they may prefer to use dedicated tools provided by the vendor of their database systems.

Databases and Web Services

RAD includes a number of wizards which use the database metadata to speed up the development process of Web Services applications, for example, a wizard to create a DADX Web Services file based on a stored procedure definition, and a wizard to create a UDF wrapper for a DB2 Web Services client.

Tools for Web services development

RAD provides tools to assist you with the following aspects of Web services development:

- Service discovery. You can browse UDDI Business Registries or WSIL documents to locate existing Web Services for integration.
- Creation of new services. You can turn existing artifacts such as Java beans, Enterprise JavaBeans™ or HTTP URLs into Web Services. RAD maps methods of the beans to Web Service operations and creates a WSDL document containing the service interface and the service binding. In the following chapters we show how to create Web Service interfaces from existing Java beans, DB2 stored procedures and SQL queries.

- Build service-aware applications. RAD supports an easy integration of Web Services into your Java application or DB2 database. Wizards assist you in the creation of Java client proxies. The proxies can be used within your application to greatly simplify the client programming required to access a Web Service. RAD wizards also support the creation of DB2 wrapper user-defined functions which consume Web Services.

  RAD can create a sample Web application which uses the proxies mentioned above and adds a JSP-based Web interface to call the Web Services. You can use this Web application to create simple Web Service test frameworks.
- Testing facilities. Apart from the sample Web application, RAD includes the Universal Test Client, the Web Services Explorer and a TCP/IP monitor to assist you in the testing of Web Services.
- Service deployment. RAD supports the deployment of Web applications which contain Web Services into IBM WebSphere Application Server or Apache Tomcat environments. You can enable security for the Web Services using configuration tools.
- Publishing of services. RAD assists you when publishing Web Services to a UDDI Business Registry.

5.3 WebSphere Developer for zSeries

In this section we describe WebSphere Developer for zSeries® Version 6. This product provide a set of capabilities that help make traditional mainframe and Web developments more efficient. In addition, the product provides a set of tools that leverage your ability to create Web service as part of SOA architecture for your existing COBOL and PL/I programs.

This section contains the following:

- Product overview
- Development tools

5.3.1 Product overview

WebSphere developer for zSeries consist of a common workbench and integrated set of tools that support end-to-end development of on demand applications and help to make traditional mainframe development, Web development, and integrated mixed workload or composite development faster and more efficient.

WebSphere Developer supports a broad range of developers with added flexibility and the ability to integrate with existing applications. It accelerates the development of:

- Dynamic Web applications including JAVA and JAVA 2 Enterprise Edition (J2EE).
- Traditional COBOL and PL/I applications.
- High level Enterprise Generation Language (EGL) applications.
- Web services to integrate these applications together.

The WebSphere Developer for zSeries application development workbench and tools provide these features and benefits:

- Deploys applications to multiple run times including WebSphere, CICS, IMS™, batch, and DB2 via stored procedures.
- Leverages existing skills to write Web or COBOL applications by using the high level Enterprise Generation Language.

- Improves the productivity for developers to create, maintain, debug and deploy traditional transactional and batch applications to the z/OS platform, while providing additional tools for assisting them in their Web integration efforts.
- Supports SOA and Web service creation, That can be deployed on WebSphere, CICS and IMS environments. Provides the option to publish COBOL and PL/I application as Web service as part of a SOA architecture.
- Helps developers create dynamic Web applications including support for J2EE, XML, and Web services technologies.
- Provides the option to create on demand system that integrate WebSphere software and traditional transactional environments, including CICS, IMS, and batch processing.

WebSphere developer for zSeries help make traditional mainframe development, Web development, and integrated mixed workload or composite development faster and more efficient.

5.3.2 Development tools

WebSphere Developer for zSeries Version 6 provides a set of development tools that intend to make the SOA development process easier and faster. This section provides a short description of these development tools.

z/OS application development tools

z/OS application development tools are interactive, workstation-based environment where you can develop mainframe applications in assembler language, COBOL, or PL/I. The environment gives you a seamless way to edit on the workstation and prepare output on the mainframe. The interaction with z/OS include the following steps:

- Create or modify the code in the z/OS LPEX editor. The editor maintains fixed record lengths, sequence numbers, and file locking (ISPF ENQ/DEQ) as appropriate.
- Validate the source using the syntax check function.
- Debug the code.
- Generate and customize JCL as needed.
- Transfer the source to the host, where z/OS tools submit the JCL or otherwise prepare the source, including pre-preparation steps for CICS and DB2.
- Inspect the results of code preparation.You can access z/OS data sets by way of a workstation

z/OS application developer tools provider you the option to access z/OS data sets in a workstation-like directory structure. This mean that you can process CLISTs and REXX scripts by editing it on the workstation, transferring it to the z/OS, and run it. Another feature is that you can view the output in the workstation environment.

XML services for the enterprise

XML Services for the Enterprise lets you easily adapt COBOL-based business applications so that they can process and produce XML messages. The tool provides a new kind of access to a called application, so that an Internet user, for example, can access an existing CICS application.The tool can also help you embed a COBOL application in a larger system that uses XML for data interchange.

New XML to COBOL Mapping tools allow you to enable existing COBOL applications to process and produce XML documents when the XML documents do not identically match the names or data types of COBOL data items. This situation can occur when the XML documents are derived from sources other than the target COBOL data structure. These tools are very useful when various parts of separate enterprise information systems (EIS) need to be merged and consolidated. The interfaces between various enterprise applications most likely do not precisely match.

Enterprise Generation Language (EGL)

Enterprise Generation Language (EGL) is a development environment and programming language that enables you to write full-function applications quickly, thereby freeing you to focus on the business problem your code is addressing, rather than on software technologies. You can use similar I/O statements to access different types of external data stores, whether those data stores are files, relational databases, or message queues. In addition, the details of Java and J2EE are hidden from you so that you can deliver enterprise data to browsers even if you have minimal experience with Web technologies.

After you code an EGL program, you generate it to create Java source, then EGL prepares the output to produce executable objects. EGL can also provide these services:

- Places the source on a deployment platform outside of the development platform
- Prepares the source on the deployment platform
- Sends status information from the deployment platform to the development platform, allowing you to check the results

EGL can also produce output that facilitates the final deployment of the executable objects.

An EGL program written for one target platform can be converted easily for use on another. The benefit is that you can code in response to current platform requirements, and many details of any future migration are handled for you. EGL can also produce multiple parts of an application system from the same source.

Service Flow Modeler

Flow Modeler is a multifunctional tool supporting modern application architectures and the transformation and reuse of existing application processes.

Service Flow Modeler enables the move towards service-oriented architecture (SOA). You can use Service Flow Modeler to perform the following tasks:

- Model a newly composed business service, or flow, by defining an interface and outlining execution steps.
- Capture existing EIS (screen or communication area) interfaces to implement steps in the flow.
- Map data between elements in the flow and the request and response messages used in its invocation.
- Expose business flows as a service or Web service.

Adapter services and Web services generated using the Service Flow Modeler can be deployed to multiple runtime environments.

Common Access Repository Manager (CARMA)

Common Access Repository Manager (CARMA) provides a generic interface to z/OS software configuration managers (SCMs), such as IBM Source Code Library Manager (SCLM) from WebSphere Developer for zSeries. When making use of CARMA, you can avoid writing specialized code for accessing SCMs and allow support for virtually any SCM.

For information about the prerequisites, refer to the product page at:

http://www-306.ibm.com/software/awdtools/devzseries

5.4 WebSphere Integration Developer

Business integration means integrating applications, data, and processes within an enterprise or amongst a set of enterprises. The challenge of this task and how it is met by WebSphere Integration Developer is discussed in this chapter.

In this section we describe the following topics:

- What business integration is
- WebSphere Integration Developer

5.4.1 What business integration is

Business integration means integrating applications, data and processes with an enterprise or amongst a set of enterprises.

The best way to present business integration is by using a scenario. Let us assume that your manager asked you to build a portal for the company customers. The portal have to provide access to ten of the company's critical applications and the data spread through all of the company's business units. In addition, you have also asked to add the company's business partners application to the portal. Another requirement is that the portal have to be available on the Web 24 hours a day. Figure 5-16 illustrates the challenge that you are facing when approaching this problem.

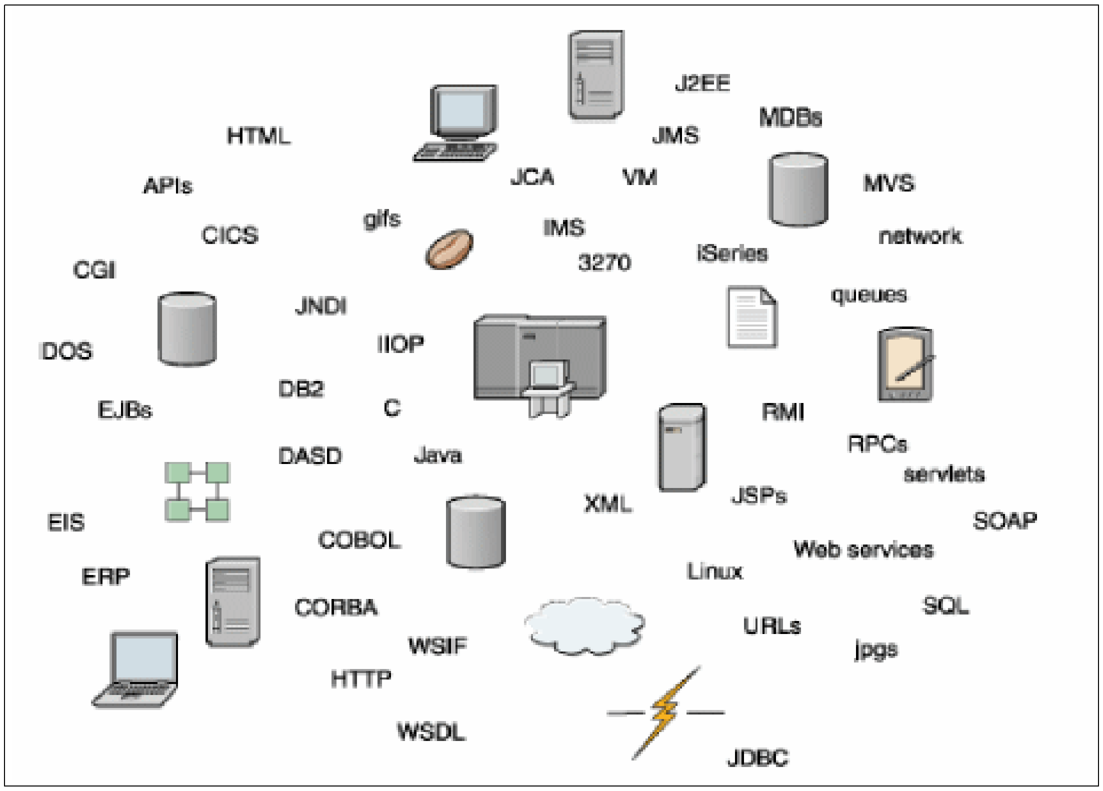


Figure 5-16 Business integration problem

It looks overwhelming, but not impossible. The most difficult problem will probably be the time and resources that you going to need to complete this mission.

There are two major problems that you will have to face approaching this problem are integration between business units within an enterprise (two or more different application on your company information system) and integration between enterprises (integration with the company's customer information systems). We describe these problems shortly.

Integration between business unit

Business units frequently find themselves collaborating today, creating a need for close integration of their applications.

Formerly autonomous business units are being integrated because technology allows them to be connected and because efficiency says they must operate in a more cooperative way to minimize overhead and maximize output. A common corporate goal also drives business units together. A marketing unit and a research-and-development unit both want to produce a profitable product. By integrating the knowledge of the market with the product development information, the odds of producing that successful product increase. Collaboration between business units also lets corporations leverage their many existing business applications by permitting their reuse in different business contexts. Integration between business units is easier than integration between enterprises because there is less security risk, and managing the interactions between the units should not be as difficult. The business units are probably using the same protocols, operating systems, and computer languages. In other words, it is a relatively homogenous environment. The key, however, is to have the right tools to quickly integrate the applications.

Integration between enterprises

The forces driving the integration of applications between business units also applies between enterprises, as partnerships or takeovers require shared data and processes. Technology enables enterprises to be linked in mutually beneficial areas. For example, an automobile manufacturer can set up an integrated process with a tire supplier so that when the stock of tires is low the supplier is notified automatically. Integration between enterprises is being driven by economic necessity. Having closer ties amongst corporations means less time delays and less overhead to get things done. These automated processes mean that people spend less time to process transactions between enterprises and travel costs and face-to-face meeting time can be reduced significantly. Administration costs are similarly reduced and turnaround time between notification, delivery and invoices is improved. But different enterprises have different histories. Their applications are coded in different languages on different platforms using different communication protocols. There are also greater security risks when working with different organizations. Whatever the benefits and even the necessity of integration between enterprises, the costs in development time can be significant without the right tools.

5.4.2 WebSphere Integration Developer

WebSphere Integration Developer has been designed as a complete integration development for building integrated applications.The WebSphere Integration Developer provides a layer of abstraction that separate the visually-presented components you work with from the underlying implementation.

Integrated applications can call applications on Enterprise Information Systems (EIS), involve business processes across departments or enterprises, and invoke applications locally or remotely written in a variety of languages and running on a variety of operating systems. For example, in Figure 5-17 on page 125 eMerged Corporation was created by merging DOM bank and M&M Discount brokers. The merger meant all of the above: applications on EIS systems, business processes, and applications within each former corporation had to be shared between the corporations and presented in a seamless way to the new set of customers. However, eMerged accomplished the task and, as shown in the following diagram, customers from both of the former separate businesses can access all their financial information online.

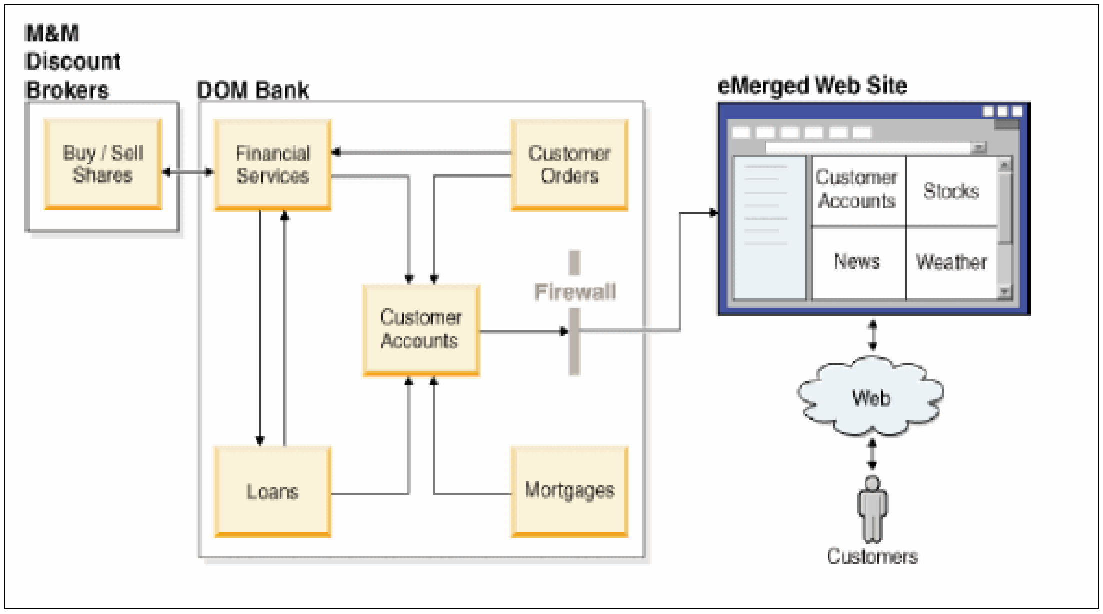


Figure 5-17 Integration sample

eMerged used WebSphere Integration Developer's tools to build the integrated applications for themselves and their customers. These tools present applications, including applications that exist remotely on EIS systems, and business processes as components. The components are created and assembled into other integrated applications (that is, applications created from a set of components) through visual editors. The visual editors present a layer of abstraction between the components and their implementations. A developer using the tools can create an integrated application without detailed knowledge of the underlying implementation of each component. The tools allow both a top-down design approach to building an integrated application, where the implementation for one or more components does not exist and is added later; or a bottom-up approach, where the components are already implemented and the developer assembles them by dragging and dropping them in a visual editor and then creates a logical flow amongst them by joining them with lines.

A debugging and test environment means full testing before your applications are deployed to a production server. Setting monitoring points lets you see how an application is used in real time in order to fine-tune it for optimal performance. WebSphere Integration Developer's tools are based on a service-oriented architecture. Components are services and an integrated application involving many components is a service. The services created comply to the leading, industry-wide standards. Business processes, which also become components, are similarly created with easy-to-use visual tools that comply to the industry-standard Business Process Execution Language (BPEL). WebSphere Integration Developer is available on both Windows and Linux platforms.

For complete information about the WebSphere Integration Developer refer to the product overview at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp

5.5 Enterprise Generation Language (EGL) and SOA

Enterprise Generation Language (EGL) is a simplified high level programming language that lets you write full-function applications quickly. It frees you to focus on the business problem rather than on complex software technologies. The details of middleware programming and Java/J2EE are hidden from you, so you can deliver enterprise data to browsers even if you have minimal experience with Web technologies.

EGL supports an easy way to connect to basically any kind of data server and allows the retrieval and manipulation of the underlying data source through a very powerful but still simple data access abstraction layer.

Due to the introduction of Web services support in a recent version of EGL, in combination with easy access to data sources, EGL makes a great development environment for developers who want create SOA based solutions in a Java/J2EE environment without the requirement to actually learn Java and J2EE technologies.

What is EGL?

IBM EGL is a procedural language used for the development of business application programs. The IBM EGL compiler outputs Java/J2SE or Java/J2EE code, as needed. With IBM EGL, one can develop business application programs with no user interface, a text user interface, or a multi-tier graphical Web interface.

Additionally, IBM EGL delivers the software re-use, ease of maintenance, and other features normally associated with object oriented programming languages by following the MVC design pattern. Because IBM EGL outputs Java J2EE code, IBM EGL benefits from and follows the design pattern of MVC and Java/J2EE. EGL will cause the programmer to organize elements of a business application into highly structured, highly reusable, easily maintained, and high performing program components.

IBM EGL is procedural, but it is also fourth generation. While IBM EGL supports all of the detailed programming capabilities one needs to execute in order to support business (procedural), it also has the higher level constructs that offer higher programmer productivity (fourth generation). In another sense, IBM EGL is also declarative. There are lists of properties that can be applied to various EGL components; these properties greatly enhance or configure the capability of these objects. Lastly, the term enterprise, as in Enterprise Generation Language, connotes that EGL can satisfy programming requirements across the entire enterprise. For example, with EGL, one can deliver intranet, extranet, and Internet applications, including Web services.

EGL provides a simplified approach to application development that is based on these simple principles:

- **Simplifying the specification:** EGL provides an easy to learn programming paradigm that is abstracted to a level that is independent from the underlying technology. Developers are shielded from the complexities of a variety of supported runtime environments. This results in a reduced training costs and a great improvement in productivity.
- **Code generation:** High productivity comes from the ability to generate the technology neutral specifications (EGL) or logic into optimized code for the target runtime platform. This results in less code that is written by the business oriented developer and in turn a reduced number of bugs in the application.
- **EGL Based Debugging:** Source level debugging is provided in the technology neutral specification (EGL) without having to generate the target platform code. This provides complete, end-to-end isolation from the complexity of the underlying technology platform.

Benefits of EGL

Many companies are under business pressure to quickly roll out new systems with the overall strategy to adopt the emerging J2EE and Web Services standards because of the obvious benefits of these technologies.

However, the pool of available developers, although extremely valuable because of their expertise in the business domain and their comfort in understanding the business requirements and how to implement them, cannot be simply re-trained in Java and J2EE. The costs of such training have been estimated to be in the order of well over $20,000 per developer and the time required to reach a level of proficiency in new technologies may not be compatible with the business pressures. Considering the degree of complexity in the new technologies and the relative inexperience of the developer pool, the results may not be ideal.

As a development environment, EGL can address many of today's development challenges. Integrated into the IBM/Rational development tool offerings, EGL can be the enabling technology that breaks through these challenges. It can allow you to leverage your current business-domain knowledgeable staff to use the latest technologies with minimal costs and effort. The result will allow your company to be more flexible and responsive to new business opportunities.

Who should consider using EGL?

Simply put - developers who will benefit most from the EGL technology are developers who need to solve business problems, not technology problems. If your developers fall into any of the following categories, they would probably be an excellent candidate for adopting EGL:

- Developers who need higher productivity.
- Developers who need to deploy to diverse platforms.
- "Business Oriented" Developers:
  - **Database Developers:** EGL takes the pain out of having to learn the database manipulation language and code the Create, Read, Update, Delete (CRUD) functionality by simply doing it for you.
  - **Informix 4GL Developers:** As a new capability in the portfolio, IBM is now enabling Informix 4GL based applications to easily migrate to a modern extensible IBM Rational development tools.
  - **Visual Age Generator Developers:** IBM provides time tested and easy to use and highly automated migration capabilities that can bring your valued former or existing Visual Age based application to a modern development environment and to a modern set of runtime technologies.
  - **Visual Basic Developers:** EGL offers similar but more powerful development efficiencies for the less technically skilled developers as does VB.
  - **Other 4GL Developers (Oracle Forms, etc…):** A community of IBM Business Partners can help you transform your former or existing 4GL applications to the IBM/Rational development platform with EGL.
  - **RPG Developers:** EGL offers a procedural language that is familiar to RPG developers. This will enable developers to move to a modern platform with minimal training costs while reaping the benefits of the latest technologies.
  - **COBOL/PLI Developers:** By generating COBOL from EGL, your COBOL developers can move to a new platform that leverages the latest technologies. Moving to EGL within the IBM/Rational development tools will free developers that have been trapped in earlier platforms but who can contribute greatly to new projects with their business domain expertise.

5.5.1 Application development with EGL

This section describes elements of EGL that are important to developing applications.

EGL language

The EGL language is a full featured, procedural language that abstracts out the details of a target technology from developers.

EGL has verbs like “get” that simplify the programming model by providing a consistent specification to a variety of target data sources. For example, a “get” statement can refer to records in a database or messages in a message queue. Developers are not required to learn and code technology dependent database manager or message oriented middleware programming.

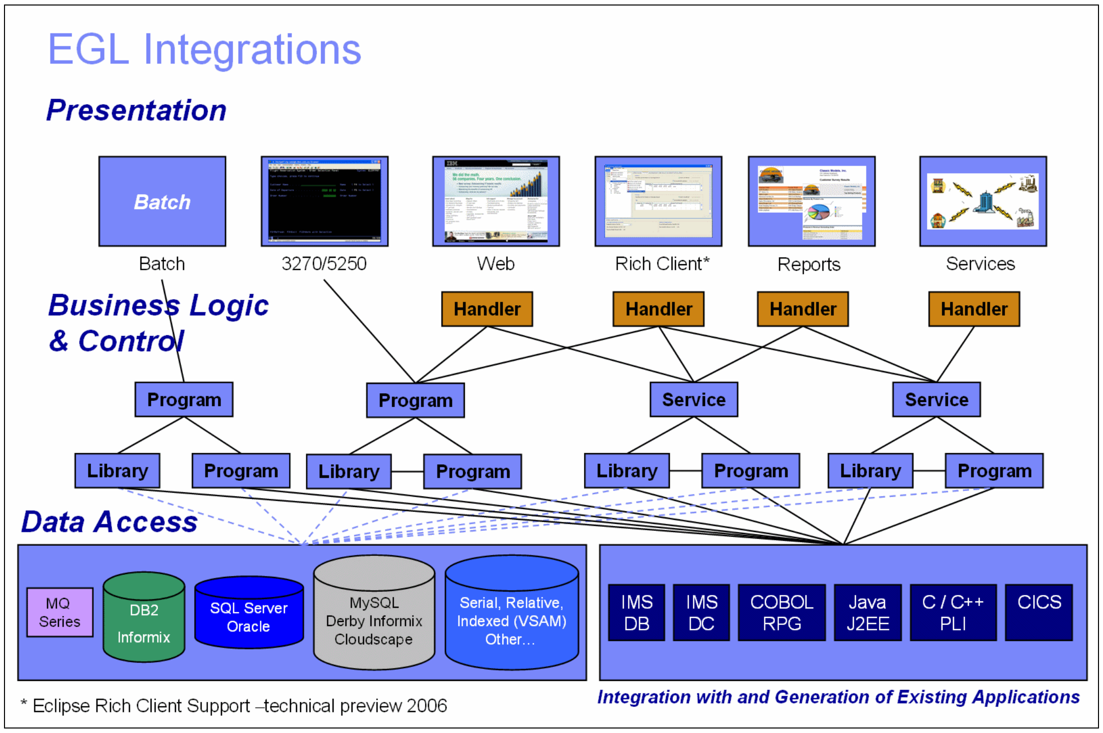


Figure 5-18 EGL integrates easily with different deployment environments

Writing your applications in EGL can also protect your development investment. The abstracted language can be cast or generated into any other language. Currently, EGL can generate Java or COBOL. As technology changes and evolves, your investment is protected by having the ability to re-generate into new target platform that have been improved or to entirely new platforms – without the need to modify your application.

EGL Libraries

EGL has a construct called a Library. An EGL Library is simply a file that includes EGL code. EGL libraries provide the application developer with the ability to easily decouple the business logic from other application code. EGL Libraries provide a variety of entry points – one per function. These functions can be called from other functions in other Libraries or from EGL code in EGL Programs or EGL Page Handlers.

EGL Programs

EGL Programs can also be used to package up business logic, but with a single entry point.

EGL Page Handler

In an EGL based Web application, every page will have a “shadow” page handler. The EGL page handler controls a user's run-time interaction with a Web page. Specifically, the page handler provides data and services to the JSP that displays the page. The page handler itself includes variables and the following kinds of logic:

- An OnPageLoad function, which is invoked the first time that the JSP renders the Web page
- A set of event handlers, each of which is invoked in response to a specific user action (specifically, by the user clicking a button or link)
- Optionally, validation functions that are used to validate Web-page input fields
- Private functions that can be invoked only by other page-handler functions

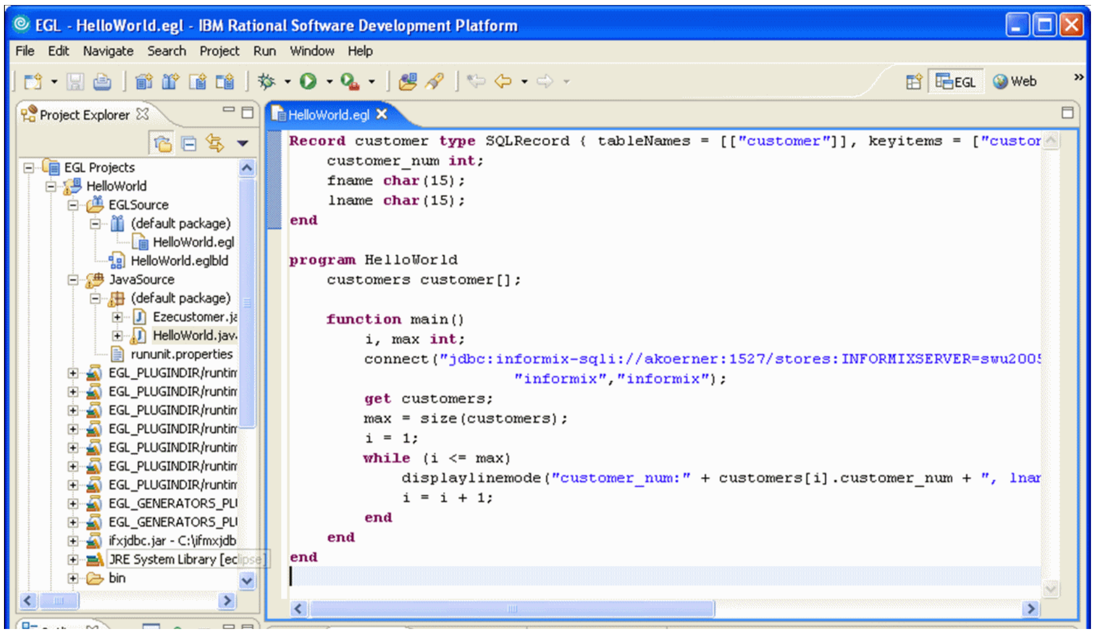
It is considered a best practice that the page handler should have no logic. It implements the controller component of the MVC model. Although the page handler might include lightweight data validations such as range checks, it's best to invoke other programs or functions to perform complex business logic so that you follow MVC principles.

Database connectivity with EGL

Accessing data from databases can sometimes be challenging to developers whose primary objective is to provide their users with the information that is optimal for them to make business decisions. To be able to access data, a developer needs to:

- Connect to a database
- Know and use the database schema
- Be proficient in SQL in order to get the appropriate data
- Provide the primitive functions to perform the basic CRUD database tasks
- Provide a test environment to efficiently test your application

EGL provides capabilities that make this task very easy for that business oriented developer.



```
Record customer type SQLRecord { tableNames = [["customer"]], keyitems = ["custor
    customer_num int;
    fname char(15);
    lname char(15);
end

program HelloWorld
    customers customer[];

    function main()
        i, max int;
        connect("jdbc:informix-sqli://akoerner:1527/stores:INFORMIXSERVER=swu2005
                        "informix","informix");
        get customers;
        max = size(customers);
        i = 1;
        while (i <= max)
            displaylinemode("customer_num:" + customers[i].customer_num + ", lnar
            i = i + 1;
        end
    end
end
```

Figure 5-19 A simple EGL program accessing an IBM data service (Informix IDS) via JDBC

Connectivity: Wizards will take these developers through a step by step process of defining connectivity.

Database Schema: If you are using an already existing database, EGL provides an easy to use import capability that will make the schema structure available to your application.

SQL Coding: EGL provides the generation of SQL statements based on your EGL code. You then have the option to use the SQL that was generated or for those power SQL users, you can alter the generated SQL to suit your needs.

Primitive functions: The EGL generation engine will automatically generate the typical CRUD – Create, Read, Update, and Deleted functions that are the workhorse functions for database driven applications.

Test capabilities: The IBM/Rational development tools have a test environment that eliminates the complexities that are associated with deploying and running your application in complex target platforms.

5.5.2 EGL and Web services support

Since version 6.0.1 of the Rational Software Development Platform (SDP), EGL offers a simple but powerful way of supporting Web services. You can write EGL applications which can provide Web services and/or consume Web services.

The current Web services support in EGL relies on the WebSphere Application Server Web services runtime framework, but future releases of EGL will likely support all standard J2EE application servers in combination with standard Web services runtime frameworks (like for instance Apache's Axis).

The combination of easy data server access in EGL and the included Web services support makes EGL a very interesting alternative to coding SOA applications in Java, especially for developers who have a non-Java background.

Since EGL is an already established conversion path for VisualAge® Generator and Informix 4GL applications, the recently added EGL Web services support allows those customers to easily integrate their earlier or existing applications into a modern SOA framework by just converting their existing applications into EGL.

In the next two sections we will show how easily EGL can be utilized to provide and consume Web services on top of an IBM data server.

EGL Web service providing

Before we start on the details on how to develop an EGL based Web service, one should mention that EGL actually supports two kind of services:

- EGL Service - a type of service for applications written entirely in EGL
- EGL Web service - a type of service that is created using EGL, but can be accessed from both EGL and non-EGL clients

This section focuses only on EGL Web services which can be also called from any standards compliant Web service client.

The high-level steps to create an EGL Web service are:

1. Configure your project for Web services: In this first step you set project and Workspace properties to enable Web Service development and testing.
   a. Specify Workspace Capabilities to support Web services development
   b. Specify Web service-specific Build Options

2. Define the Web Service EGL Part: Here you will create a new EGL file of type Service
3. Code the Web Service Business Logic: In this new Service EGL file, you will add EGL functions and variables (statements), that perform the service (for example, business logic) required by your application
4. Generate the Web Service: After you've finished coding you will save and generate Java for your Service
5. Optional: Test the Web Service interactively: You can then test your Web service interactively using the Rational SDP Web service testing tools. If there are logic issues, you can return to step3 and fix them.
6. Optional: Create the Web service Binding Library: When you are satisfied with your logic/testing results, you will create a Web Service Binding Library, which contains entries that allow you to deploy and call your new Web service from an EGL client (a Pagehandler file, program, library or some other Service Part)
7. Optional: Create a JSP page to test the Web service as a client: Finally, you can create a page and EGL Pagehandler to call the Web service from an Internet application.

A simple EGL Web service providing example

To better understand EGL Web services, let us look at the simple example in Example 5-1.

The task is to provide a Web service called getAllCustomers which returns a list of all banking customers from the customer table in the underlying database. The service has one parameter which is actually being used to return the customer record entries and one return value which contains the actual number of customer records returned.

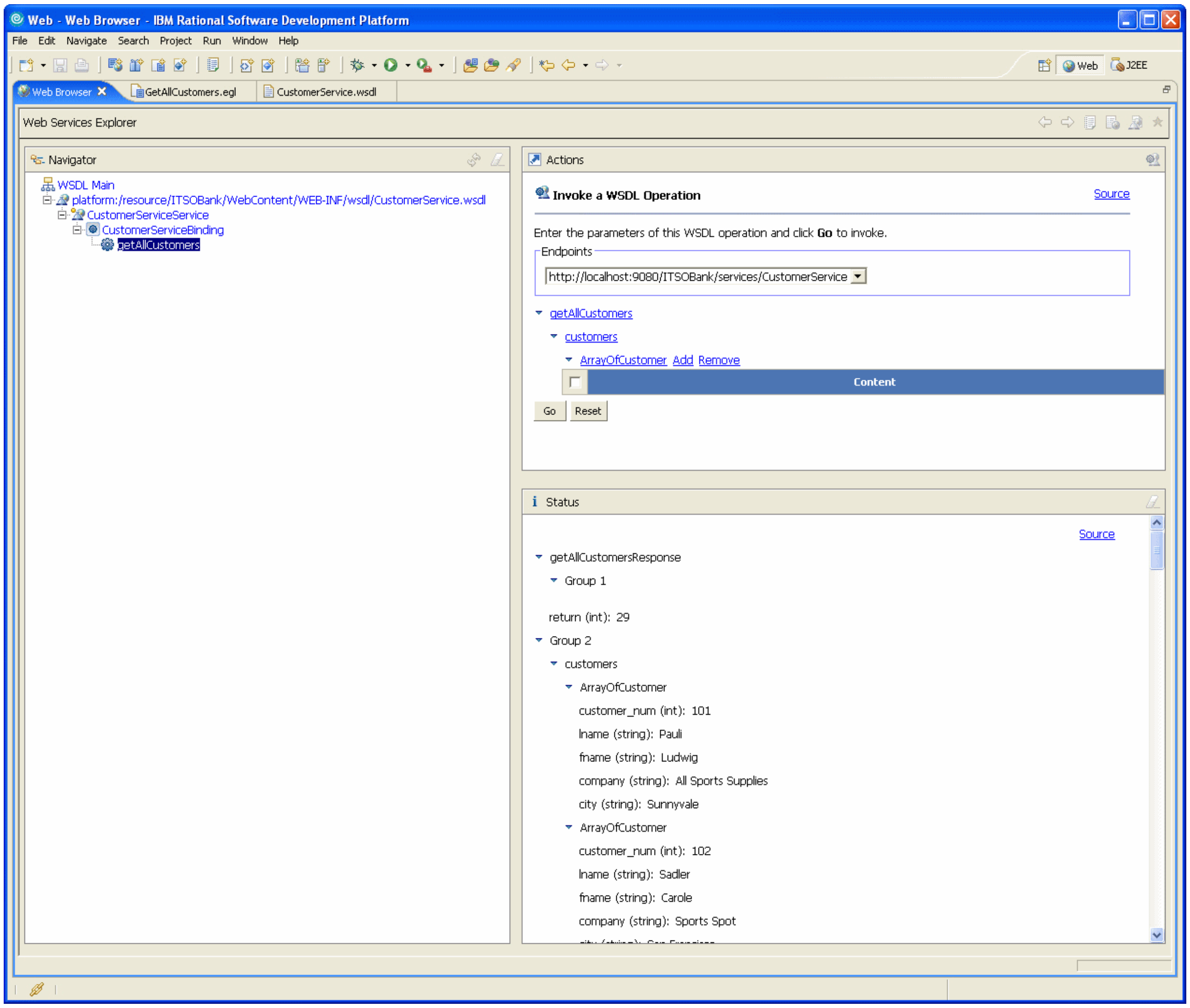
Example 5-1 A simple EGL Web services providing example

```
// EGL Web service

package EGLWebServices;

record Customer type SQLRecord
                     {tableNames = [["customer"]], keyItems = ["customer_num"]}
   customer_num int;
   lname char(15);
   fname char(15);
   company char(20);
   city char(15);
end

Service CustomerService

   function getAllCustomers(customers Customer[]) returns(int)
      get customers;
      return (size(customers));
   end

end
```

After coding and generating the EGL service part (Example 5-1), you will notice that the EGL code generator already generated the necessary WSDL file for the newly developed Web service into the *WebContent/WEB-INF/wsdl* folder.

In order to test the new EGL Web service within the Rational SDP first of all make sure that the integrated WebSphere Application Server test environment is up and running. In the next step simply navigate to the Web Services/Services/CustomerServiceService folder. Then right-click the WSDL file and select Test with Web Services Explorer.

After selecting the getAllCustomers Web service operation you see a result like in Figure 5-20.


Web - Web Browser - IBM Rational Software Development Platform
File Edit Navigate Search Project Run Window Help
Web Browser
GetAllCustomers.egl
CustomerService.wsdl
Web Services Explorer
Navigator
Actions
Status

WSDL Main
platform:/resource/ITSOBank/WebContent/WEB-INF/wsdl/CustomerService.wsdl
CustomerServiceService
CustomerServiceBinding
getAllCustomers

Invoke a WSDL Operation — Source

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints: http://localhost:9080/ITSOBank/services/CustomerService

getAllCustomers
customers
ArrayOfCustomer Add Remove
Content
Go Reset

```
Source
getAllCustomersResponse
  Group 1

  return (int): 29
Group 2
  customers
    ArrayOfCustomer
      customer_num (int): 101
      lname (string): Pauli
      fname (string): Ludwig
      company (string): All Sports Supplies
      city (string): Sunnyvale
    ArrayOfCustomer
      customer_num (int): 102
      lname (string): Sadler
      fname (string): Carole
      company (string): Sports Spot
```

Figure 5-20 Using the Web Services Explorer to test the EGL Web service

EGL Web service consuming

In order to consume a Web service in EGL the following basic three steps are necessary:

1. Obtain the WSDL file of the desired Web service and import it into an EGL Web project
2. Right-click the WSDL file within the project → **Generate EGL binding lib**
3. Call the Web service function from the generated EGL binding lib / interface

A simple EGL Web service consuming example

For this example we want to access one of the public accessible Web services like the CurrencyExchangeService provided by www.xmethods.net.

The finished application should be a Web based application which allows to enter two currency codes and returns the current exchange rate after call the xmethods.net Web service.

So in the first step we need to create a new EGL Web project and then import the downloaded WSDL file for the CurrencyExchangeService (Example 5-2).

Example 5-2 The CurrencyExchangeService.wsdl file

```
<?xml version="1.0"?>
<definitions name="CurrencyExchangeService"
targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
   <message name="getRateRequest">
      <part name="country1" type="xsd:string"/>
      <part name="country2" type="xsd:string"/>
   </message>
   <message name="getRateResponse">
      <part name="Result" type="xsd:float"/>
   </message>
   <portType name="CurrencyExchangePortType">
      <operation name="getRate">
         <input message="tns:getRateRequest" />
         <output message="tns:getRateResponse" />
      </operation>
   </portType>
   <binding name="CurrencyExchangeBinding" type="tns:CurrencyExchangePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getRate">
         <soap:operation soapAction=""/>
         <input >
            <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output >
            <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
   </binding>
   <service name="CurrencyExchangeService">
      <documentation>Returns the exchange rate between the two
currencies</documentation>
      <port name="CurrencyExchangePort" binding="tns:CurrencyExchangeBinding">
         <soap:address location="http://services.xmethods.net:80/soap"/>
      </port>
   </service>
</definitions>
```

In step two, we right-click the imported WSDL file and choose Create EGL Interfaces and Binding Library to generate the necessary files to actually call the Web service from within an EGL application (Example 5-3 on page 134 and Example 5-4 on page 134).

Example 5-3 CurrencyExchangeService.egl

```
package net.xmethods.www;

interface CurrencyExchangePortType{@wsdl {elementName="CurrencyExchangePortType",
namespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"}}

   function getRate(country1 string in, country2 string in)
returns(smallfloat){@wsdl {elementName="getRate"}};
end
```

Example 5-4 CurrencyExchangeService_ServiceBindingLib.egl

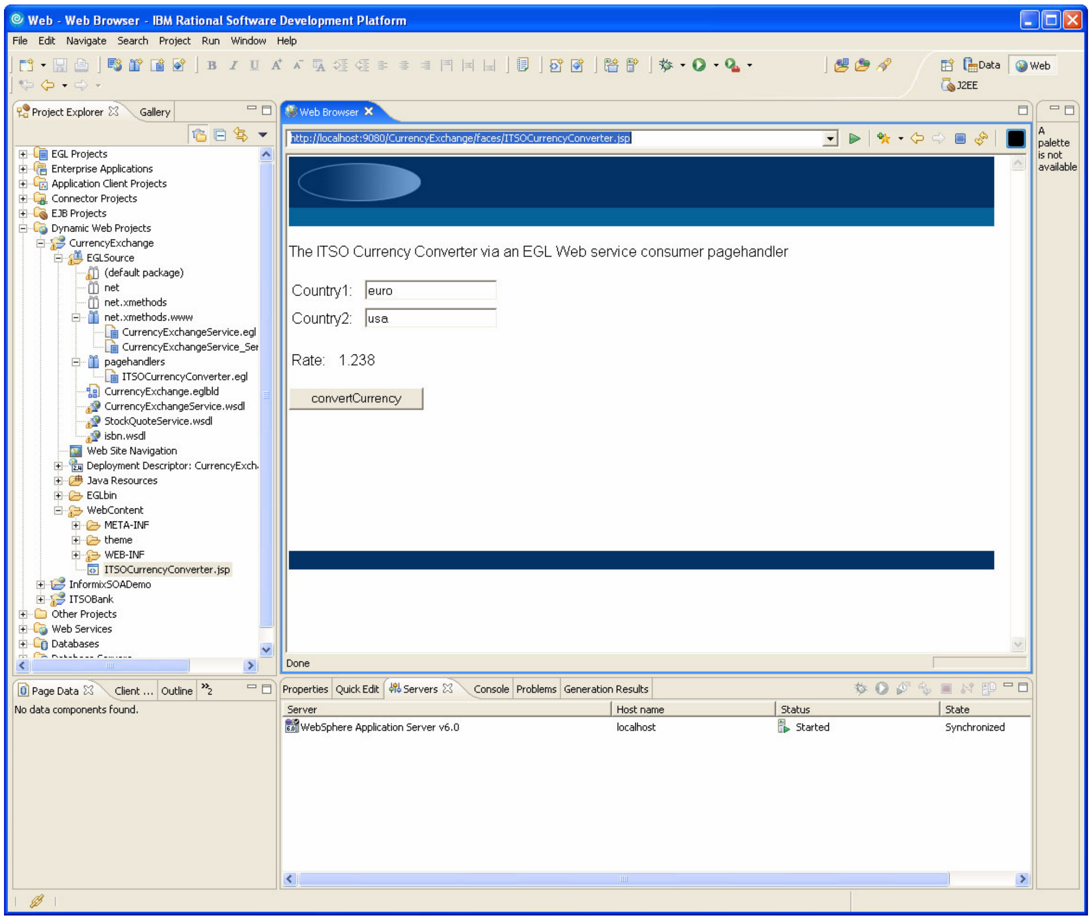
```
package net.xmethods.www;

library CurrencyExchangeService_ServiceBindingLib type ServiceBindingLibrary

CurrencyExchangePort CurrencyExchangePortType{@WebBinding {
   wsdlFile="WebContent/WEB-INF/wsdl/CurrencyExchangeService.wsdl",
   wsdlService="CurrencyExchangeService",
   wsdlPort="CurrencyExchangePort",
   endpoint="http://services.xmethods.net:80/soap"}};
end
```

In our last step we just need to write a simple EGL based Web application which refers to the two generated files (see above) and then calls the Web service function **getRate()** with two valid country codes (like usa and euro).

Example 5-5 The finalized EGL Web application which calls the getRate() Web service

```
package pagehandlers;

import net.xmethods.www.*;

PageHandler ITSOCurrencyConverter
   {onPageLoadFunction = "onPageLoad",
    view = "ITSOCurrencyConverter.jsp"}

   country1   string;
   country2   string;
   rate       float;

   Function onPageLoad()
      country1 = "usa";
      country2 = "euro";
   End

   Function convertCurrency()
      rate =
CurrencyExchangeService_ServiceBindingLib.CurrencyExchangePort.getRate(country1,
country2);
   end
End
```

After finalizing the EGL example above, just give it a test run by right-clicking on the ITSOCurrencyConverter.jsp file and then select **Run** → **Run on** Server. After a short while you should see the ITSO Currency Converter example application in the built-in Web browser of the Rational development tool. Just enter two valid values for the county codes (for example, euro and usa) and click the button. The button click will call the EGL function convertCurrency() which itself calls the external Web service (see also Figure 5-21).



http://localhost:9080/CurrencyExchange/faces/ITSOCurrencyConverter.jsp

The ITSO Currency Converter via an EGL Web service consumer pagehandler

Country1: euro

Country2: usa

Rate: 1.238

convertCurrency

Done

No data components found.

Server	Host name	Status	State
WebSphere Application Server v6.0	localhost	Started	Synchronized

Figure 5-21   The ITSO Currency Converter EGL example application

Part 3

IBM data servers and SOA access services

In Part 3 we discuss the contributions brought by the data servers to the SOA solution within these four chapters:

© Copyright IBM Corp. 2006. All rights reserved.

6

DB2 for z/OS and SOA

In this chapter we describe what the DB2 for z/OS brings to SOA. We look at both sides of DB2 involving Web services - a provider side and a consumer side.

We provide the introduction and the facilities to use Web Services Object Runtime Framework (WORF) in a DB2 for z/OS environment, which covers the benefits of exposing DB2 objects, such as SQL statements and stored procedures on the provider side, and also how to consume Web Services from DB2 using user-defined functions (UDFs) on the consumer side.

We describe the benefits for using stored procedures to provide Web services and the best way to use JCC to connect to the Database.

This chapter contains these topics:

- DB2 for z/OS and Web services
- DB2 for z/OS providing Web services
- Web services object runtime framework (WORF)
- How WORF processes a Web service
- Creating a DADX file
- Why use stored procedures?
- Connecting your services to DB2 for z/OS through JCC (JDBC)
- DB2 for z/OS consuming Web services

© Copyright IBM Corp. 2006. All rights reserved.

6.1 DB2 for z/OS and Web services

Since Version 7 DB2 for z/OS allows you to enable your DB2 data and applications as Web services like a Provider or Consumer.Enabling DB2 as a Web service provider allows you to create Web services on z/OS with your DB2 data and applications. Enabling DB2 as a Web service consumer allows you to receive Web service data in your DB2 applications.

Figure 6-1 shows how DB2 for z/OS can be a provider and consumer of Web services.

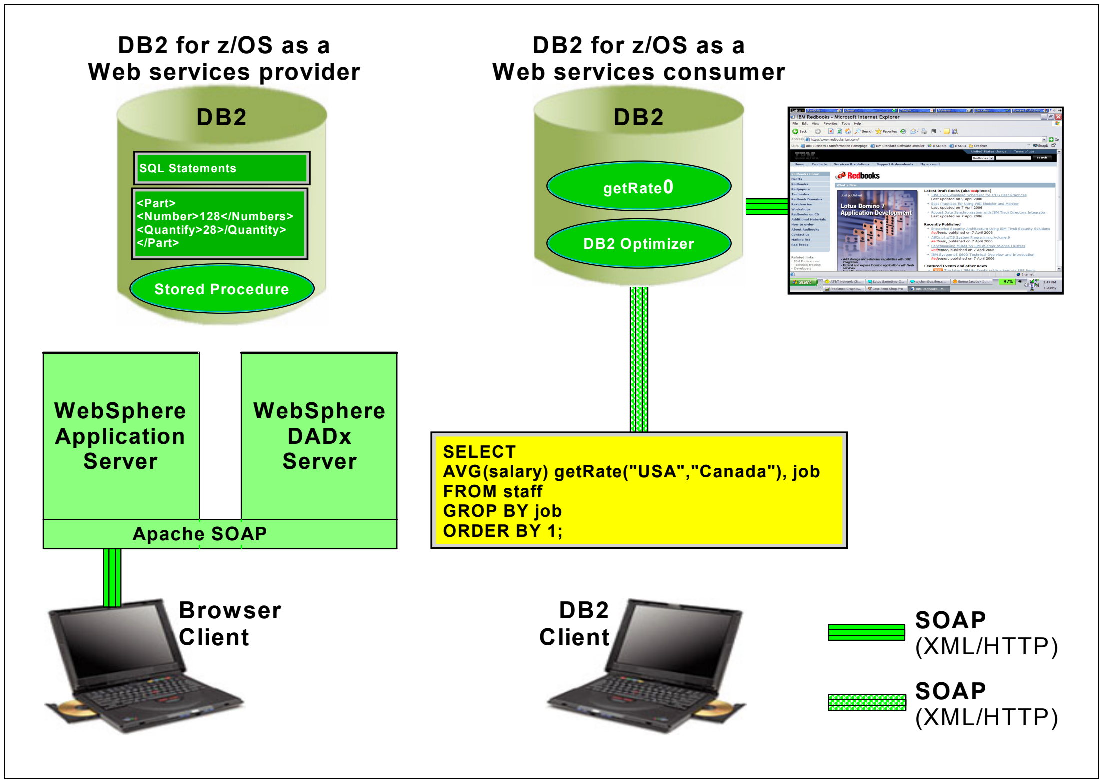

DB2 for z/OS as a Web services provider
DB2
SQL Statements
<Part> <Number>128</Numbers> <Quantity>28>/Quantity> </Part>
Stored Procedure
WebSphere Application Server
WebSphere DADx Server
Apache SOAP
Browser Client
DB2 for z/OS as a Web services consumer
DB2
getRate()
DB2 Optimizer
SELECT AVG(salary) getRate("USA","Canada"), job FROM staff GROP BY job ORDER BY 1;
DB2 Client
SOAP (XML/HTTP)
SOAP (XML/HTTP)

Figure 6-1   DB2 for z/OS providing and consuming Web services

6.2 DB2 for z/OS providing Web services

In this section we describe how DB2 for z/OS provides data to Web services.

Figure 6-2 on page 141 shows a scenario with DB2 for z/OS as a provider, and the Web application is not local in z/OS.

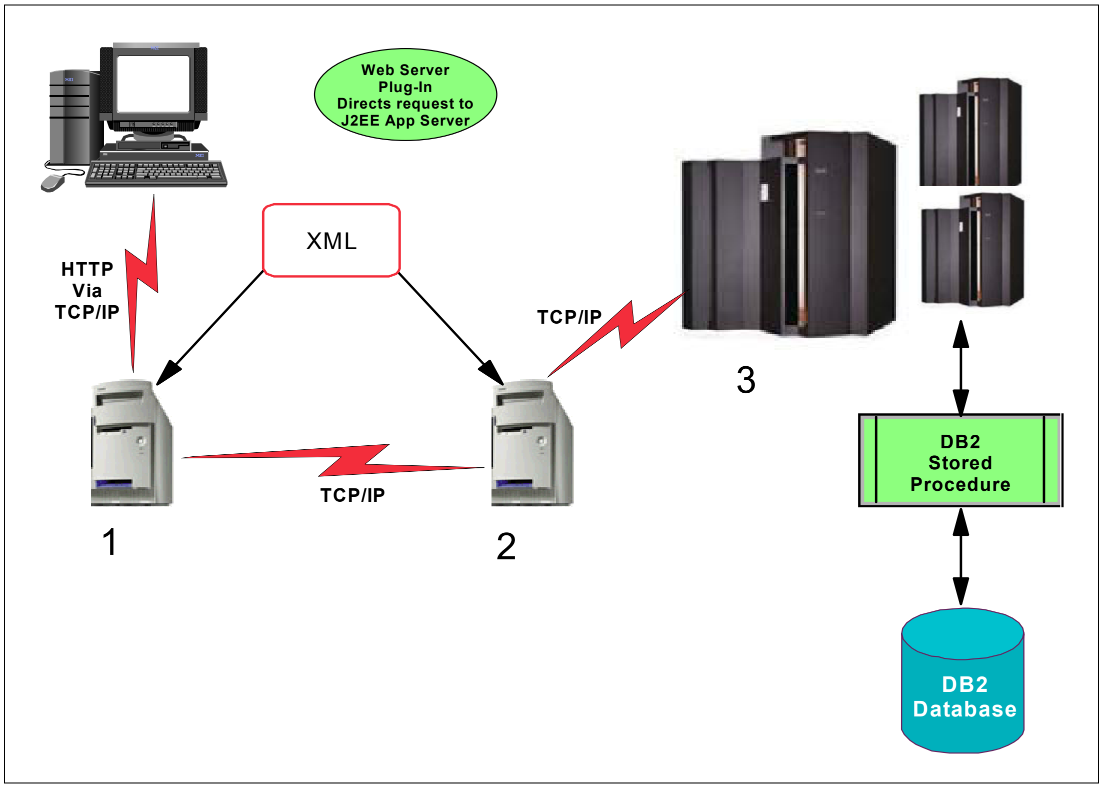


Figure 6-2 Web application out of z/OS

In reference to Figure 6-2:

1 - The server that runs the Web application

2 - The server that runs WORF and the DADX files

3 - The z/OS server that access the DB2 for z/OS

Figure 6-3 on page 142 shows a scenario with DB2 for z/OS as a provider, and the Web application is local in z/OS.

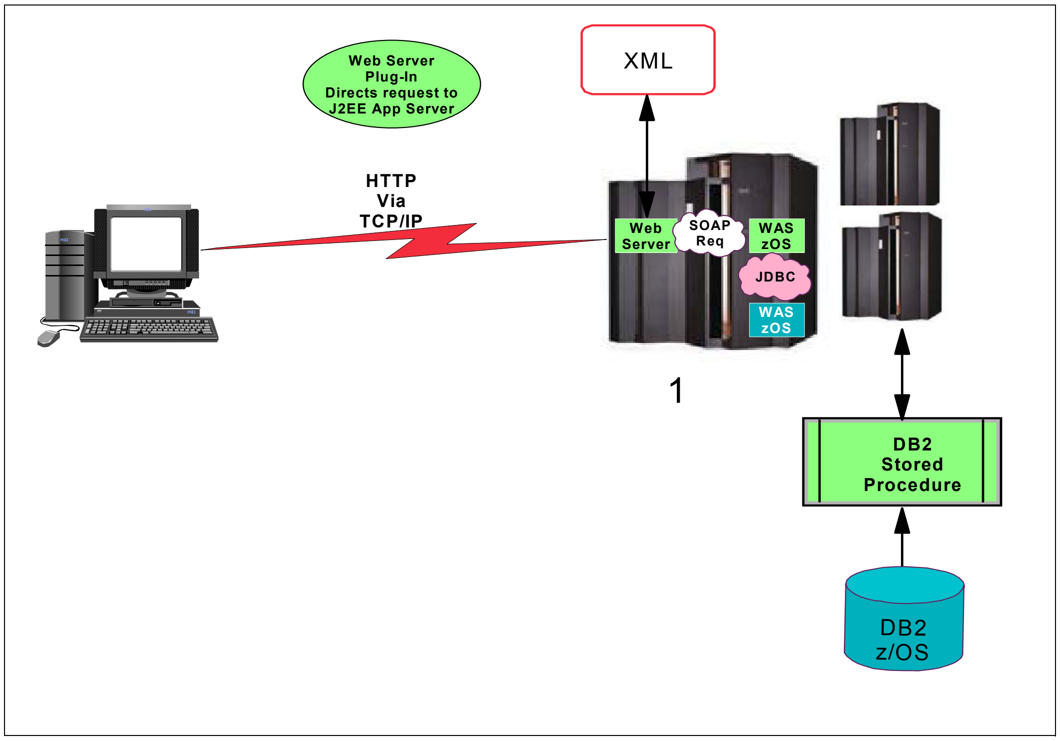


Figure 6-3 Web application inside z/OS

In reference to Figure 6-3:

1- The z/OS server that contains the Web application running in the WebSphere Application Server for z/OS and accessing DB2 for z/OS.

The WebSphere Application Server can be used local or out of z/OS to access Web services, in the next section we describe a feature in the WebSphere called Web Services Object Runtime Framework (WORF) that provides Web services with DB2.

6.3 Web services object runtime framework (WORF)

DB2 for z/OS allows you to enable your DB2 data and applications as Web services through the Web Services Object Runtime Framework (WORF).WORF is included in Rational Application Developer and is also available as a separate download in DB2. It needs to be run with WebSphere Application Server. It provides an environment to create XML-based Web services that access DB2.

Note: WORF is available on DB2 for z/OS since Version 7 with the PTF for APAR PQ91315.

WORF uses Apache SOAP 2.2 or later, Apache Axis 1.1 and the WebSphere SOAP engine. You can define a Web service in DB2 by using a Document Access Definition Extension (DADX). In the DADX file, you can define Web services based on SQL statements and stored procedures.Accessing DB2 for z/OS data using DADX requires a Java Web Application Server, regardless of platform. If the WebSphere Application Server is hosted on zSeries the native DB2 for zSeries JDBC driver is required. Alternately, the WebSphere environment can be hosted on UNIX, Windows, or Linux platforms and use the DB2 UDB UNIX, Windows, or Linux JDBC driver via JDBC using DRDA®.

The Web services that are created from a DADX file are called DADX Web services or DB2 Web services. A Java properties file describes for each DADX environment how to identify the DB2 JDBC driver details to connect to the DB2 database, including the user information. The operation is executed within the authorization scope of the defined user within the DADX descriptor.

Figure 6-4 shows a flow involving the user and providers using DADX files to access DB2.

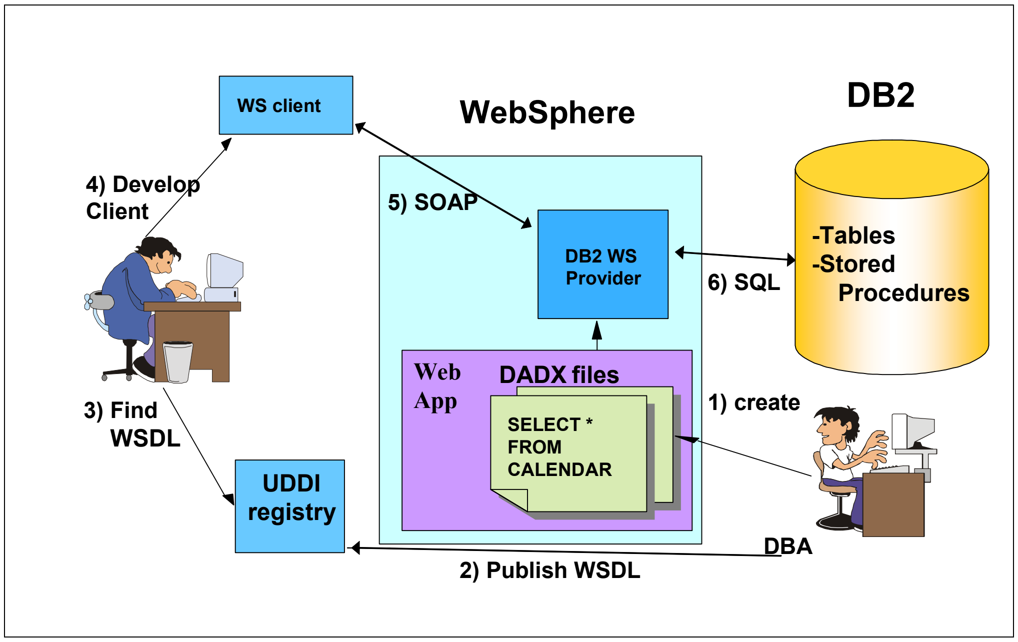


Figure 6-4 WORF's flow

6.3.1 What does WORF do?

Based on your definitions in the DADX file, WORF performs the following actions:

- Resource-based deployment and invocation.
- Automatic service redeployment at development time when defining resources change.
- HTTP GET and POST bindings in addition to SOAP.
- Handles the generation of any Web Services Definition Language (WSDL) and an XML description of all Web services via the Web Service Inspection Language (WSIL).
- Automatic documentation and test page generation.
- Provides DB2 JDBC connection information for the target database.
- Can be easily generated via WSAD/RAD tooling.
- Formats the result into XML, converting types as necessary.
- Provides a URL to invoke Web Service identifies the DADX file as well as the SQL Operation (Method) within it that is to be invoked.

6.3.2 WORF security

Security of a Web application or a Web Service application can consist of many parts. Many of these security aspects are known to a database administrator.

This section explains how this works in WebSphere. The things we focus on here are:

- Authentication
- Authorization
- Integrity/Confidentiality

Authentication means that you tell the service who you are. Of course that doesn't mean much without authentication where you supply a proof that shows that you really are who you claim you are. This proof could be a password or some security token.

Authorization deals with allowing or disallowing certain things for users. In database systems this is done with "GRANT" statements. We will explain how the user would do something like "GRANT SELECT, INSERT ON CALENDAR TO USER PHIL" in the Web Service context.

Integrity is a means of ensuring that the message has not been tempered with. By using Confidentiality we can make sure that nobody can read the message that is sent over insecure communication channels. One example to ensure confidentiality is to use encryption.

Note: We describe Web Services security in Chapter 3, "Web services and service-oriented architecture" on page 23.

DB2 Web Service provider security

The problems of an administrator who sets up security for the DB2 Web Service provider are the problem of identification and authentication, which we have already mentioned. We are going to solve this by requiring the user to have the client authenticate with HTTP authentication. HTTP authentication means that the HTTP request has to have an HTTP header field with the user ID and password. When you encounter a Web page in a browser that requires authentication you usually get a dialog box that lets you enter your user ID and password for this Web page. In the case of SOAP, the client program has to be modified to send the user ID and password.

We address authorization by using the J2EE (Java enterprise edition) mechanism of authorization for URLs. Since all of the Web Service requests are based on sending a message to a certain URL, we can configure the Web application so that only certain users can send requests to a certain URL. A URL could be either one DADX or a whole group of DADX files.

There is one last problem of mapping the users that authenticate with WebSphere to a database user that executes the statements in a DADX. Since our runtime cannot determine the user ID and password that is used in HTTP authentication, we cannot use this to connect to the database. In some cases this is not even realistic if the application server users are different from database users. An example is the case that the application server and database server run on different machines and both use the operating system as a user registry. Instead, on a group (that contains multiple DADX files) you can specify one user ID and password. This user will be used to execute all SQL statements in DADX in that group. If you want to distinguish users who execute SQL, you can create a separate group, such as a group for users in accounting and one for users in engineering.

Confidentiality and integrity can simply be solved by requiring the user to use HTTPS. This means that all network traffic is encrypted, and tampering with messages is also detected.

Note: For more information refer to:

`http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0404wollscheid`

6.4 How WORF processes a Web service

Figure 6-5 shows how WORF processes a Web service request.

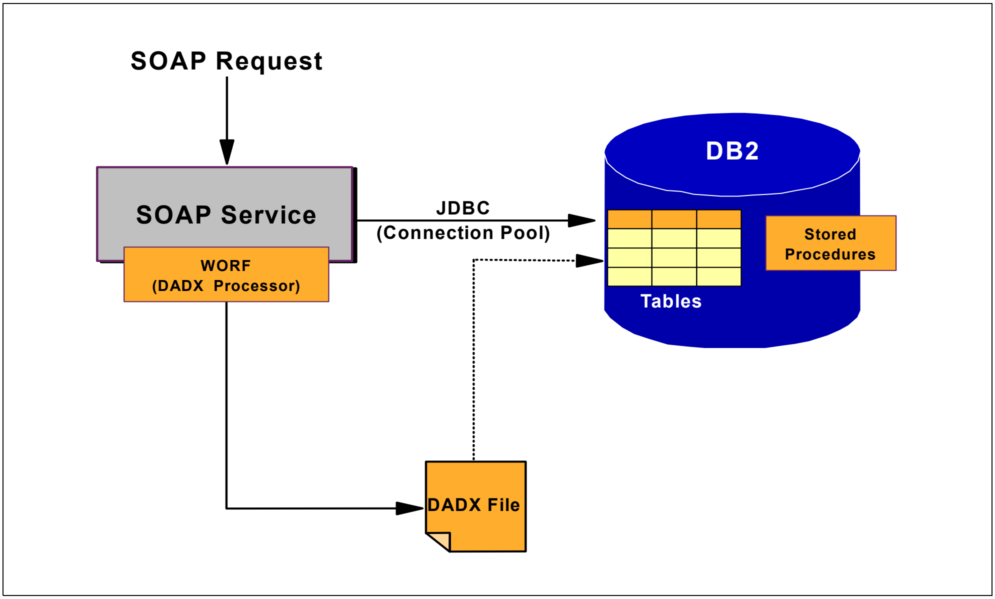


Figure 6-5 DADX file using WORF to access DB2

The Web services client sends a service request using the SOAP client. The WORF processes the received URL request and checks for a DADX and the requested action. The action can be TEST, XSD, or WSDL.

When WORF receives the request, it performs the following steps in response to a Web service request:

1. Load the DADX file specified in the URL or request from file systems or internal cache.
2. Check whether the request is for operation (SQL) or command (TEST, XSD, or WSDL).

If the DADX includes SQL operations:

- Replace parameters in the statement string inside the DADX file with requested values.
- Connect to DB2 and execute any SQL statements or SQL calls specified in DADX file.
- Format the result into XML.
- Return the response to the service requestor.

If the request is for a command, generate the required files, test pages, or other responses and return the response to the service requestor.

6.5 Creating a DADX file

A document access definition extension (DADX) file specifies how to create a Web service by using a set of operations that are defined by SQL statements. 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.

Table 6-1 shows what you can describe in the DADX file.

Table 6-1 XML elements used in the DADX files

Operations	Element	Description
SQL Operations	**<query>**	**Queries the database.**
	<update>	**Updates the database.**
	<call>	**Call stored procedures.**

Note: DADX files also can execute XML collections operations using elements <retrieveXML> and <storeXML>, it needs the installation of XML Extender. In this book, we are not considering using XML extender, because DB2 for z/OS V8 and mainly V9 support XML/SQL using publishing functions and Native XML. See Appendix B, "XML and DB2 for z/OS" on page 575 about XML support in z/OS.

In these example, we perform SQL operation to get CARS information. The operation listCars, lists information about all Cars in a Rental Car. To pass SQL parameters, use a colon prefix as shown in the Example 6-1.

Example 6-1 SQL query using DADX

```
<SQL_query>SELECT * FROM CARS WHERE CAR_NUM =:CARNUMBER</SQL_query>
```

To define a parameter, use the <parameter> element as shown in the Example 6-2.

Example 6-2 Defining parameter

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

The second operation, getEmpInfo, takes the employee number as its input parameter and returns employee information for that number. See Example 6-3 that shows the DADX file which contains the definition for both SQL operations.

Example 6-3 DADX file

```
<?xml version="1.0" encoding="UTF-8"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx">
<operation name="listDepartments">
<documentation> Lists all the departments in the DEPARTMENT table of
the SAMPLE database </documentation>
<query>
<SQL_query>SELECT * FROM department</SQL_query>
</query>
</operation>
<operation name="listSales">
<query>
<SQL_query>SELECT * FROM sales WHERE SALES_PERSON = :name</SQL_query>
<parameter name="name" type="xsd:string"/>
</query>
</operation>
</DADX>.
```

Note: See Chapter 15, "Developing SOA access services" on page 401 for more information and examples developing DADX files.

6.6 Why use stored procedures?

When you expose DB2 data to Web services clients using DADX, consider embedding data access logic into DB2 stored procedures. Each DADX operation is currently limited to a single SQL statement and executes within a single unit of work. DB2 stored procedures provide a very powerful technique for creating an abstraction layer for DB2 data access. DB2 stored procedures can be created in various programming languages, including Java and the standard SQL procedure language.

Other stored procedures advantages using DB2 for z/OS:

- Reduce network traffic.
- Simplify development and maintenance.
- Remove client dependency on database design at the server.
- Allow for the ability to dynamically change application programs.
- Changed and refreshed code at the server.
- No need to change client code.
- Can continue to run client code while changes are made.
- Allow most of an application to exist at the server, not the client.
- Need less code to be globally changed at all client locations.
- Reusable code.
- Improve security.
- Eliminate the need for end user table authority.
- Move processing away from end users.
- Provide the ability to access and update data that is not stored in DB2, such as VSAM IMS data.

Note: Note that in DB2 Version 8, the Java stored procedure infrastructure has been rearchitected to improve performance.

6.6.1 Stored procedure as a Web service

Everything we have worked through so far has given us all we need to move on to the other features and functionality of DB2 Web services. In this section we show you how to expose a DB2 stored procedure as a Web service. This will include a simple stored procedure as well as a stored procedure which will require input variables and join a few tables together.

See Figure 6-6 for a graphical representation of how to call a stored procedure.

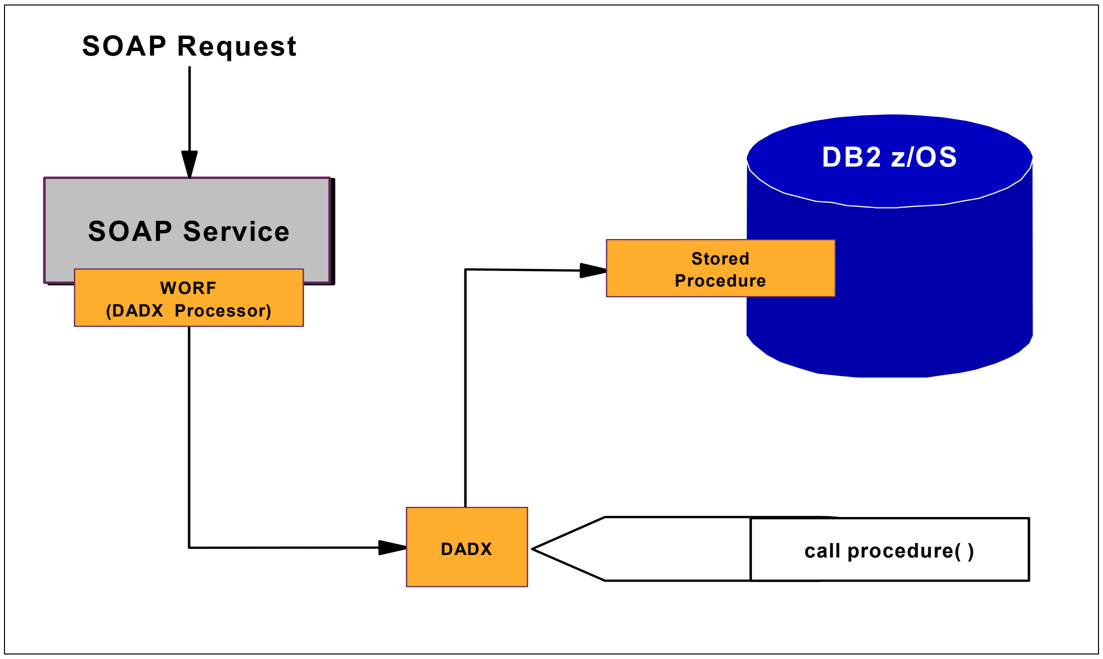


Figure 6-6 Calling a stored procedure

See Example 6-4 a short example to call a stored procedure using DADX file.

Example 6-4 Calling stored procedures

```
<?xml version="1.0" encoding="UTF-8"?>
<dadx:DADX xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx dadx.xsd">
    <dadx:operation name="StoredProcForSOA">
        <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
            <![CDATA[ ]]>
        </dadx:documentation>
        <dadx:call>
            <dadx:SQL_call>
                <![CDATA[
                CALL PAOLOR1.SPSOA2(:NUMBER, :DESC1, :DESC2)
                ]]>
            </dadx:SQL_call>
            <dadx:parameter name="NUMBER" type="xsd:int" kind="in"/>
            <dadx:parameter name="DESC1" type="xsd:string" kind="out"/>
            <dadx:parameter name="DESC2" type="xsd:string" kind="out"/>
        </dadx:call>
    </dadx:operation>
</dadx:DADX>
```

Example 6-5 shows the DDL to create the stored procedure used in the previous example in z/OS.

Example 6-5 DDL

```
CREATE PROCEDURE SPSOA2 ( IN NUMBER INTEGER,
                       OUT DESC1 CHAR(10),
                       OUT DESC2 CHAR(30) )
   RESULT SETS 1
   LANGUAGE SQL
   FENCED
   COLLID TEST
   WLM ENVIRONMENT DB8AWLM1
   RUN OPTIONS 'NOTEST(NONE,*,*,*)'
------------------------------------------------------------------------
-- SQL Stored Procedure
   -- NUMBER
   -- DESC1
   -- DESC2
------------------------------------------------------------------------
P1: BEGIN
   -- Declare variables
   DECLARE DESC1_TMP CHAR(10) DEFAULT ' ';
   DECLARE DESC2_TMP CHAR(30) DEFAULT ' ';

   -- Declare cursor
   DECLARE cursor1 CURSOR WITH RETURN FOR
      SELECT DESC1_SOA, DESC2_SOA
        FROM TBSOA
        WHERE NUM_SOA = NUMBER;

   -- Cursor left open for client application
   OPEN cursor1;
   SET DESC1 = DESC1_TMP;
   SET DESC2 = DESC2_TMP;
END P1
```

Result sets

Stored procedures can return one or more result sets. These result sets are included in the message, but you have to define its structure in the DADX file.

<result_set_metadata> - Metadata for a stored procedure result set must be defined explicitly in the DADX using the element.

At run-time, you obtain the metadata of the result set. The metadata must match the definition contained in the DADX file. Therefore, 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.

See Example 6-6 on page 149 that shows of a <result_set_metadata> element.

Example 6-6 <result_set_metada> element

```
<result_set_metadata name="listOrg" rowName="Org">
<column name="DEPTNUMB" type="SMALLINT" nullable="false"/>
<column name="DEPTNAME" type="VARCHAR" nullable="true"/>
```

```
<column name="MANAGER" type="SMALLINT" nullable="true"/>
<column name="DIVISION" type="VARCHAR" nullable="true"/>
<column name="LOCATION" type="VARCHAR" nullable="true"/>
</result_set_metadata>
<operation name="listOrg">
<documentation> Lists all the Organisations in the Org table of the SAMPLE
database </documentation>
<call>
<SQL_call>CALL <username>.org()</SQL_call>
<result_set name="Orgs" metadata="listOrg"/>
</call>
</operation>
```

Table 6-2 shows the main XML elements used in the Example 6-6.

Table 6-2 XML elements and its descriptions

XML element	Description
<result_set_metadata>	The result set metadata definitions are global to the DADX and must precede all of the operation definition elements.
<operation>	The operation element and its children specify the name of an operation, and the type of operation the Web service performs.
<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.
<documentation>	Document the operation, specifying a comment or statement about the purpose and content of the operation.
<call>	Call the stored procedure.
<SQL_call>	Specifies a call to the stored procedure.
<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.
<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.

Table 6-3 on page 150 shows the XML attributes for each element used in the previous example.

Table 6-3 XML attributes

XML element	XML attributes	Description
<result_set_metadata>	name	Identifies the root element for the result set.
	rowname	Set as the element name for each row of the result set.

XML element	XML attributes	Description
<column>	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.
</operation>	name	unique string that identifies the operation. The string must be unique within the DADX file.
<parameter>	name	The unique name of the parameter.
	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
<result_set>	name	unique identifier for the result sets in the SOAP response
	metadata	result set metadata definition in the DADX file. The identifier must refer to the name of an element.

Example 6-7 shows the DDL to create the Stored Procedure called in the previous Example 6-6 on page 149.

Example 6-7 DDL to create a stored procedure

```
CREATE PROCEDURE <username>.ORG ( )
SPECIFIC <username>.listORG
DYNAMIC RESULT SETS 1
------------------------------------------------------------------------
-- SQL Stored Procedure
------------------------------------------------------------------------
P1: BEGIN
-- Declare cursor
DECLARE cursor1 CURSOR WITH RETURN FOR
SELECT *
FROM ORG AS ORG;
```

```
-- Cursor left open for client application
OPEN cursor1;
END P1 @
```

Note: You can create this stored procedure using the DB2 Development Center. For detailed information about stored procedure on z/OS refer to DB2 for z/OS Stored Procedures: Give Them a CALL Through the Network, G24-7083.

Example 6-8 shows another DDL.

Example 6-8 DDL creating a stored procedure that does a tables join

```
CREATE PROCEDURE <username>.STAFFORG ( IN SALARY DECIMAL(7,2) )
SPECIFIC <username>.STAFFORG
DYNAMIC RESULT SETS 1
------------------------------------------------------------------------
-- SQL Stored Procedure
------------------------------------------------------------------------
P1: BEGIN
-- Declare cursor
DECLARE cursor1 CURSOR WITH RETURN FOR
SELECT STAFF.NAME, STAFF.JOB, ORG.DEPTNAME as DEPARTMENT,
ORG.DIVISION, ORG.LOCATION
FROM DB2DAS.STAFF AS STAFF, DB2DAS.ORG AS ORG
WHERE STAFF.ID = ORG.MANAGER AND STAFF.SALARY >
STAFFORG.SALARY
ORDER BY STAFF.NAME ASC;
-- Cursor left open for client application
OPEN cursor1;
END P1 @
```

6.7 Connecting your services to DB2 for z/OS through JCC (JDBC)

Java Database Connectivity (JDBC) and embedded SQL for Java (SQLJ) are the two standards-based Java application programming interfaces (APIs) supported by DB2.

6.7.1 Accessing DB2 for z/OS from Java Environment

JDBC is part of the J2EE standard. The Java 2 Platform, Enterprise Edition (J2EE) is a specification of the different technologies that are available to developers of enterprise Java applications. Typically these applications are server based, and may even include multiple servers of different types. This server focus makes it the architecture of interest for DB2 for z/OS developers.

JDBC defines the standard Application Programming Interface (API) for accessing relational database systems, such as DB2, from Java. This API is the fundamental building block for writing DB2 Java applications.

Both JDBC and SQLJ can be used to create Java applications that access DB2. DB2's support for JDBC allows Java applications to access local DB2 for z/OS data or remote server that supports Distributed Relational Database Architecture™ (DRDA).

Java applications use SQLJ for static SQL and JDBC for dynamic SQL to access any relational database. SQLJ applications exploit JDBC as an underlying foundation for tasks such as connecting to databases and handling SQL errors. This allows SQLJ to inter-operate with JDBC, making it possible for an application program to use JDBC and SQLJ within the same unit of work.

JDBC fundamentals

The JDBC API specification was developed by Sun Microsystems™ together with relational database providers, such as IBM and Oracle, to ensure the portability of Java applications across databases and platforms. There have been three major releases of the JDBC API specification to date. The most recent one, at the time of writing this publication, is the JDBC 3.0 specification. The final JDBC 3.0 specification was released in February 2002. However, a JDBC 4.0 specification was filed in 2003.

JDBC driver types

The role of the JDBC driver is to implement the objects, methods, and data types defined in the JDBC specification. There are currently four JDBC driver types defined in the JDBC 3.0 specification.

Type 1

Commonly referred to as the JDBC-ODBC bridge driver, type 1 drivers map the JDBC API to another data access API, such as ODBC. Type 1 JDBC drivers are limited in portability because of the fact that they are generally dependent on a native library. The overhead involved in having to use two APIs (JDBC and ODBC) degrades performance.

DB2 for z/OS does not supply a type 1 driver.

Type 2

Type 2 drivers are written partly in the Java programming language and partly in native code. These drivers use a native client library specific to the data source to which they connect.

Although portability is limited due to the platform-specific native code, the type 2 driver is currently the most commonly used driver type, as it provides the best performance when local to the database.

DB2 UDB for OS/390 and z/OS offers two type 2 drivers:

- IBM DB2 JDBC/SQLJ 2.0 Driver, also known as the IBM DB2 Legacy Driver
- IBM DB2 Universal Driver for SQLJ and JDBC

Type 3

A type 3 driver is a driver that is written in pure Java. With the type 3 driver, a Java client is used to communicate with a middleware server using a network-specific protocol. The middleware server translates the client request in order to access the data source.

DB2 for OS/390 and z/OS does not supply a type 3 driver.

Type 4

Type 4 drivers are written in pure Java and implement the database protocol for a specific data source. The client connects directly to the data source. DRDA is the protocol that is used when connecting to a DB2 system as a data source. The type 4 driver is fully portable since it is written purely in Java.

The IBM DB2 Universal Driver for SQLJ and JDBC type 4 connectivity.

Note: Additional information about JDBC driver types and the different scenarios when they can/should be used is available in the *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435.

IBM DB2 Legacy Driver

A DB2 JDBC type 2 driver, referred to as the DB2 JDBC Legacy type 2 driver, was introduced to DB2 UDB for OS/390 Version 6 via the maintenance stream in APAR PQ36011. DB2 UDB for z/OS and OS/390 Version 7 integrated the DB2 JDBC Legacy type 2 driver into the base code. DB2 UDB for z/OS Version 8 still ships the earlier driver for compatibility reasons, but V8 also ships the IBM DB2 Universal Driver for SQLJ and JDBC.

Attention: The DB2 JDBC Legacy Type 2 driver should not be confused with the JDBC type 2 driver provided with the IBM DB2 Universal Driver. They are not the same driver.

IBM DB2 Universal Driver for SQLJ and JDBC

The IBM DB2 Universal Driver for SQLJ and JDBC is a unique architecture-neutral JDBC driver that supports JDBC type 2 and JDBC type 4 connectivity as well as SQLJ for distributed and local access. The DB2 Universal Driver is not based on any previous DB2 JDBC drivers. It is a completely new driver that is not categorized by the conventional driver types. This driver is also known as the DB2 Universal Driver for Java Common Connectivity (JCC).

When the DB2 Universal Driver is loaded by a Java application, a single driver instance is created, providing JDBC type 2 and type 4 driver connectivity.Type 4 connectivity with the Universal Driver uses DRDA to connect to a database server (remote or local). This means that when using type 4 connectivity, Distributed Data Facility (DDF) will be used even when a Java program runs on the same z/OS system or logical partition (LPAR) as the target DB2 subsystem.

The Universal Driver also supports distributed transaction management (two-phase commit support) in the z/OS environment when type 2 or type 4 connectivity is used. However, there is a difference when using type 2 or type 4 connectivity for two-phase commit transactions.When using type 2 connectivity for distributed transactions, RRS is used as the transaction manager. In the case of type 4 connectivity for distributed transactions, there is no need for RRS, as this is handled by the type 4 (XA) driver.

Type 4 connectivity with distributed transaction support (two-phase commit) is bundled withDB2 for z/OS; IBM z/OS Application Connectivity to DB2 for z/OS and OS/390.

Note: More details on JDBC and SQLJ Universal Driver Type 4 XA connectivity distributed transaction support are also documented in the *DB2 Universal Database for z/OS Version 8 Application Programming Guide and Reference for JAVA*, SC18-7414.

Figure 6-7 on page 155 illustrates the Java application flow with DB2 Universal Driver type 2 and type 4 connectivity.

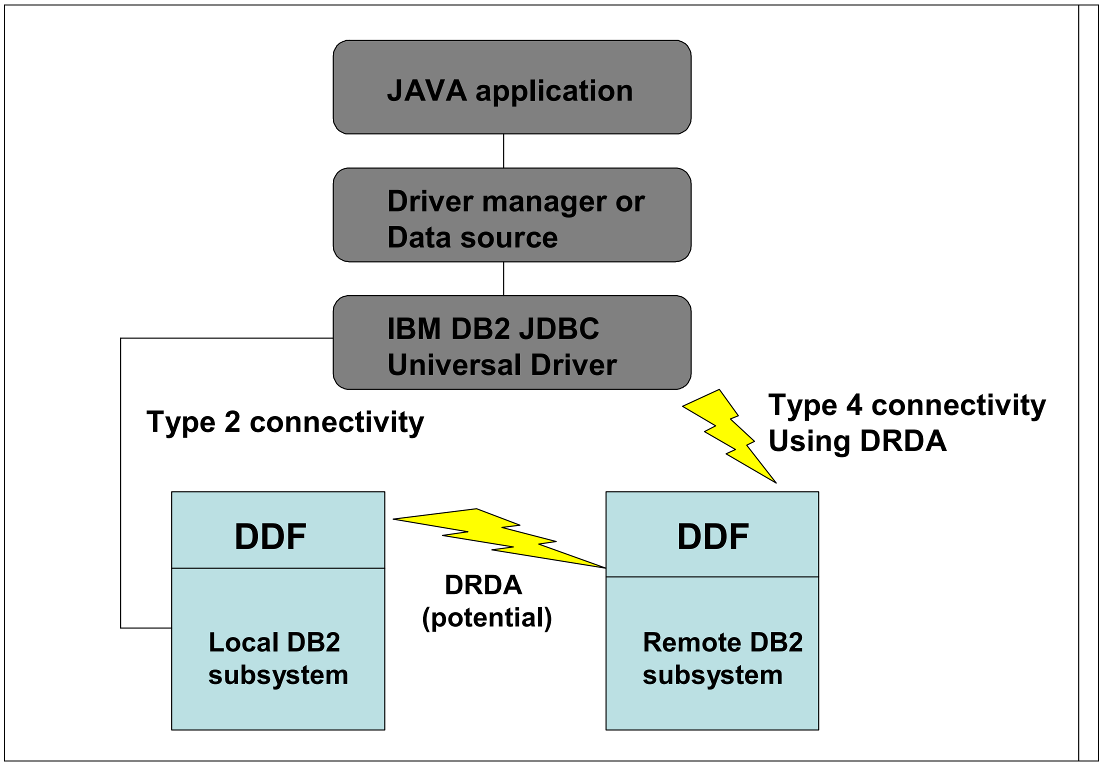


Figure 6-7 DB2 and JDBC

How a connection to a data source is made depends on the JDBC specification level being used. The DriverManager interface is available for all levels of JDBC. The DataSource interface is available with JDBC 2.0 and later.

Type 4 connectivity with the Universal Driver uses DRDA to connect to a database server (remote or local). This means that when using type 4 connectivity, DDF will be used even when a Java program runs on the same z/OS system or logical partition (LPAR) as the target DB2 subsystem. To avoid using DDF and the associated network overhead, we recommend using type 2 connectivity with the DB2 Universal Driver when a Java application is accessing a local DB2 subsystem.

The *IBM DB2 Universal Driver for SQLJ and JDBC* provides these new features:

- Cursor and stored procedure result set instances: A DB2 for z/OS server now allows multiple instances of a cursor, or multiple stored procedure result sets, to be open concurrently under the same thread.
- SQL cancel: A JDBC or CLI application can cancel long-running requests on a DB2 for z/OS server.
- Cursor extensions: DB2 for z/OS allows a requester to identify:
  - Whether the server should release read locks when a query is closed.
  - Whether the server should close a query implicitly when no more rows exist for a non-scrollable cursor, regardless of whether the cursor has the HOLD attribute.
- Better utilization of network capacity: DB2 for z/OS provides more flexibility for requesters such as DB2 Connect™ to specify larger query block sizes. This helps requesters optimize their use of network resources.

- Distributed transactions: DB2 for z/OS, Version 8 adds DRDA XA protocol support, which is needed to support Java Transaction API (JTA)/Java Transaction Service (JTS) distributed transactions. This support is available only for TCP/IP connections.
- Server location aliases: DB2 for z/OS supports location aliases that reflect the location names used by applications to route requests to all or a subset of members in a data sharing group.
- Subsets: DB2 for z/OS allows you to define subsets of data sharing group members in TCP/IP networks. A non-DB2 for z/OS requester can connect to a subset of data sharing group members by appending a port number to a location alias.
- Time-out for allocate conversation requests: If a VTAM® request to allocate a conversation for a remote SQL statement does not complete in three minutes, DDF forces VTAM to abnormally terminate the remote request.

Java packages for JDBC support

Before you can invoke JDBC methods in JAVA applications, you need to be able to access all or parts of various Java packages that contain those methods. You can do that either by importing the packages or specific classes, or by using the fully-qualified class names. You might need the following packages or classes for your JDBC program:

- **java.sql**

  Contains the core JDBC API.
- **javax.naming**

  Contains classes and interfaces for Java Naming and Directory Interface™ (JNDI), which is often used for implementing a DataSource.
- **javax.sql**

  Contains JDBC 3.0 standard extensions.
- **com.ibm.db2.jcc**

  Contains the DB2-specific implementation of JDBC for the DB2 Universal JDBC driver and some functions of the JDBC/SQLJ Driver for z/OS.
- **COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver**

  Contains some functions of the DB2-specific implementation of JDBC/SQLJ Driver for z/OS.

6.7.2 How JDBC applications connect to a data source

Before you can execute SQL statements in any SQL program, you must connect to a database server. In JDBC, a database server is known as a data source.

Figure 6-8 on page 157 shows how a Java application connects to a data source for a type 2 driver or DB2 Universal JDBC Driver type 2 connectivity.

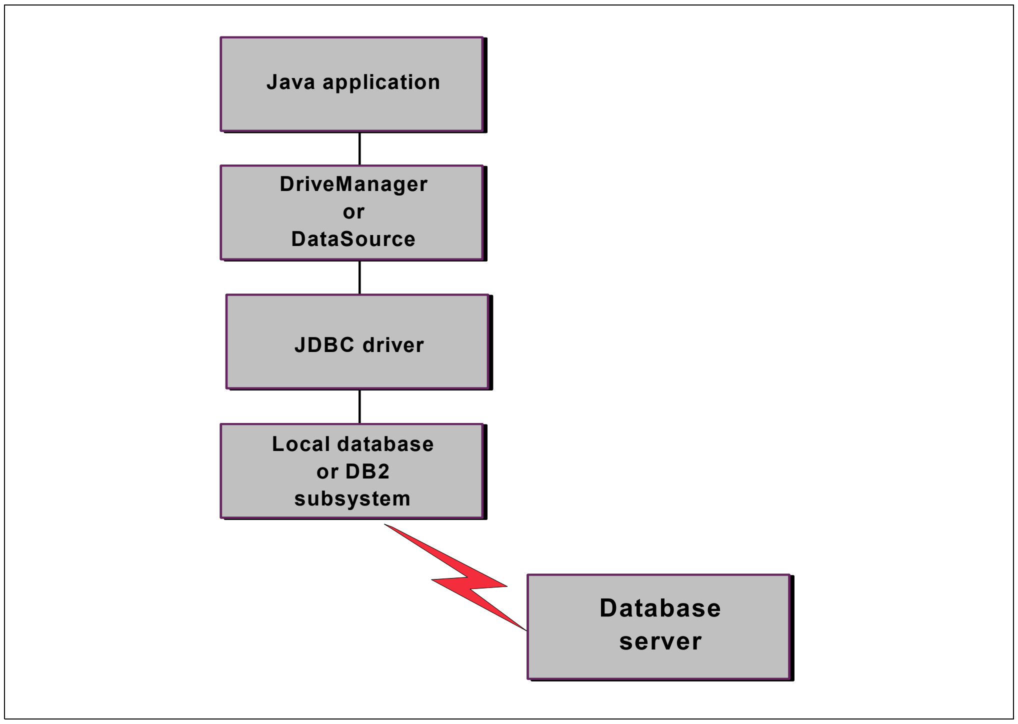


Figure 6-8 Java using Universal JDBC driver type 2

Figure 6-9 shows how a Java application connects to a data source for DB2 Universal JDBC Driver type 4 connectivity.

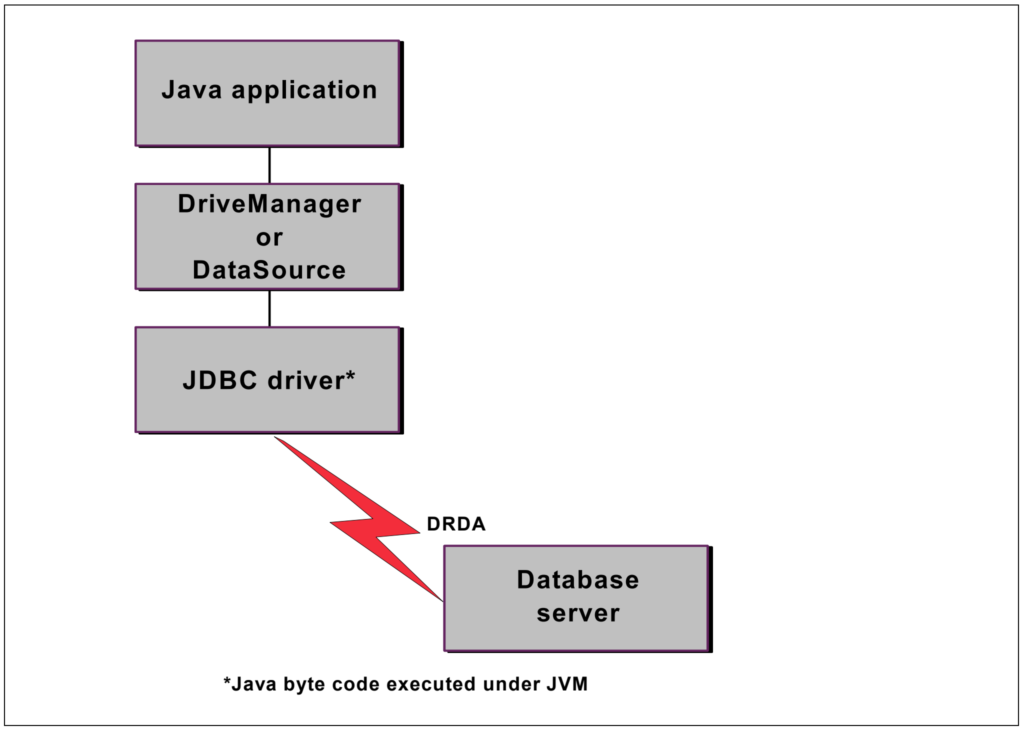

Java application
DriveManager or DataSource
JDBC driver*
DRDA
Database server
*Java byte code executed under JVM

Figure 6-9 Java using Universal JDBC driver type 4

The way that you connect to a data source depends on the version of JDBC that you use. Connecting using the DriverManager interface is available for all levels of JDBC. Connecting using the DataSource interface is available with JDBC 2.0 and above.

A JDBC application can establish a connection to a data source using the JDBC DriverManager interface, which is part of the java.sql package.

The Java application first loads the JDBC driver by invoking the Class.forName method. After the application loads the driver, it connects to a database server by invoking the DriverManager.getConnection method.

For the IBM DB2 Universal Driver for SQLJ and JDBC, you load the driver by invoking the Class.forName method with the following argument:

```
com.ibm.db2.jcc.DB2Driver
```

For compatibility with previous JDBC drivers, you can use the following argument instead:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

The code in Example 6-9 shows how to load the DB2 Universal JDBC Driver.

Example 6-9 Loading DB2 universal JDBC driver

```
Try {
  // Load the DB2 Universal JDBC Driver with DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
     e.printStackTrace();
}
```

Note: The catch block is used to print an error if the driver is not found

After you load the driver, you connect to the data source by invoking the DriverManager.getConnection method. You can use one of the forms listed in Example 6-10.

Example 6-10 Connect to a database using JDBC

```
getConnection:
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The url argument represents a data source, and indicates what type of JDBC connectivity you are using.

For DB2 Universal JDBC Driver type 4 connectivity, specify a URL with the syntax shown in Figure 6-10 on page 159.

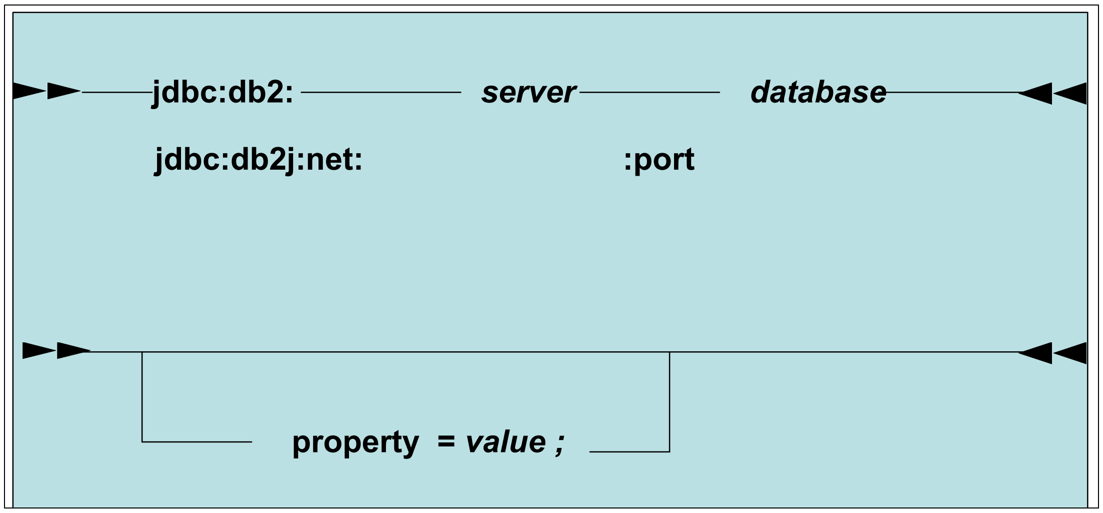


Figure 6-10 Syntax to use URL for Universal JDBC driver type 4

For DB2 Universal JDBC Driver type 2 connectivity, specify a URL of one of the following forms:

Syntax for a URL for Universal Driver type 2 connectivity, see Figure 6-11

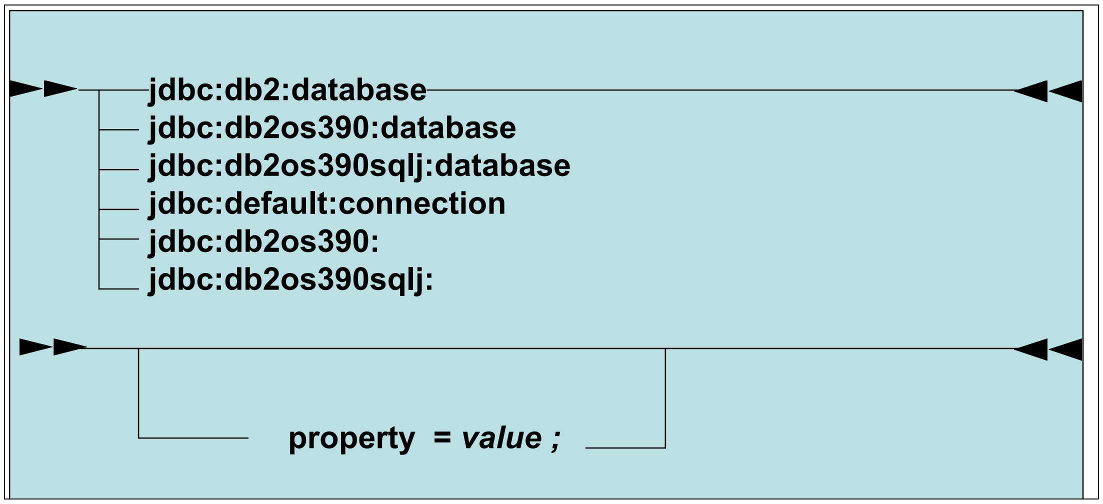


Figure 6-11 Syntax to use URL for Universal JDBC driver type 2

The meanings of the initial portion of the URL are:

- **jdbc:db2: or jdbc:db2os390: or jdbc:db2os390sqlj:**

  Indicates that the connection is to a DB2 for z/OS or DB2 UDB for Linux, UNIX, and Windows server. jdbc:db2os390: and jdbc:db2os390sqlj: are for compatibility of programs that were written for the JDBC/SQLJ Driver for OS/390.

- **jdbc:default:connection**

  Indicates that the URL is intended for environments that support an already-existing connection, such as CICS, IMS, and stored procedures.

- **jdbc:db2j:net:**

  Indicates that the connection is to a remote IBM Cloudscape server.

- **server**

  The domain name or IP address of the database server.
- **port**

  The TCP/IP server port number that is assigned to the database server. This is an integer between 0 and 65535. The default is 446.
- **database**

  A name for the database server. This name depends on whether Universal Driver type 4 connectivity or Universal Driver type 2 connectivity is used.

6.7.3 Specifying a user ID and password for a connection

There are several ways to specify a user ID and password for a connection:

- Use the form of the getConnection method that specifies url with property=value; clauses, and include the user and password properties in the URL.
- Use the form of the getConnection method that specifies user and password.
- Use the form of the getConnection method that specifies info, after setting the user and password properties in a java.util.Properties object.

Example 6-11 shows how to set up the user ID and password using the first method.

Example 6-11 Setting up ID and password

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose:" +
  "user=db2adm;password=db2adm;";
                                        // Set URL for data source
Connection con = DriverManager.getConnection(url);
                                        // Create connection
```

6.7.4 Which is the better JDBC driver type to Web Services?

The DB2 Universal JDBC Driver supports two types of connectivity: type 2 connectivity and type 4 connectivity. For the DriverManager interface, you specify the type of connectivity through the URL in the DriverManager.getConnection method. For the DataSource interface, you specify the type of connectivity through the driverType property.

The Table 6-4 summarizes the differences between type 2 connectivity and type 4 connectivity.

Table 6-4 Differences between the types 2 and 4

Function	Universal Driver type 2	Universal Driver type 4
Performance	Better for accessing a local DB2 server	Better for accessing a remote DB2 server
Installation	Requires installation of native libraries in addition to Java Classes	Requires installation of Java Classes only
Stored Procedures	Can be used to call or execute stored procedures	Can be used only to call stored procedures
Distributed transaction processing (XA)	Supported	Supported

Function	Universal Driver type 2	Universal Driver type 4
J2EE 1.4 compliance	Compliant	Compliant
CICS environment	Supported	Not supported
IMS environment	Supported	Not supported

Use Universal Driver type 2 connectivity under these circumstances

- Your JDBC or SQLJ application runs locally most of the time, local applications have better performance with type 2 connectivity.
- You are running a Java stored procedure. A stored procedure environment consists of two parts: a client program, from which you call a stored procedure, and a server program, which is the stored procedure. You can call a stored procedure in a JDBC or SQLJ program that uses type 2 or type 4 connectivity, but you must run a Java stored procedure using type 2 connectivity.
- Your application runs in the CICS environment or IMS environment

Use Universal Driver type 4 connectivity under these circumstances

- Your JDBC or SQLJ application runs remotely most of the time, remote applications have better performance with type 4 connectivity.
- Web services calling remote DB2.
- You do not have DB2 installed locally.
- Universal Driver type 2 connectivity relies on code that is part of DB2, but Universal Driver type 4 connectivity does not. Therefore, for Universal Driver type 4 connectivity, you do not need to have DB2 installed where the driver runs.

Important: Web services uses Universal Driver type 4 and we recommend it.

6.7.5 Other JDBC considerations

With the DB2 Universal JDBC Driver, you can change the isolation level only at the beginning of a transaction, after committing or rolling back JDBC transactions.In JDBC, to commit or roll back transactions explicitly, use the commit or rollback methods.

Example 6-12 shows the commands.

Example 6-12 Using commit statement

```
Connection con;
...
con.commit();
```

If autocommit mode is on, DB2 for z/OS performs a commit operation after every SQL statement completes. To determine whether autocommit mode is on, invoke the statement shown in Example 6-13.

Example 6-13 Setting autocommit

```
Connection.setAutoCommit(true);
```

6.8 DB2 for z/OS consuming Web services

Web services are increasingly used to integrate information processing within and between enterprises. When building service-based applications, Web services often have to be integrated with relational data. To accomplish this, applications must access both Web services and database management systems.

IBM DB2 for z/OS Web service consumer user-defined functions (UDFs) are now available to help with this task. These new Web service consumer UDFs enable databases to directly invoke SOAP-based Web services using SQL. This eliminates the need to transfer data between Web services and the database. The result is increased productivity and better performance. The Web services consumer converts existing WSDL interfaces into DB2 table or scalar functions.Allow mainframe-based enterprise applications to invoke Web Services regardless of their location DB2 can act as a client for Web services, which allows you to be a consumer of Web services in your DB2 applications.

Figure 6-12 shows DB2 for z/OS consuming Web services.

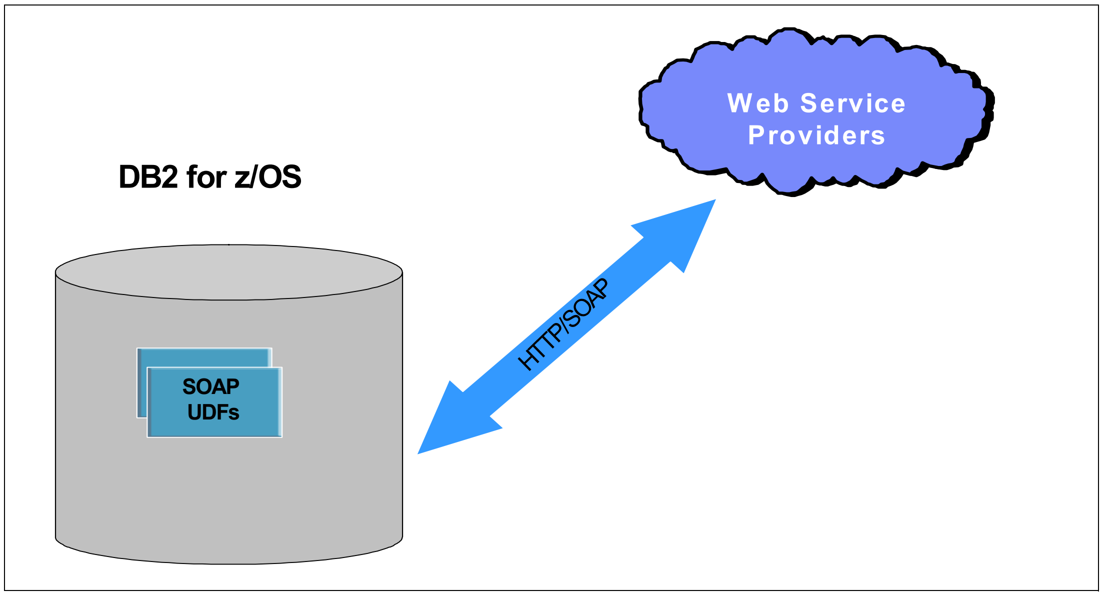


Figure 6-12 DB2 for z/OS consuming Web services

6.8.1 Using DB2 for z/OS UDFs

DB2 provides user-defined functions with which you can work with SOAP and consume Web services in SQL statements.

Figure 6-13 on page 163 shows UDF SOAP accessing Web services.

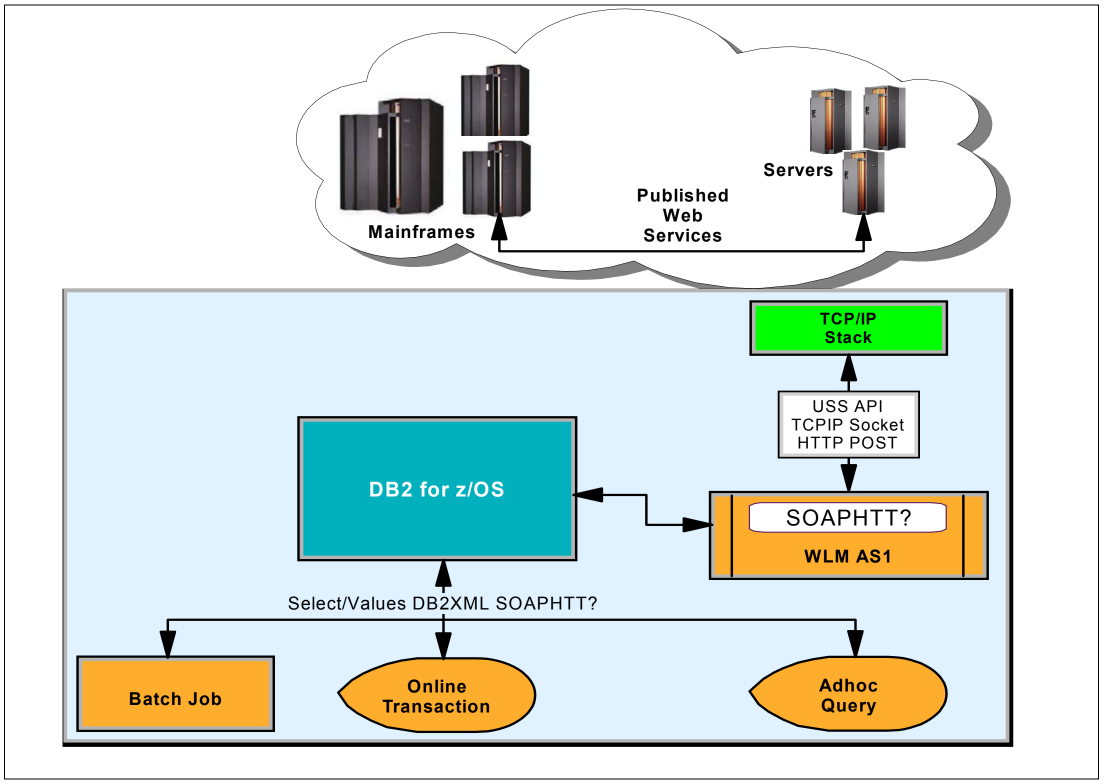


Figure 6-13 UDF SOAP accessing Web services

The user-defined functions are two varieties of SOAPHTTPV for VARCHAR data and two varieties of SOAPHTTPC for CLOB data. The user-defined functions perform the following actions:

1. Compose a SOAP request
2. Post the request to the service endpoint
3. Receive the SOAP response
4. Return the content of the SOAP body

When a consumer receives the result of a Web services request, the SOAP envelope is stripped and the XML document is returned. An application program can process the result data and perform a variety of operations, including inserting or updating a table with the result data.

DB2 provides these actions when using a SOAP UDF:

- Receive input parameters from SQL statement.
- Compose HTTP/SOAP request based on the input to the UDF.
- Invoke TCP/IP socket call via z/OS USS APIs and send HTTP/POST request.
- Receive reply from Web Service Provider.
- Validate HTTP headers.
- Strip SOAP envelope and return SOAP Body (include namespace referenced in SOAP envelope) to DB2 client application.

Note: Tooling support for DB2 for z/OS is in Rational Application Developer V7 to create a SQL UDF specifically for a Web services operation.

6.8.2 SOAPHTTPC and SOAPHTTPV

SOAPHTTPV and SOAPHTTPC are user-defined functions that allow DB2 to work with SOAP and to consume Web services in SQL statements. These functions are overloaded functions that are used for VARCHAR or CLOB data of different sizes, depending on the SOAP body. Web services can be invoked in one of four ways, depending on the size of the input data and the result data. SOAPHTTPV returns VARCHAR(32672) data and SOAPHTTPC returns CLOB(1M) data. Both functions accept either VARCHAR(32672) or CLOB(1M) as the input body.

For the syntax command, see Figure 6-14.

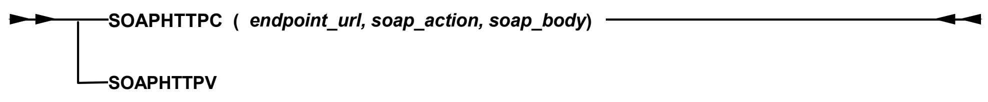


Figure 6-14 Syntax command to use SOAP UDFs

The SOAPHTTPC function returns a CLOB representation of XML data that results #from a SOAP request to the Web service specified by the first argument. The #SOAPHTTPV function returns a VARCHAR representation of XML data that results #from a SOAP request to the Web service specified by the first argument.

Three input parameters

ENDPOINT_URL: The URL of the Web service endpoint for which DB2 is acting as a client. Defined as VARCHAR(256)

SOAP_ACTION: A SOAP action URI reference. This parameter is optional depending on the Web service that is specified in endpoint_url. If it is required, the required value is defined in the WSDL of the specified Web service. Defined as VARCHAR(256)

SOAP_BODY: The name of an operation with requested namespace URI, an encoding style, and input arguments. Can include some well-formed XML content for the SOAP body. The specific operations and arguments for a Web service are defined in the WSDL of the specified Web service

The input for sapidity must be either VARCHAR(3072) or CLOB(1M) data.

One output parameter

Defined as VARCHAR(32672) or CLOB(1M)

If the arguments can be null, the result can be null; if all of the arguments # are null, the result is the null value.

The SQL statement in Example 6-14 on page 164 retrieves information (as VARCHAR data) about a Web service.

Example 6-14 Selecting data using SOAP UDF

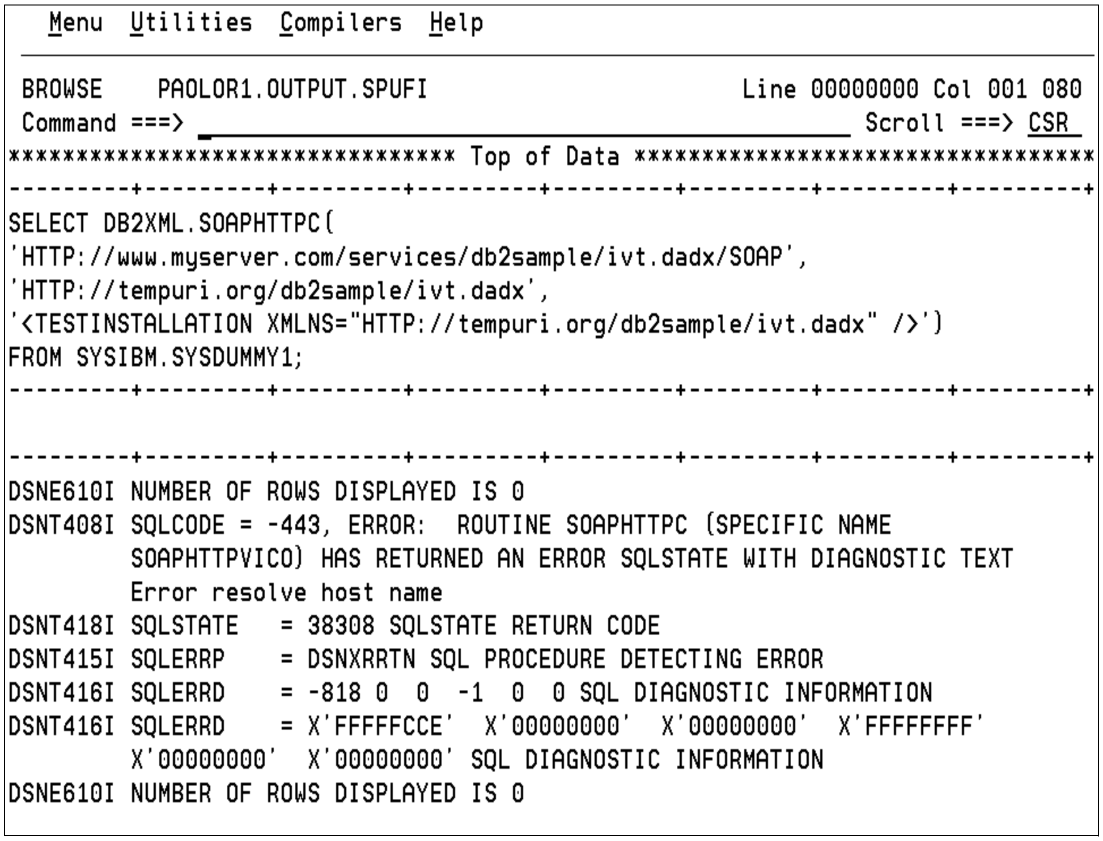
```
SELECT DB2XML.SOAPHTTPV(
'http://www.myserver.com/services/db2sample/ivt.dadx/SOAP',
'http://tempuri.org/db2sample/ivt.dadx',
'<testInstallation xmlns="http://tempuri.org/db2sample/ivt.dadx" />')
FROM SYSIBM.SYSDUMMY1;
```

The SQL statement in Example 6-15 inserts the results (as CLOB data) from a request to a Web service into a table.

Example 6-15 Inserting data using SOAP UDF

```
INSERT INTO EMPLOYEE(XMLCOL)
VALUES (DB2XML.SOAPHTTPC(
'http://www.myserver.com/services/db2sample/list.dadx/SOAP',
'http://tempuri.org/db2sample/list.dadx',
'<listDepartments xmlns="http://tempuri.org/db2sample/list.dadx">
<deptNo>A00</deptNo>
</listDepartments>'));
```

Figure 6-15 shows an SQLSTATE error using SOAP UDFs.



```
 BROWSE    PAOLOR1.OUTPUT.SPUFI                        Line 00000000 Col 001 080
 Command ===> _                                                Scroll ===> CSR
********************************* Top of Data **********************************
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT DB2XML.SOAPHTTPC(
'HTTP://www.myserver.com/services/db2sample/ivt.dadx/SOAP',
'HTTP://tempuri.org/db2sample/ivt.dadx',
'<TESTINSTALLATION XMLNS="HTTP://tempuri.org/db2sample/ivt.dadx" />')
FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+---------+---------+


---------+---------+---------+---------+---------+---------+---------+---------+
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNT408I SQLCODE = -443, ERROR:  ROUTINE SOAPHTTPC (SPECIFIC NAME
         SOAPHTTPVICO) HAS RETURNED AN ERROR SQLSTATE WITH DIAGNOSTIC TEXT
         Error resolve host name
DSNT418I SQLSTATE   = 38308 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXRRTN SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -818 0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFCCE'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
```

Figure 6-15 SPUFI screen

Table 6-5 on page 165 shows SQLSTATE values that DB2 for z/OS returns for error conditions related to using DB2 as a Web services consumer.

Table 6-5 SQL errors with DB2 consuming Web services

SQLSTATE	Description
38301	An unexpected NULL value was passed as input to the function.

SQLSTATE	Description
38302	The function was unable to allocate space.
38304	An unknown protocol was specified on the endpoint URL.
38305	An invalid URL was specified on the endpoint URL.
38306	An error occurred while attempting to create a TCP/IP socket.
38307	An error occurred while attempting to bind a TCP/IP socket.
38308	The function could not resolve the specified hostname.
38309	An error occurred while attempting to connect to the specified server.
38310	An error occurred while attempting to retrieve information from the protocol.
38311	An error occurred while attempting to set socket options.
38312	The function received unexpected data returned for the Web service.
38313	The Web service did not return data of the proper content type.
38314	An error occurred while initializing the XML parser.
38315	An error occurred while creating the XML parser.
38316	An error occurred while establishing a handler for the XML parser.
38317	The XML parser encountered an error while parsing the result data.
38318	The XML parser could not convert the result data to the database codepage.
38319	The function could not allocate memory when creating a TCP/IP socket.
38320	An error occurred while attempting to send the request to the specified server.
38321	The function was unable to send the entire request to the specified server.
38322	An error occurred while attempting to read the result data from the specified server.
38323	An error occurred while waiting for data to be returned from the specified server.
38324	The function encountered an internal error while attempting to format the input message.
38325	The function encountered an internal error while attempting to add namespace information to the input message.
38327	The XML parser could not strip the SOAP envelope from the result message.
38328	An error occurred while processing an SSL connection.
38305	An invalid URL was specified on the endpoint URL.
38306	An error occurred while attempting to create a TCP/IP socket.
38307	An error occurred while attempting to bind a TCP/IP socket.
38308	The function could not resolve the specified hostname.
38309	An error occurred while attempting to connect to the specified server.
38310	An error occurred while attempting to retrieve information from the protocol.

SQLSTATE	Description
38311	An error occurred while attempting to set socket options.
38312	The function received unexpected data returned for the Web service.
38313	The Web service did not return data of the proper content type.
38314	An error occurred while initializing the XML parser.
38315	An error occurred while creating the XML parser.
38316	An error occurred while establishing a handler for the XML parser.
38317	The XML parser encountered an error while parsing the result data.
38318	The XML parser could not convert the result data to the database codepage.
38319	The function could not allocate memory when creating a TCP/IP socket.
38320	An error occurred while attempting to send the request to the specified server.
38321	The function was unable to send the entire request to the specified server.
38322	An error occurred while attempting to read the result data from the specified server.
38323	An error occurred while waiting for data to be returned from the specified server.
38324	The function encountered an internal error while attempting to format the input message.
38325	The function encountered an internal error while attempting to add namespace information to the input message.
38327	The XML parser could not strip the SOAP envelope from the result message.
38328	An error occurred while processing an SSL connection.
38319	The function could not allocate memory when creating a TCP/IP socket.

7

DB2 for Linux, UNIX and Windows and SOA

In this chapter, we look at how DB2 UDB is being extended to provide support for Web services. DB2 can be a Web service provider as well as a Web service consumer. We show how Web services can be used in a database environment, and explain how Web services can be incorporated into SQL.

As a Web service provider, DB2 exposes DB2 objects as a Web service.

As a Web service consumer, DB2 has a set of user-defined functions (UDF) that provide a high-speed client Simple Object Access Protocol (SOAP) to access Web services over the Hypertext Transfer Protocol (HTTP) interface.

This chapter contains these topics:

- DB2 Web service components: Provider and consumer
- Web services provider
- Web services consumer

7.1 DB2 Web service components: Provider and consumer

Web services provide a flexible, programming-language independent technique for accessing DB2 databases. Web services provide a simple interface between the provider and consumer of application resources using a Web Service Description Language (WSDL). Web services can be exploited within a DB2 database in two manners:

- Web Services Provider
- Web Services Consumer

Figure 7-1 shows you how DB2 can be a Web service provider and also a Web service consumer, and where other Web services components come into the picture.

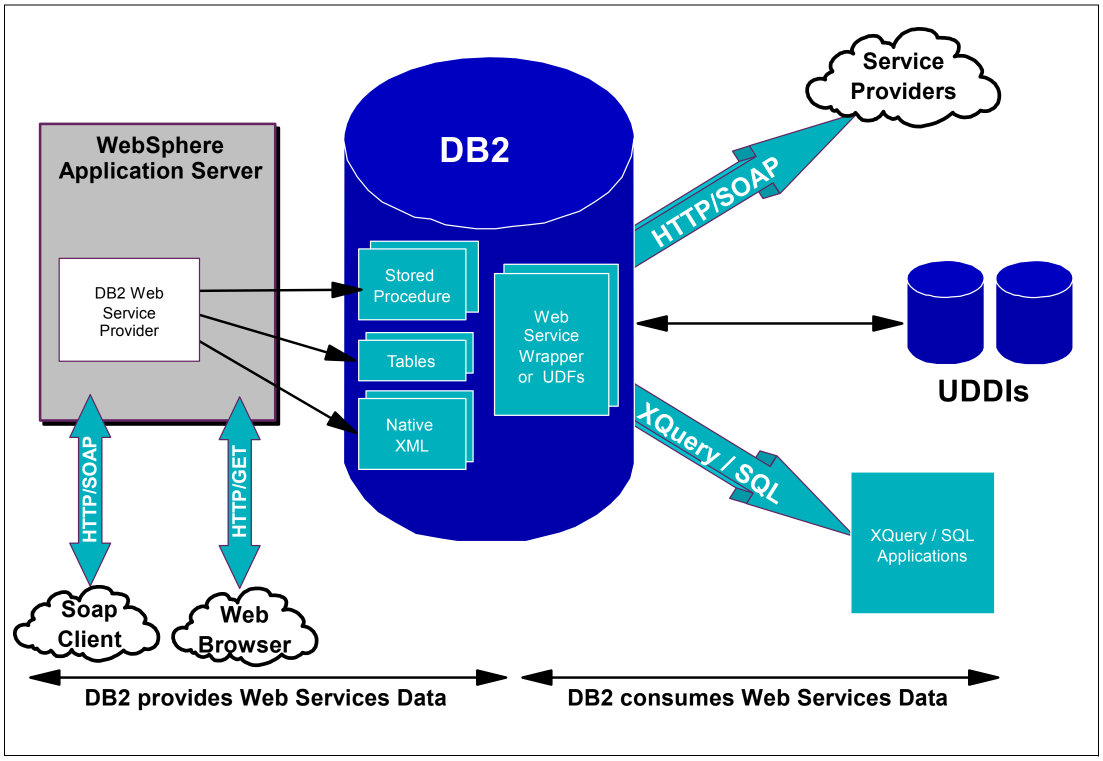


Figure 7-1 DB2 Web service components - Provider and consumer

7.2 Web services provider

DB2 provides the Web Services Object Runtime Framework (WORF) facility that can be used in conjunction with WebSphere Application Server to perform SQL queries, utilize DB2 V9.1 native XML data store to manipulate XML data, and invoke stored procedures. WORF provides an environment in which you can easily create simple XML-based Web services that can access a DB2 database. WORF uses an XML data format called Document Access Definition Extension (DADX) to simplify access to DB2 data using Web services standards.The idea is to define Web Services through simple XML files so that you can list database operations to be exposed as Web services.

Note: XML Extender is still supported in DB2 Version 9.1, but the native support presents several advantages. You can migrate your database application from XML Extender to use the native XML storage in DB2 Version 9.1. See 11.5, “Migrating from XML Extender” on page 366 for detail on how to migrate from DB2 XML Extender to native XML.

A document access definition extension (DADX) file specifies how to create a Web service. You can create a Web service by using a set of operations that are defined by SQL statements, stored procedure calls, or DAD files. Web services store XML documents or retrieve XML documents. Web services that are specified in a DADX file are called DADX Web services, or DB2 Web services.

WORF provides the run-time support for invoking DADX documents as Web services. These Web services use the Apache Axis engine (Version 1.1) or the WebSphere Web services engine. Both of these SOAP engines are supported by WebSphere Application Server. Apache Jakarta Tomcat does not support IBM WebSphere SOAP engine. Figure 7-2 shows the WORF architecture.

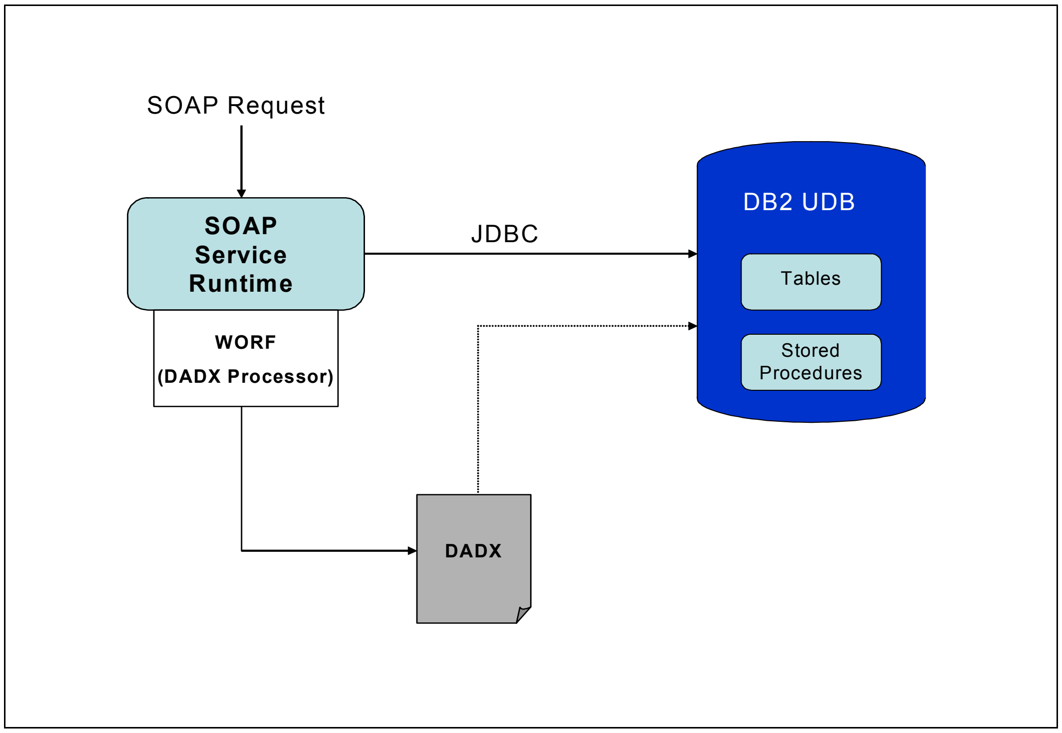


Figure 7-2 WORF architecture

WORF provides the following features:

- Resource-based deployment
  - A key feature of WORF is that it supports resource-based deployment of Web services.
  - WORF can generate the appropriate Web services from resource files. When you request the resource file, WORF loads the file and makes it available as a Web service.

- Web services automatic reloading
  - Automatic reloading makes developing DADX Web services as simple as the developing of Java Server Pages. If the WebSphere Application Server SOAP engine is used, the WORF application needs to be restarted if a DADX file got changed.
  - If the resource file is updated, WORF automatically detects the change and loads the new version. Automatic reload of the resource file makes Web service development more productive.
- Accessing the Web service with HTTP GET, POST bindings, in addition to SOAP bindings
  - You can access the Web service by using the HTTP GET, and SOAP bindings.
- Automatic WSDL generation
  - The WSDL document is used to describe the interface of business services. The DADX document contains the information required to implement the Web service, and it also contains the information required to generate the WSDL document.
- Automatic Web services documentation
  - You can include documentation in the DADX file for the Web service as a whole and for each operation in the Web service.
- Find Web services that exist from Web services provider
  - There is a large number of Web Services that you can use on your network. You must find these Web services and get information about them before you can use them. Web Services Inspection Language (WSIL) makes this search process easier.
  - By using WORF, you can inspect all of the Web services available within an DADX application. The inspection generator produces an XML document that is a list of available Web services of your DADX group.

7.2.1 Web service provider operations and DADX

In order to understand Web service provider operations and DADX, we need to first define a number of common terms.

The *document access definition extension (DADX)* file specifies a Web service. The DADX file defines the Web services by specifying a set of operations which you can invoke. The operation definition contains a list of parameters and actions to be performed. A Web service developer creates the DADX by using SQL statements, a list of parameters, and optionally, document access definition (DAD) file references that define a set of operations. A DADX file makes one Web service - meaning there is one WSDL document for each DADX file.

A *DADX group* is a grouping of one or more DADX files. Each DADX group has a `group.properties` file that defines the properties of the group. The DADX group contains connection (JDBC and JNDI) and other information that is shared between DADX files within the group. The implication is that each DADX group can only access one database.

A *WORF instance* is a J2EE Web application consists of DADX files, DADX group and WORF libraries.

7.2.2 Syntax of a DADX

The Document Access Definition Extension (DADX) file is an XML document. Figure 7-4 on page 176 shows the syntax of the DADX file.

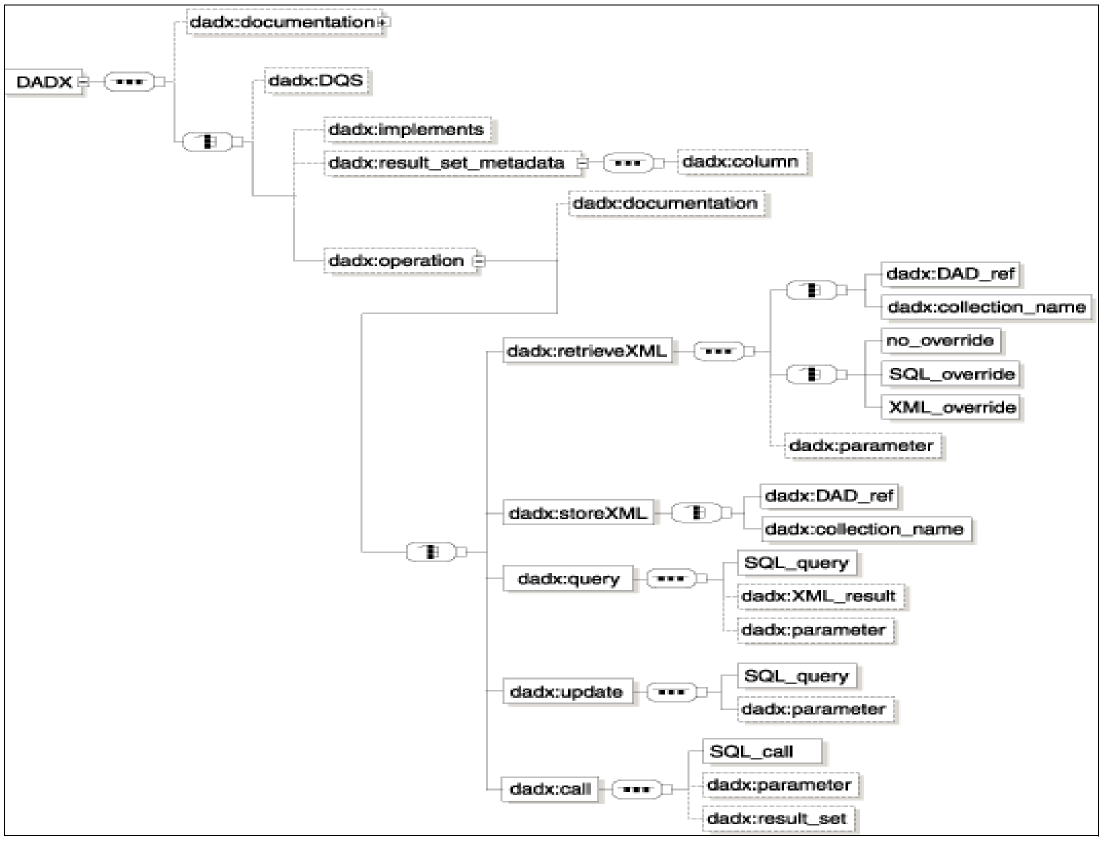


Figure 7-3 Syntax diagram of a DADX file

Note: In Figure 7-3, the numbers next to the nodes and elements in DADX syntax definitions 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.

Example 7-1 shows the explanation for the <result_set_metadata> element and its attributes which we use later to expose a stored procedure as a Web service.

Example 7-1 <result_set_metadata> element

```
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.
```

Example 7-2 shows the detail about the `<operation>` element, its attributes and children. Again we need this information when we expose a stored procedure as a Web service in the next example.

Example 7-2 <operation> element

```
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:

      ...

      1.4.6 <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

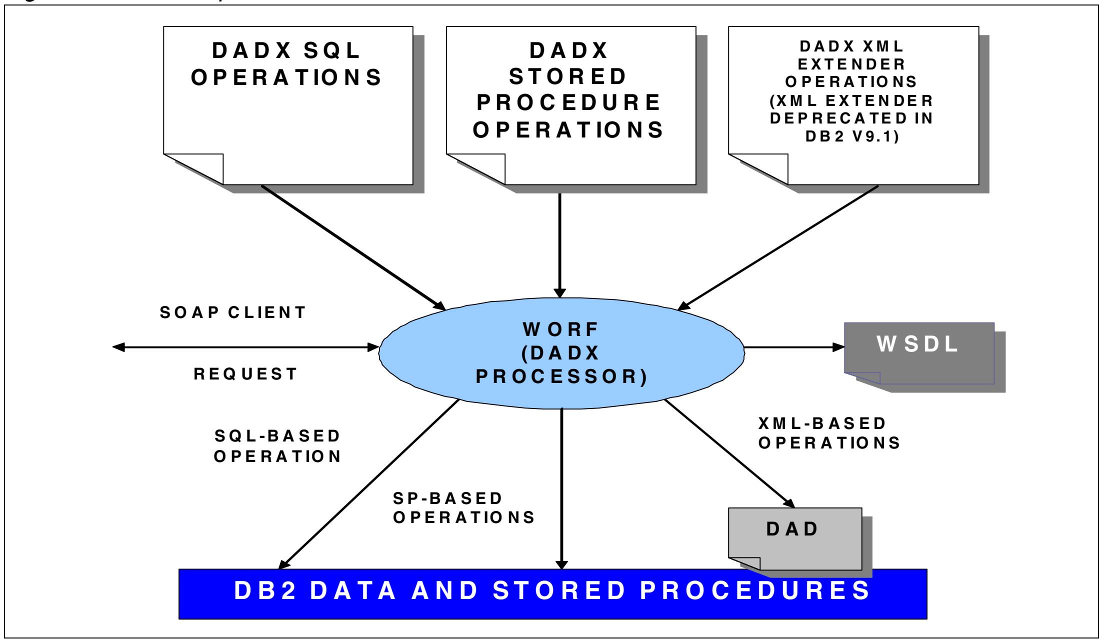
A result set metadata definition in the DADX file. The identifier must refer to the name of an element.

For full listing of DADX elements see Appendix C.12, “Syntax of the DADX file” on page 653. Also refer to Appendix C.11, “XML schema for the DADX file” on page 645 for the complete dadx.xsd file that describes the DADX schema.

7.2.3 DADX operations

DADX files support three kinds of Web service operations: non-dynamic SQL operations, dynamic SQL operations, and XML collection operations.

Figure 7-4 DADX operations


DADX SQL OPERATIONS
DADX STORED PROCEDURE OPERATIONS
DADX XML EXTENDER OPERATIONS (XML EXTENDER DEPRECATED IN DB2 V9.1)
SOAP CLIENT
REQUEST
WORF (DADX PROCESSOR)
WSDL
SQL-BASED OPERATION
SP-BASED OPERATIONS
XML-BASED OPERATIONS
DAD
DB2 DATA AND STORED PROCEDURES

See Example 7-3 for a summary of all the elements.

Example 7-3 Web service provider operations used with DADX file

```
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.

XML collection operations (from DB2 XML Extender)
<retrieveXML>
Generates XML documents

<storeXML>
Stores XML documents

Non-dynamic queries that use the Web service provider

A non-dynamic DADX file defines a set of predetermined SQL operations and contains information that is used to create the Web service. In Example 7-4, we show a very simple DADX file that contains only one SQL query. This particular DADX file is used for non-dynamic queries.

Example 7-4 ListDepartment.dadx

```
<?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>
```

Important: 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.

Dynamic queries that use the Web services provider

You can also define a set of dynamic Web service operations with a DADX file that contains only the dynamic query service tag (<DQS/>). A DADX file with a dynamic query services tag (</DQS>) contains only that tag which acts as a switch to enable dynamic queries for that group only. 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. With dynamic query services you can dynamically build and submit queries at run time that select, insert, update and delete application data, and call stored procedures rather than run queries that are predefined at deployment time. By using the dynamic query services of the Web services provider, Web applications can be more flexible.

To prepare your Web services environment to run dynamic queries on a DB2 Web services provider, you 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. Then you save the file in the directory of the group for which you will run dynamic queries. Using the WSDL, develop a client for the application. The client must contain the group name, name of the DADX file, and the Web service operation. Example 7-5 on page 178 shows a sample DADX file using the

<DQS/> tag. You want to place the DADX file in the directory of the group for which you will execute dynamic queries.

Example 7-5 DADX that contains the dynamic query service tag

```
<?xml version="1.0"?>
<DADX xmlns="http://schemas.ibm.com/db2/dxx/dadx">
<documentation>Using dynamic query service/documentation>
<DQS/>
</DADX>
```

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.

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. db2WebRowSet.xsd is packaged in dxxworf.zip as part of the DB2 installation. In 11.3, “Preparing the installation of the Web services provider” on page 363 we further discuss how to setup the Web service provider on supported Web servers.

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.

For a full listing of all dynamic query operations, see Appendix C.13, “Dynamic query service operations in the Web services provider” on page 659.

Figure 7-5 shows the relationship between DB2 and various DADX components.

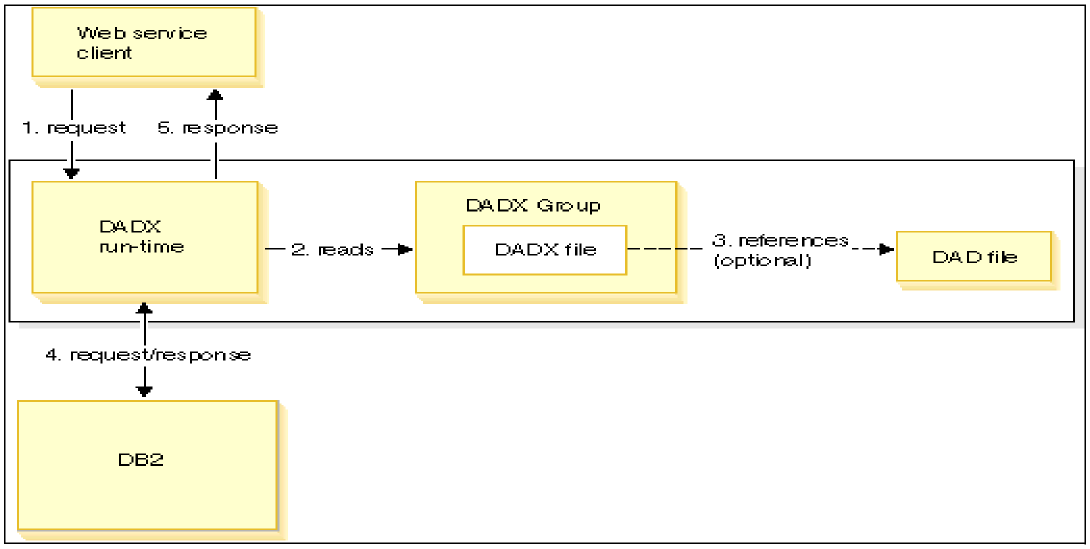


Figure 7-5 Relationship between DB2, DADX runtime, DADX group and DADX file

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.

Figure 7-6 shows a typical development scenario for a Web service provider. A Web service developer develops the DADX file, then deploys it to the WebSphere Application Server or any supported Web server. Then a Web application is created to host the DB2 Web Service. You may then choose to publish your Web Service using UDDI so that other programmers can search for your Web service and use it.

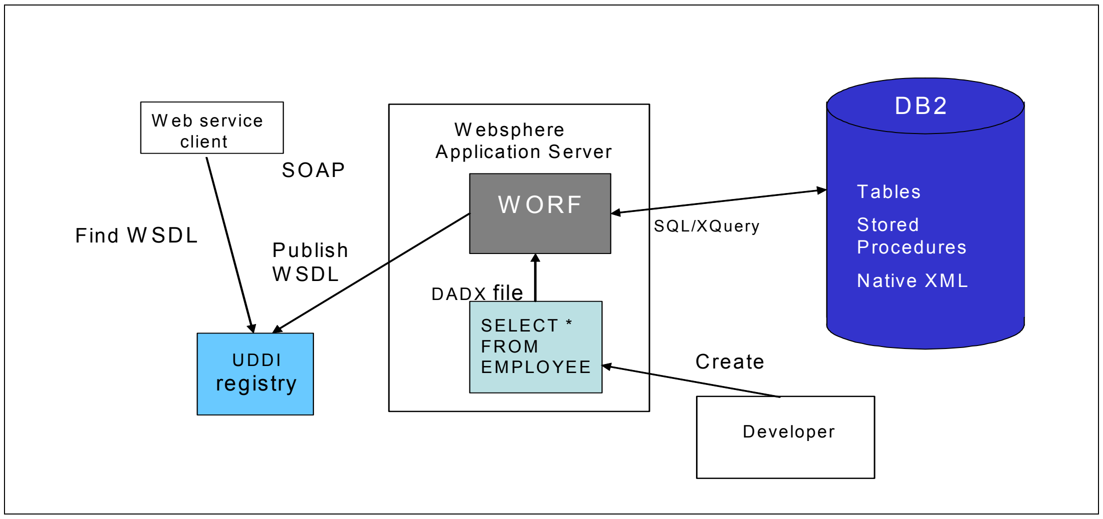


Figure 7-6 Development scenario for a Web service provider

7.2.4 Exposing a stored procedure as a Web service

To expose DB2 objects and operations as Web services, you create a Document Access Definition Extension (DADX) file. As explained earlier in 7.2.1, “Web service provider operations and DADX” on page 172, a DADX file is an XML file that is used to create a Web service that accesses a relational database.

You can use the Web services DADX File wizard in the IBM Rational Application Developer to create a DADX file. The Web Service DADX Group Configuration wizard assists you in creating a DADX group. Once you have created a DADX group, you can proceed to generate your DADX file and deploy your new Web service.

In Example 7-6, you see a DADX file that shows a CALL operation to invoke a stored procedure `TWO_RESULT_SETS`. A <result_set_metadata> element is generated in the DADX for each result set returned by the stored procedure.

Example 7-6 DADX file: CALL to the TWO_RESULT_SETS stored procedure

```
<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)</SQL_call>
      <parameter name="salary" type="xsd:double" kind="in" />
      <result_set name="employees1" metadata="employeeSalaryReport" />
      <result_set name="employees2" metadata="employeeSalaryReport" />
</call>
</operation>
</DADX>
```

The `TWO_RESULT_SETS` stored procedure is part of the sample programs shipped with DB2's `SAMPLE` database. The source code for this procedure can be found in SpServer.java.

On Windows: %DB2PATH%\sqllib\samples\java\jdbc (where %DB2PATH% is a variable that determines where DB2 is installed)

On UNIX: $HOME/sqllib/samples/java/jdbc (where $HOME is the home directory of the instance owner)

The CREATE PROCEDURE statement for the TWO_RESULT_SETS procedure can also be found in SpCreate.db2 in the same directory. The TWO_RESULT_SETS procedure takes a single input parameter of data type double which is the salary input. The procedure queries the STAFF table in the SAMPLE database, and returns two result sets to the caller. The first result set consists of employee data of all employees with salaries greater than the salary input parameter. The second result set contains employee data for employees with salaries less than the salary input parameter.

Example 7-7 Calling the TWO_RESULT_SETS with an input value of 20000.00

```
db2 call TWO_RESULT_SETS(20000.00)

  Result set 1
  --------------

  NAME      JOB   3
  --------- ----- ------------------------
  Lu        Mgr     +2.00100000000000E+004
  Hanes     Mgr     +2.06598000000000E+004
  Graham    Sales   +2.10000000000000E+004
```

```
Fraye     Mgr     +2.11500000000000E+004
Jones     Mgr     +2.12340000000000E+004
Molinare  Mgr     +2.29592000000000E+004

6 record(s) selected.


Result set 2
--------------

NAME      JOB   3
--------- ----- ------------------------
Quill     Mgr     +1.98180000000000E+004
Williams  Sales   +1.94565000000000E+004
Daniels   Mgr     +1.92602500000000E+004
Wilson    Sales   +1.86745000000000E+004
Lea       Mgr     +1.85555000000000E+004
Sanders   Mgr     +1.83575000000000E+004
Plotz     Mgr     +1.83528000000000E+004
Pernal    Sales   +1.81712500000000E+004
O'Brien   Sales   +1.80060000000000E+004
Koonitz   Sales   +1.80017500000000E+004
Edwards   Sales   +1.78440000000000E+004
Smith     Sales   +1.76545000000000E+004
Marenghi  Mgr     +1.75067500000000E+004
Gonzales  Sales   +1.68582000000000E+004
Quigley   Sales   +1.68083000000000E+004
Rothman   Sales   +1.65028300000000E+004
Davis     Sales   +1.54545000000000E+004
Wheeler   Clerk   +1.44600000000000E+004
Sneider   Clerk   +1.42527500000000E+004
James     Clerk   +1.35046000000000E+004
Lundquist Clerk   +1.33698000000000E+004
Gafney    Clerk   +1.30305000000000E+004
Naughton  Clerk   +1.29547500000000E+004
Ngan      Clerk   +1.25082000000000E+004
Kermisch  Clerk   +1.22585000000000E+004
Abrahams  Clerk   +1.20097500000000E+004
Scoutten  Clerk   +1.15086000000000E+004
Burke     Clerk   +1.09880000000000E+004
Yamaguchi Clerk   +1.05059000000000E+004

29 record(s) selected.

Return Status = 0
```

By exposing the `TWO_RESULT_SETS` stored procedure as a Web service, anyone that has access to the salary Web Service provided by the `TWO_RESULT_SETS` stored procedure can easily query the database and obtain a list of employee records that has salaries above and below the threshold by providing a specific input salary parameter.

WORF produces WSDL and XML schema for the operations included in the DADX file, and the client application does not need to be aware that it is invoking DB2 stored procedure or executing any SQL statements. This opens the possibility of allowing Web services to access stored procedures and UDF that perform complex business logics and data processing. Now you see how Web services can open up many DB2 functionalities to a wide range of users.

In 15.1.2, “Implementation of the Web Services using WORF” on page 404, we walk you through the steps to creating a Web service from a DADX file using the IBM SOAP runtime with Rational Application Developer (RAD), when we present our use case.

7.3 Web services consumer

DB2 can also consume data from existing Web services based applications using SQL language extensions. DB2 Web Service consumer consists of a set of UDFs that provide a high-speed client Simple Object Access Protocol (SOAP) over Hypertext Transfer Protocol (HTTP) interface to accessible Web services. These SOAP UDFs allows you to directly invoke Web Services using SQL statements.

7.3.1 Web services consumer user-defined functions

The following are the SOAP UDFs provided by DB2:

- db2xml.soaphttpv
- db2xml.soaphttpc
- db2xml.soaphttpcl

Example 7-8 Web service consumer UDFs provided by DB2

```
db2xml.soaphttpv (endpoint_url VARCHAR(256),
                  soap_action VARCHAR(256),
                  soap_body VARCHAR(3072))
       RETURNS VARCHAR(3072)

db2xml.soaphttpv (endpoint_url VARCHAR(256),
                  soap_action VARCHAR(256),
                  soap_body CLOB(1M))
       RETURNS VARCHAR(3072)

db2xml.soaphttpc (endpoint_url VARCHAR(256),
                  soapaction VARCHAR(256),
                  soap_body VARCHAR(3072))
       RETURNS CLOB(1M)

db2xml.soaphttpc (endpoint_url VARCHAR(256),
                  soapaction VARCHAR(256),
                  soap_body CLOB(1M))
       RETURNS CLOB(1M)

db2xml.soaphttpcl(endpoint_url VARCHAR(256),
                  soapaction VARCHAR(256),
                  soap_body varchar(3072))
       RETURNS CLOB(1M) as locator
```

Each UDF takes the endpoint URL, a soap action, a soap body of the SOAP request as input parameters, and return the body of the SOAP response. These UDFs are overloaded depending on whether the SOAP body of the request or response is a VARCHAR or a CLOB.

There are two versions of SOAPHTTPV and two versions of SOAPHTTPC, giving you five SOAP UDFs in total.

These UDFs perform the following sequence of actions:

1. Compose a SOAP request
2. Post the request to the service endpoint
3. Receive the SOAP response
4. Return the content of the SOAP body of the response

A Web services consumer consists of SOAP requests and responses. SOAP is an XML messaging 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 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`
- XML content of the SOAP body, which includes:
  - Name of an operation with requested namespace URI
  - Encoding style
  - Input arguments
- An *optional* SOAP action URI reference which may be an empty string

Example 7-9 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`.

Example 7-9 A DB2 constructed SOAP request envelope

```
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>
```

Example 7-10 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.

Example 7-10 Using DB2 to extract the contents of the SOAP response envelop

```
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-OF51ZDk;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">52<return>
</ns1:getTempResponse> </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP UDFs allows DB2 to consume Web services in SQL statements, When a consumer receives the result of a Web service request, the SOAP envelop is stripped and the XML document is returned.

Most networks consists of a firewall. The network traffic is restricted to certain machines and certain ports numbers. 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. 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.

For more information regarding the installation of the Web services consumer user-defined functions, refer to Chapter 11, “The Linux, UNIX, and Windows products for SOA” on page 359.

7.3.2 Generating Web services consumer functions from WSDL

The consumer SOAP UDFs can invoke operations that are defined by a user-specified Web services description language (WSDL) file. You must construct the SOAP body of the SOAP request according to the WSDL of a Web service. You want make Web Services a natural extension to DB2 SQL environment. In order to do this, we have to address a number of issues. First, we need to map the UDF signature to the Web Services signature it implements. Second we need to use the data to construct and send the SOAP messages to Web Services end points. After receiving the response, the reply must be decomposed into the set of result parameters that the user expects.

DB2's implementation uses two layers of functions: a set of SOAP UDFs that are specific for each WSDL operation, and a set of underlying functions that actually perform the Web service invocation. In simple words, we always create a wrapper UDF to a Web services consumer function, because invoking the consumer functions directly cluttered the application code. Further, we have to pass the Web service URL and operating name each time we make a Web service call when we use the db2xml.soaphttp() functions described above. By creating a wrapper UDF, we wrap the call to a Web service inside another UDF, so that each time we call the wrapper UDF, the wrapper UDF internally calls the db2xml.soaphttp() functions.

Let us look at an example. The WSDL for a Temperature Service is shown in Example 7-11.

DemoTemperatureService.wsdl is also available at:

http://www.xmethods.net/sd/2001/DemoTemperatureService.wsdl

Example 7-11 DemoTemperatureService.wsdl

```
<?xml version="1.0" ?>
<definitions name="TemperatureService"
   targetNamespace="http://www.xmethods.net/sd/TemperatureService.wsdl"
   xmlns:tns="http://www.xmethods.net/sd/TemperatureService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
<message name="getTempRequest">
   <part name="zipcode" type="xsd:string" />
</message>
<message name="getTempResponse">
   <part name="return" type="xsd:float" />
</message>

<portType name="TemperaturePortType">
   <operation name="getTemp">
      <input message="tns:getTempRequest" />
      <output message="tns:getTempResponse" />
   </operation>
</portType>

<binding name="TemperatureBinding" type="tns:TemperaturePortType">
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
   <operation name="getTemp">
      <soap:operation soapAction="" />
      <input>
         <soap:body use="encoded" namespace="urn:xmethods-Temperature-Demo"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
         <soap:body use="encoded" namespace="urn:xmethods-Temperature-Demo"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
   </operation>
</binding>
```

```
<service name="TemperatureService">
   <documentation>Returns current temperature in a given U.S.
zipcode</documentation>
   <port name="TemperaturePort" binding="tns:TemperatureBinding">
      <soap:address
location="http://services.xmethods.net:80/soap/servlet/rpcrouter" />
   </port>
</service>
</definitions>
```

Note: The bold content is important. These are the parameters we need to pass to the Web service consumer functions. More detail below. Also note that the current temperature service simply return a hard coded value of 52 degrees. This is not very useful in real life situations, but for demonstration purposes this suffices.

By analyzing the WSDL, you can discover how to map the elements of the WSDL the parameters of Web service consumer functions. Remember that the Web service consumer functions takes a service endpoint, a SOAP action URI (optional) and a SOAP body to build a SOAP request and receive a SOAP response:

```
db2xml.soaphttp* (endpoint_url VARCHAR(256),
                  soap_action VARCHAR(256),
                  soap_body VARCHAR(3072) | CLOB(1M))
```

Refer to DemoTemperatureService.wsdl in Example 7-11 on page 185 as necessary when you go through steps 1 - 3:

1. To get the service endpoint, we look at the `<service>` section that contains the `<port>` information. Look at DemoTemperatureService.wsdl and we found the following:

   `http://services.xmethods.net:80/soap/servlet/rpcrouter`

2. To get the soap action, we look at the `<binding>` section that contains a list of all the operations offered by this Web service. We found the SOAP action, and it is an empty string "" (As explained earlier, soap action is optional and can either be an empty string or even be null. DB2 process empty string or null the same way in this context.)

   <soap:operation soapAction="" />

3. To get the soap body, we look at the `<portType>` definition and look at the input and output elements.

   ```
   <part name="zipcode" type="xsd:string" />

     ...
     <output>
        <soap:body use="encoded" namespace="urn:xmethods-Temperature-Demo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
     </output>
   ```

   Combining all the information we get our SOAP body:

   ```
   <m:getTemp xmlns:m="urn:xmethods-Temperature-Demo"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <zipcode xsi:type="xsd:string">ZIPCODE input</zipcode> </m:getTemp>
   ```

Note: *ZIPCODE input* is the actual zip code value the user supplies.

We can test the Web service consumer function in the DB2 Command Line Processor (CLP) or DB2 Command window. Assuming you have already connected to a database where SOAP UDFs are enabled, cut and paste the following into a text file and save it. In our case, we saved the command into a file getTemp.sql. Then enter the following to execute the command from the file:

db2 -tvf getTemp.sql

Example 7-12 Testing the temperature Web service from the DB2 CLP

```
VALUES DB2XML.SOAPHTTPV
('http://services.xmethods.net:80/soap/servlet/rpcrouter',
'',
'<m:getTemp xmlns:m="urn:xmethods-Temperature-Demo"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> <zipcode
xsi:type="xsd:string">95120</zipcode> </m:getTemp>');
```

You get the result shown in Example 7-13. It shows that we got a temperature of 52 degrees.

Example 7-13 Result from the temperature Web service

```
<ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature-Demo"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
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">
   <return xsi:type="xsd:float">52.0</return>
</ns1:getTempResponse>
```

Now we define our wrapper UDF `getTemperature` which takes a zipcode as input parameter of type VARCHAR(5), and returns the result as DECIMAL(5,1) in degrees. In Example 7-14, the user provides SQL input, and the SOAP UDF constructs the XML input according to the WSDL description of the provider.

Example 7-14 getTemperature UDF

```
CREATE FUNCTION getTemperature (zipcode VARCHAR(5))
RETURNS DECIMAL (5,1)
LANGUAGE SQL CONTAINS SQL
EXTERNAL ACTION NOT DETERMINISTIC
RETURN
DB2XML.EXTRACTDOUBLE(
   DB2XML.XMLVARCHAR(
      DB2XML.SOAPHTTPV(
         'http://services.xmethods.net:80/soap/servlet/rpcrouter',
    '',
    varchar('<m:getTemp xmlns:m="urn:xmethods-Temperature-Demo" ' ||

'SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">' ||
                 '<zipcode xsi:type="xsd:string"> </zipcode>' ||
                 '</m:getTemp>'
                )
      )
   ),'//return'
);
```

The same wrapper UDF getTemperature can also be written in a different manner. Example 7-15 shows a more reader friendly version of the `getTemperature` UDF. We have also added comments in Example 7-15 so you can see clearly where the endpoint URI and SOAP body from the WSDL file (DemoTemperatureService.wsdl) is used, where the SOAP UDF constructs the XML input and returns the output, and where the type conversion take place.

> **Note: DB2XML.EXTRACTDOUBLE** and **DB2XML.XMLVARCHAR** are XML Extender extracting functions. In Example 7-18 on page 189 we show a new version of the `getTemperature` UDF that uses DB2 V9 native XML functions.

Example 7-15 Reader friendly version of the getTemperature UDF

```
CREATE FUNCTION getTemperature (zipcode VARCHAR(5))
RETURNS DECIMAL (5,1)
LANGUAGE SQL CONTAINS SQL
EXTERNAL ACTION NOT DETERMINISTIC
RETURN
WITH

--1. Perform type conversions and prepare SQL input parameters for SOAP envelope

   soap_input (in)
          AS
          (VALUES varchar(
               '<m:getTemp xmlns:m="urn:xmethods-Temperature-Demo" ' ||
'SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">' ||
              '<zipcode xsi:type="xsd:string"> </zipcode>' ||
              '</m:getTemp>') ),

--2. Submit SOAP request with input parameter and receive SOAP response

        soap_output(out)
          AS
          (VALUES DB2XML.SOAPHTTPV(
               'http://services.xmethods.net:80/soap/servlet/rpcrouter',
               '',
               (SELECT in FROM soap_input)))

--3. Shred SOAP response and perform type conversions to get SQL output parameters

        SELECT DB2XML.EXTRACTDOUBLE(DB2XML.XMLVARCHAR(out), '//return')
        FROM soap_output;
```

Once you have created the wrapper UDF, you can easily invoke the temperature Web service simply by issuing the following command in DB2 CLP as shown in Example 7-16. Notice how much simpler the command is compared to the previous example shown in Example 7-12 on page 187.

Example 7-16 Invoking the wrapper UDF from the DB2 CLP

```
db2 values getTemperature('95120')

1
-------
```

```
  52.0

  1 record(s) selected.
```

DB2 allows users to define new functions that may be invoked from SQL, thus extending the SQL language. Example 7-17 is a very simple example that shows you how to incorporate our `getTemperature` UDF into a regular SQL statement. In this example, we store the temperature we received from the temperature Web service into the temperature table `TempTable` in our database. If you also keep track of the time stamp each time you invoke the temperature Web service, you can easily build a table that store historical temperature data for the area presented by zip code `95120`.

Example 7-17 Integrating your UDF within a SQL statement

```
db2 create table TempTable(t timestamp, temp DECIMAL(5,1))
DB20000I  The SQL command completed successfully.

db2 insert into TempTable values (current timestamp, getTemperature('95120'))
DB20000I  The SQL command completed successfully.

db2 select * from TempTable

T                          TEMP
-------------------------- -------
2006-04-07-11.17.59.302001    52.0

  1 record(s) selected.
```

As mentioned earlier, previous variations of the `getTemperature` UDF in Example 7-14 on page 187 and Example 7-15 on page 188 are still using XML Extender extracting functions to extract data and perform conversions. You could start moving away from these functions and begin using the native XML functions in DB2 V9.1 for Linux, UNIX and Windows. Example 7-18 shows the changes we made to the `getTemperature` function to make use of the new native XML functions in DB2 V9.1 for Linux, UNIX and Windows. The changes are highlighted in **BOLD**.

Example 7-18 New getTemperature UDF using native XML functions in DB2 V9.1

```
CREATE FUNCTION getTemperature (zipcode VARCHAR(5))
RETURNS DECIMAL (5,1)
LANGUAGE SQL CONTAINS SQL
EXTERNAL ACTION NOT DETERMINISTIC
RETURN
WITH

--1. Perform type conversions and prepare SQL input parameters for SOAP envelope

   soap_input (in)
          AS
          (VALUES varchar(
             '<m:getTemp xmlns:m="urn:xmethods-Temperature-Demo" ' ||
'SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">' ||
              '<zipcode xsi:type="xsd:string"> </zipcode>' ||
              '</m:getTemp>') ),

--2. Submit SOAP request with input parameter and receive SOAP response
```

```
        soap_output(out)
          AS
          (VALUES XMLPARSE
             (DOCUMENT DB2XML.SOAPHTTPC(
                'http://services.xmethods.net:80/soap/servlet/rpcrouter',
                '',
                (SELECT in FROM soap_input))))

--3. Shred SOAP response and perform type conversions to get SQL output parameters

        SELECT DECIMAL(CAST(XML2CLOB(
           XMLQUERY('$t//return/text()' PASSING out as "t")) AS CHAR(5)))
        FROM soap_output;
```

Example 7-18 also shows that we made use of XQuery:

`SELECT ... XMLQUERY('$t//return/text()' PASSING out as "t")`

Since you can still get your DB2 Web services up and running without XQuery, we will not discuss XQuery in detail here. However, you can read further information about DB2 V9.1 for Linux, UNIX and Windows new native XML store, XQuery, and our new Developer Workbench tool in Appendix C, “XML and DB2 for Linux, UNIX and Windows” on page 593.

The above examples show how you can use very simple, yet powerful standard DB2 SQL queries that utilizes Web services. By invoking Web services as UDFs, you can take advantage of the full power of SQL to perform queries across combinations of Web services and persistent data. In the `getTemperature` example we showed how a Web service that returns a single value can be integrated with DB2 SQL, but we can also handle multiple return values. In the case where we want to return multiple values, we will create a table function instead of a scalar function.

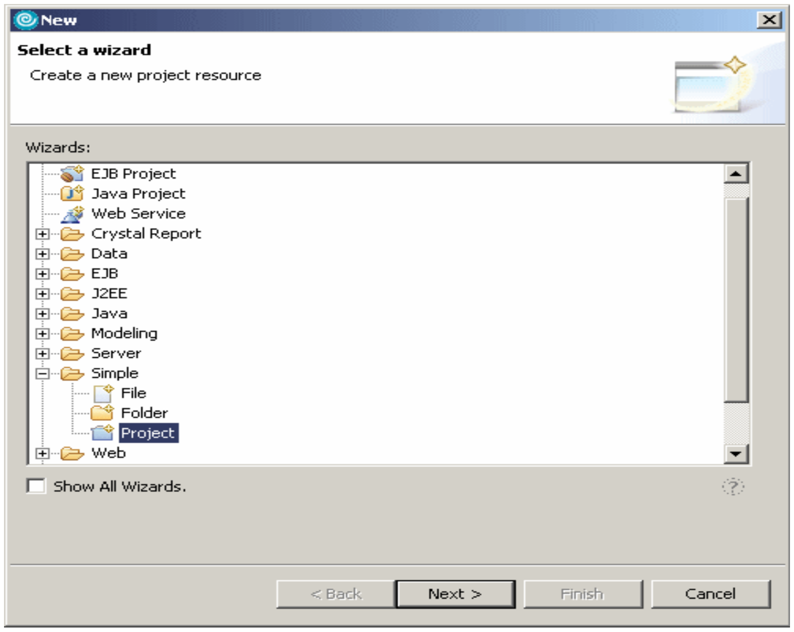
7.4 Development tools

The task of developing Web Services UDF can be simplified with the suites of IBM Development Tools such as Rational Application Developer (RAD) and WebSphere Integration Developer (WID). Now we show you how you can easily enable your database for Web service consumer UDFs, and create the `getTemperature` UDF in our previous example using RAD.

At the time of writing this redbook, DB2 V9.1 for Linux, UNIX and Windows is still beta version, and the latest version of RAD V6 currently only supports DB2 V8.1 and V8.2, but does not work with DB2 V9.1 for Linux, UNIX and Windows beta version. Therefore, we show how to create a Web service UDF using RAD V6 against a DB2 V8.2 database. Future release of RAD will support database connections to DB2 V9.1 for Linux, UNIX and Windows. Our example uses RAD V6 and DB2 V8.2 on Windows platform.

Note: In DB2 V8 you need to enable XML Extender and SOAP UDFs before proceeding with the steps below. In DB2 V9.1 for Linux, UNIX and Windows, you only need to enable SOAP UDFs and may optionally enable XML Extender if your application need backward compatibility with DB2 V8 XML Extender functions. See 11.3, “Preparing the installation of the Web services provider” on page 363 and 11.4, “Installation of the Web services consumer UDFs” on page 364 for further detail.

1. From the menu click **File** → **New** → **Project**.Then click **Simple** → **Project** to create a simple project as shown in Figure 7-7 on page 191.


New

Select a wizard

Create a new project resource

Wizards:

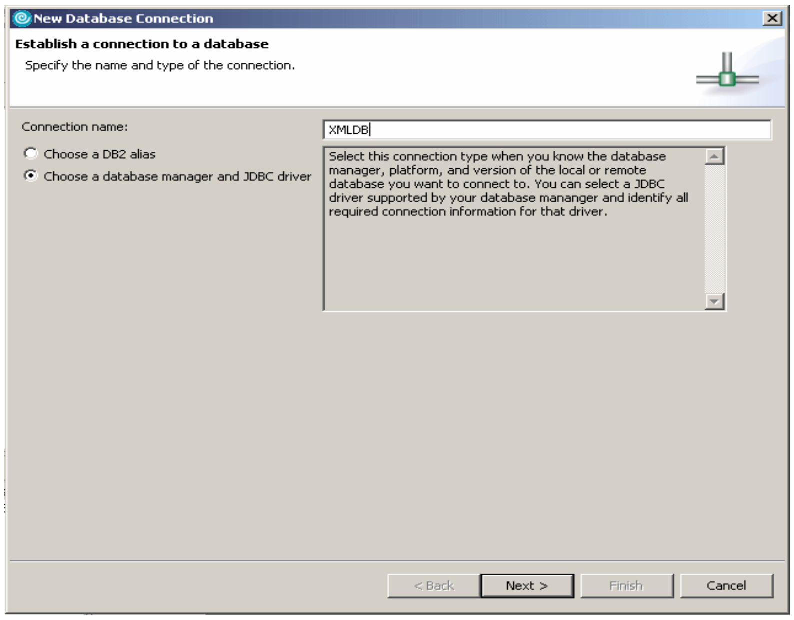
- EJB Project
- Java Project
- Web Service
- Crystal Report
- Data
- EJB
- J2EE
- Java
- Modeling
- Server
- Simple
  - File
  - Folder
  - Project
- Web

Show All Wizards.

< Back | Next > | Finish | Cancel

Figure 7-7 New Project Wizard in Rational Application Developer

2. Enter `Temperature` as your project name and click **Finish**
3. Right-click the Database Explorer view and click **New Connection**. The New Database Connection wizard opens.
4. Enter a new connection name, in this case we call our connection `XMLDB`.


New Database Connection

Establish a connection to a database

Specify the name and type of the connection.

Connection name: XMLDB

( ) Choose a DB2 alias

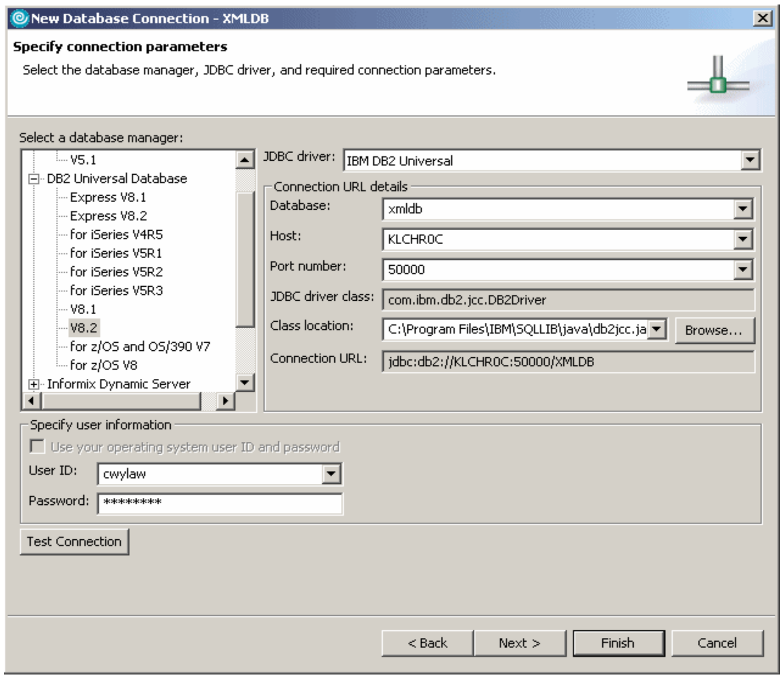
(•) Choose a database manager and JDBC driver

Select this connection type when you know the database manager, platform, and version of the local or remote database you want to connect to. You can select a JDBC driver supported by your database mananger and identify all required connection information for that driver.

< Back | Next > | Finish | Cancel

Figure 7-8 New Database Connection wizard

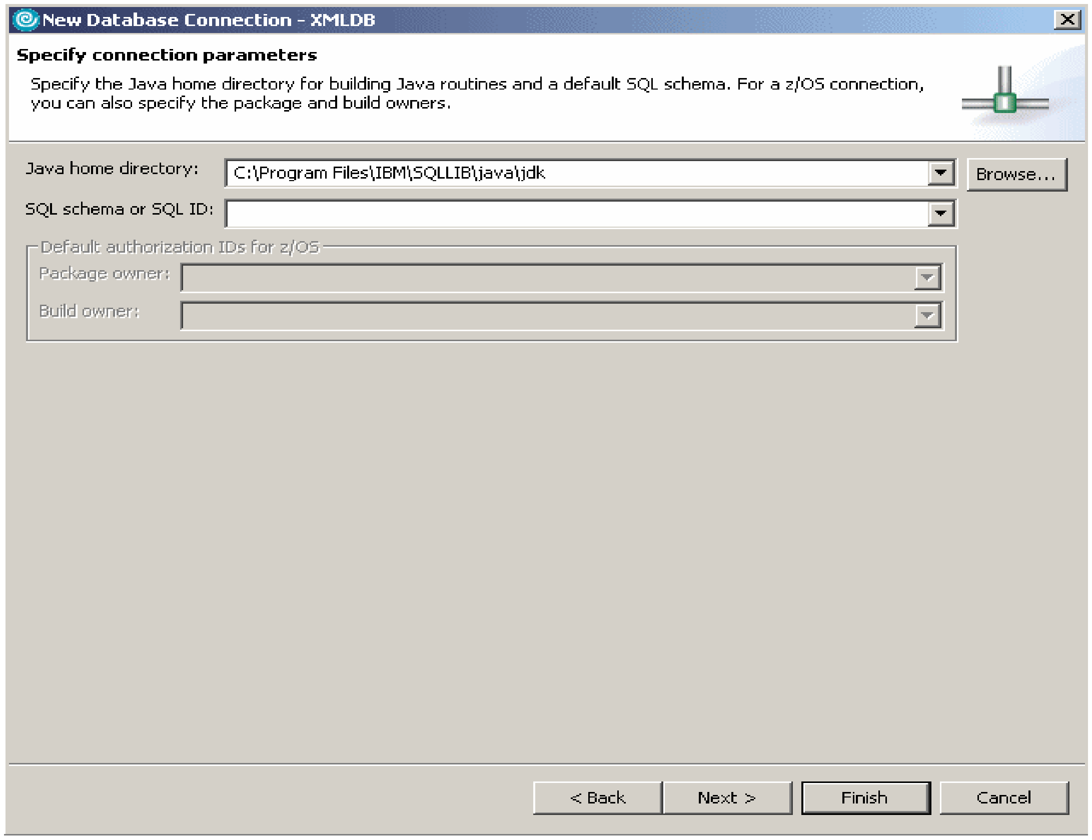
5. Click Next.



JDBC driver: IBM DB2 Universal
Database: xmldb
Host: KLCHR0C
Port number: 50000
JDBC driver class: com.ibm.db2.jcc.DB2Driver
Class location: C:\Program Files\IBM\SQLLIB\java\db2jcc.ja
Connection URL: jdbc:db2://KLCHR0C:50000/XMLDB
User ID: cwylaw
Password: ********

Figure 7-9 Connection parameters specification

6. Select **IBM DB2 Universal** in **JDBC driver** drop down list. Enter your database name, host name. Port number is 50000 by default on DB2 for Linux, UNIX and Windows. Enter your user ID and password as necessary. You may optionally choose to click **Test Connection** to confirm whether your database connection is successful before you proceed further.
7. Click Next. You will have the option to apply filters to limit the objects retrieved from the database.
8. Click **Next**. You can specify your Java home directory where your Java SDK is installed here. By default it should have automatically selected the SDK that is installed during the DB2 installation. See Figure 7-10 on page 193.


New Database Connection - XMLDB

Specify connection parameters

Specify the Java home directory for building Java routines and a default SQL schema. For a z/OS connection, you can also specify the package and build owners.

Java home directory: C:\Program Files\IBM\SQLLIB\java\jdk

SQL schema or SQL ID:

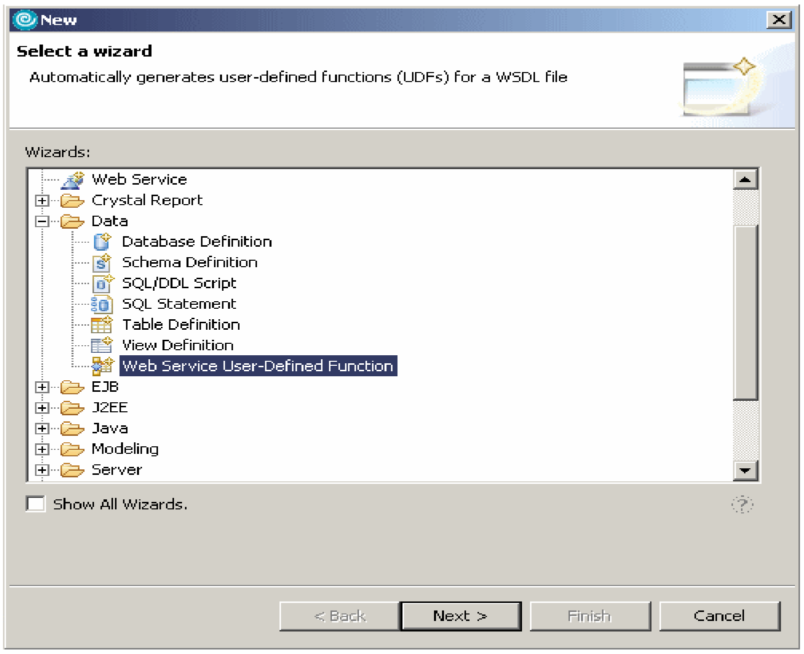
Default authorization IDs for z/OS

Package owner:

Build owner:

Figure 7-10 Specifying the home directory

9. Click **Next** to review a summary of your settings, and click **Finish** to complete the **database connection** setup. If prompted to copy database metadata to your **project** folder, click **Yes** to copy metadata to the `Temperature` project. Now our database connection is setup completely.
10. From the menu **File** → **New** → **Other**
11. Select the **Web Service User-Defined Function** wizard from the **Data** folder as shown in Figure 7-11 on page 194.


New

Select a wizard

Automatically generates user-defined functions (UDFs) for a WSDL file

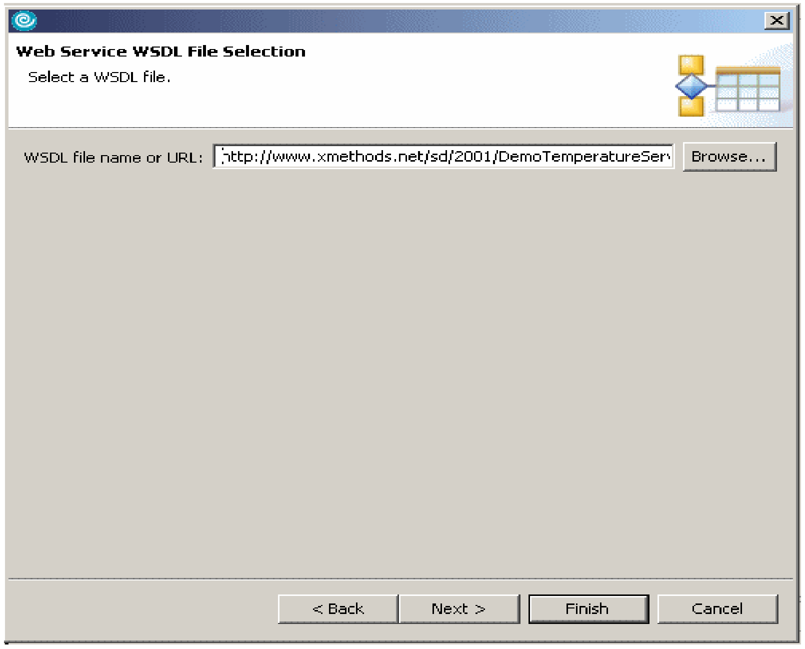
Wizards:

- Web Service
- Crystal Report
- Data
  - Database Definition
  - Schema Definition
  - SQL/DDL Script
  - SQL Statement
  - Table Definition
  - View Definition
  - Web Service User-Defined Function
- EJB
- J2EE
- Java
- Modeling
- Server

Show All Wizards.

Figure 7-11 New Web Service User-Defined Function wizard

12. Click **Next.**
13. In the WSDL file name or URL field enter the following URL:
   `http://www.xmethods.net/sd/2001/DemoTemperatureService.wsdl`


Web Service WSDL File Selection

Select a WSDL file.

WSDL file name or URL: http://www.xmethods.net/sd/2001/DemoTemperatureSer

Figure 7-12 Inputting the location of WSDL file

14. Click **Next.**
15. Click **Browse** to select a database schema from the workspace. Then we may select generate and deploy the UDF, or just generate the UDF but deploy later. If you choose to

deploy later, remember to right-click the UDF after it is generated and click **Build** before attempting to run the UDF. Otherwise, **Run** will be disabled.

Note: If you are unable to click **Browse** because it is disabled, it is likely because the data definitions is not copied into your project yet. In order to fix this, copy the data definitions by using the Copy to Project wizard. Expand the connection to show the data definitions in the DB Servers view, then click **Copy to Project**. Supply the project name in the next screen (in our case the project name is `Temperature`) then click Finish. This is shown in Figure 7-13.

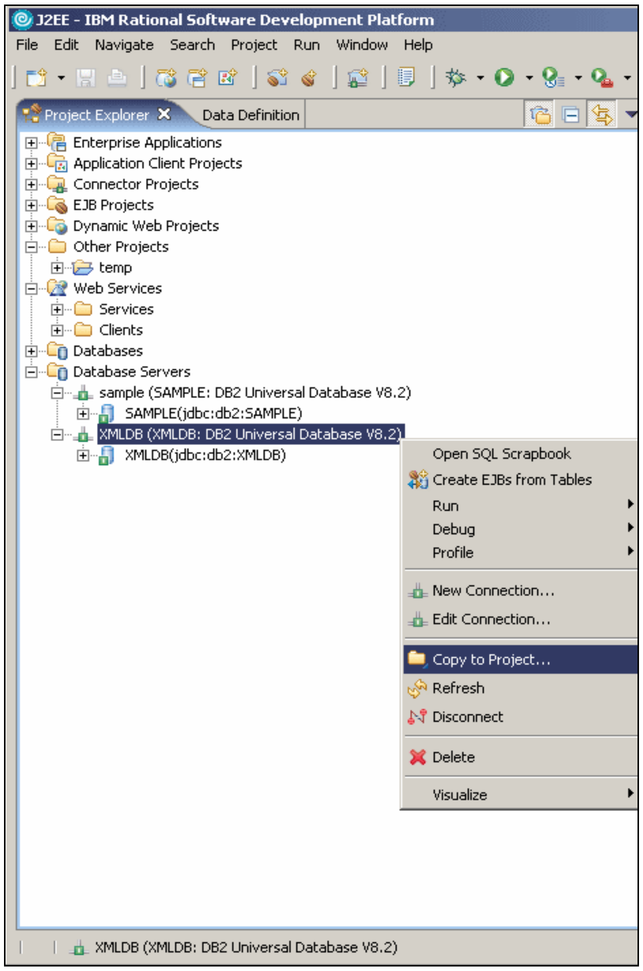
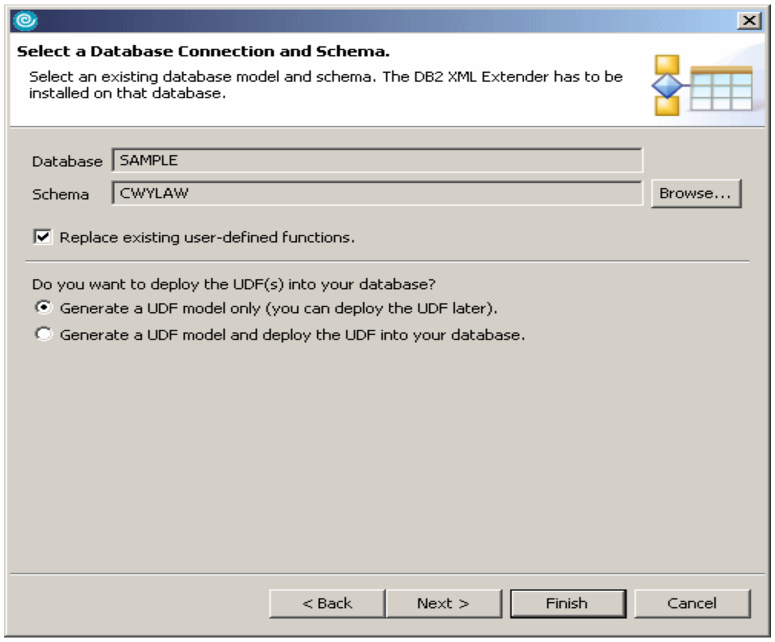


Figure 7-13 Copy the data definitions if Browse is disabled



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: SAMPLE

Schema: CWYLAW

[✓] Replace existing user-defined functions.

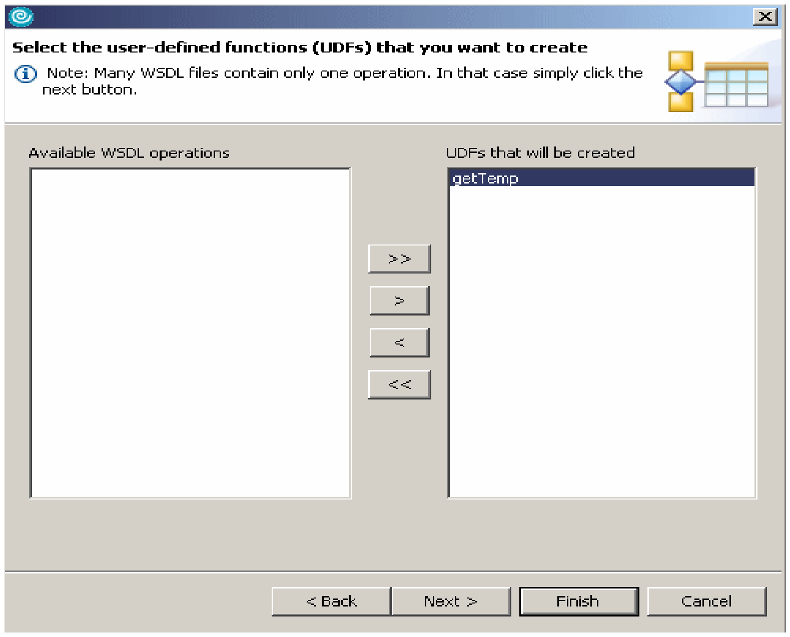
Do you want to deploy the UDF(s) into your database?

[● Generate a UDF model only (you can deploy the UDF later).]

○ Generate a UDF model and deploy the UDF into your database.

Figure 7-14 Browse and select the database schema

16. The WSDL document `DemoTemperatureService.wsdl` only provides one operation - `getTemp`, so only one UDF will be generated. If there are more operations then one UDF will be generated for each operation. Figure 7-15 shows that one UDF will be generated for the WSDL file we supplied.



Select the user-defined functions (UDFs) that you want to create

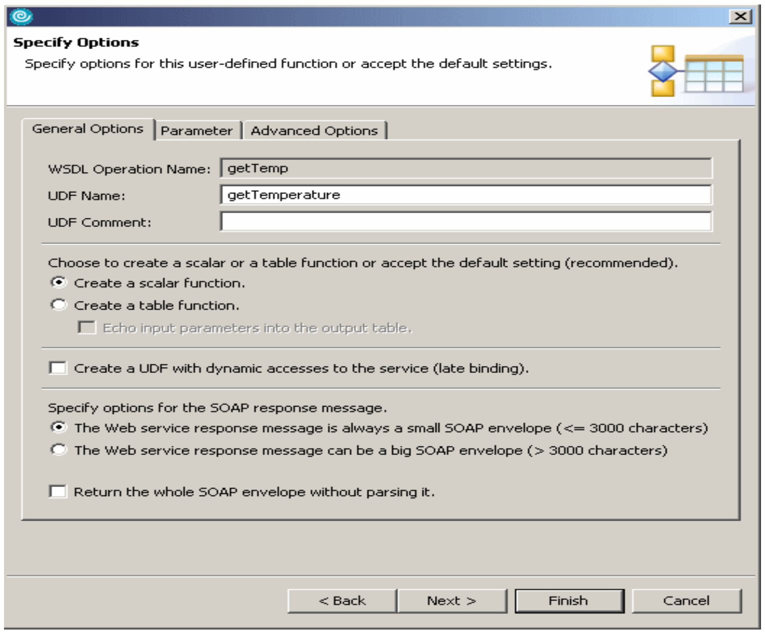
Note: Many WSDL files contain only one operation. In that case simply click the next button.

Available WSDL operations

UDFs that will be created: getTemp

Figure 7-15 Creating UDF from a WSDL file

17. In the UDF Options we specific that we are creating a scalar function. Further, we can select a small SOAP envelop since we know that our temperature Web service only returns a temperature value which is obviously less than 3000 characters. Click the **Parameters** tab to view the UDF parameters. You may also optionally specify a specific name for the UDF, but in our case we choose not to specify a specific name.


Specify Options

Specify options for this user-defined function or accept the default settings.

WSDL Operation Name: getTemp

UDF Name: getTemperature

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.

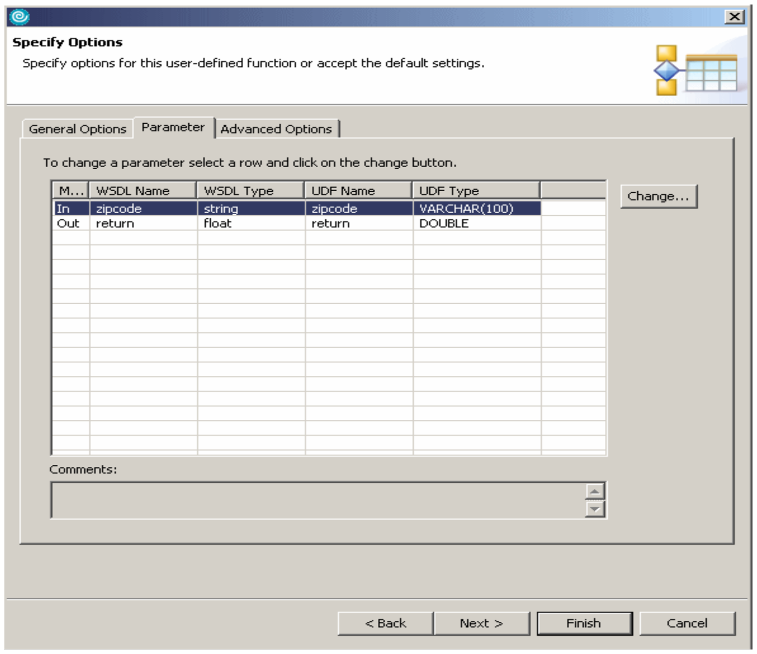
- [ ] Create a UDF with dynamic accesses to the service (late binding).

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 7-16 Specifying UDF options


Specify Options

Specify options for this user-defined function or accept the default settings.

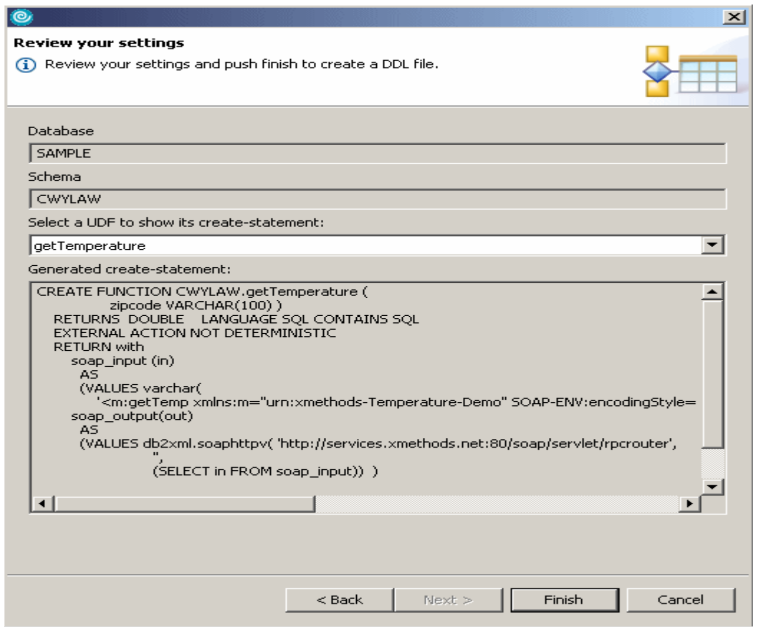
To change a parameter select a row and click on the change button.

M...	WSDL Name	WSDL Type	UDF Name	UDF Type
In	zipcode	string	zipcode	VARCHAR(100)
Out	return	float	return	DOUBLE

Comments:

Figure 7-17 Viewing the parameters of the UDF

18. Click **Next** to review the UDF settings, and then click **Finish** to complete generating the UDF.



Review your settings

Review your settings and push finish to create a DDL file.

Database: SAMPLE

Schema: CWYLAW

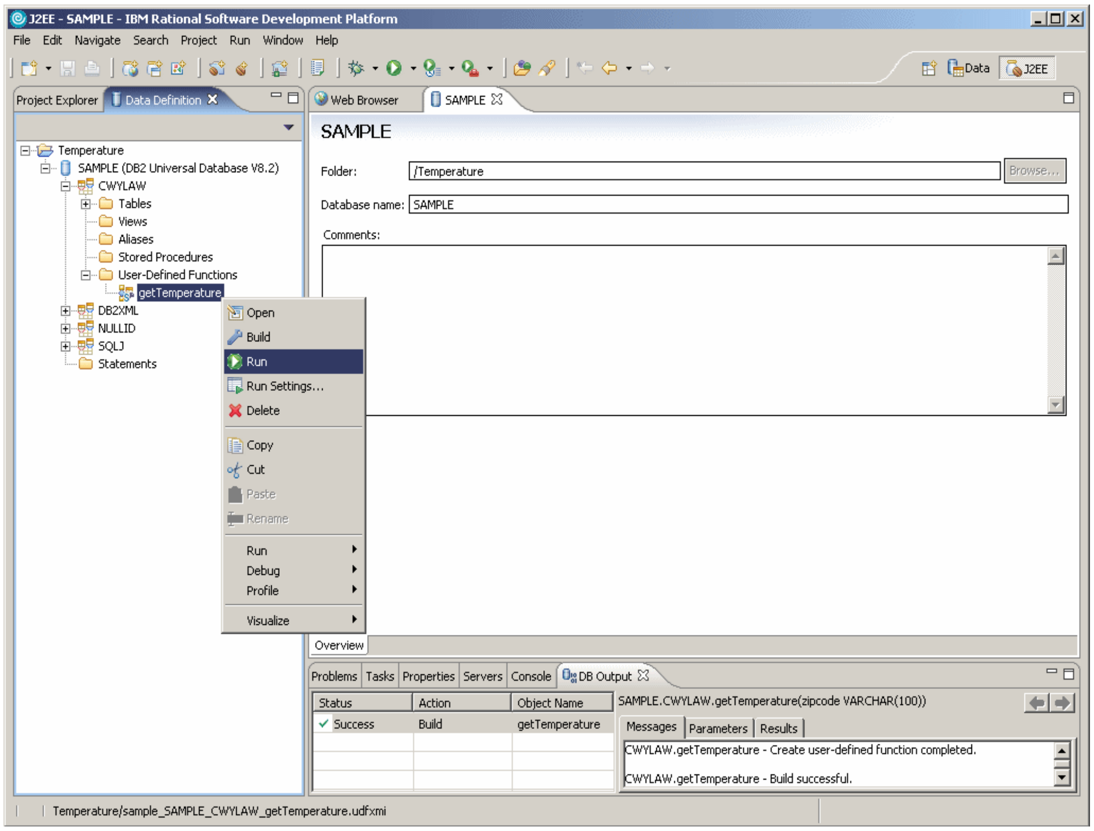
Select a UDF to show its create-statement: getTemperature

Generated create-statement:

```
CREATE FUNCTION CWYLAW.getTemperature (
            zipcode VARCHAR(100) )
   RETURNS  DOUBLE    LANGUAGE SQL CONTAINS SQL
   EXTERNAL ACTION NOT DETERMINISTIC
   RETURN with
      soap_input (in)
        AS
        (VALUES varchar(
           '<m:getTemp xmlns:m="urn:xmethods-Temperature-Demo" SOAP-ENV:encodingStyle=
      soap_output(out)
        AS
        (VALUES db2xml.soaphttpv( 'http://services.xmethods.net:80/soap/servlet/rpcrouter',
                    '',
                    (SELECT in FROM soap_input))  )
```

Figure 7-18   Reviewing UDF settings

19. Now we can run the UDF to test it. If you have chosen to deploy later instead of deploy immediately in Step 10 you need to click **Build** first before you can run the UDF.



SAMPLE

Folder: /Temperature

Database name: SAMPLE

Comments:

Status	Action	Object Name
✓ Success	Build	getTemperature

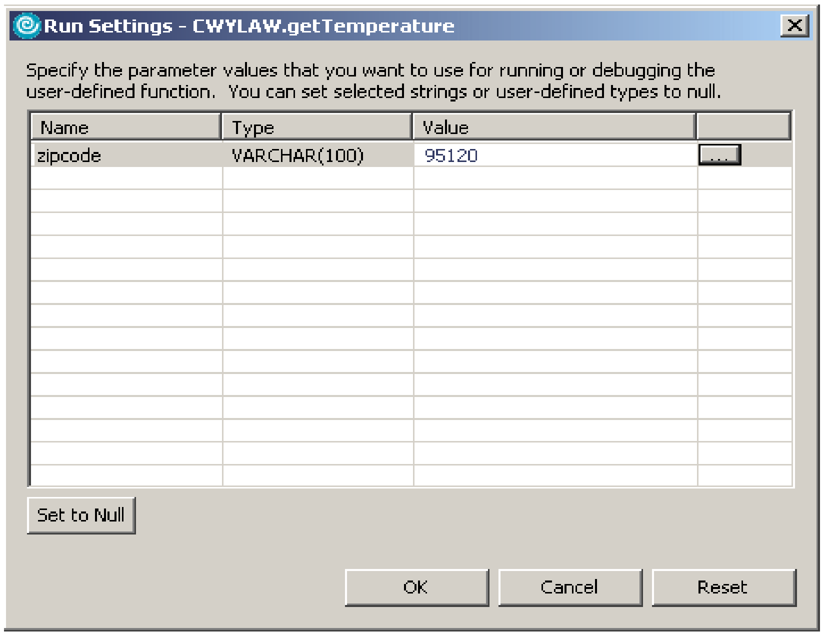
SAMPLE.CWYLAW.getTemperature(zipcode VARCHAR(100))

```
CWYLAW.getTemperature - Create user-defined function completed.
CWYLAW.getTemperature - Build successful.
```

Temperature/sample_SAMPLE_CWYLAW_getTemperature.udfxmi

Figure 7-19   Run the UDF to test it in RAD

20. We supplied the zip code of `95120` to test the `getTemperature` UDF.


Run Settings - CWYLAW.getTemperature

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	
zipcode	VARCHAR(100)	95120	...

Figure 7-20 Testing the getTemperature UDF

In Chapter 15, “Developing SOA access services” on page 401, we present a detail user scenario that shows you how to use Rational Application Developer to develop DB2 Web Services. The user scenario will give you more information in regards to exposing DB2 business logic as Web services, as well as how to consume Web services.

8

IMS and SOA

IMS has been around for many years and it is currently at Version 9. Many customers have made major investments in IMS based applications. Traditionally IMS applications were written in third generation languages (3GL), like PL/I, Cobol and C. Recently it has become possible to write applications also in Java, both for message (JMP region) and batch processing (JBP region).

IMS provides a variety of solutions for providing access to IMS applications and data as Web services.

IMS also provides the ability to store and retrieve XML data natively. IMS converts non-XML data to XML for interchange and converts it back or stores it natively.

When customers need rapid response for business transactions and inquiries from many locations using numerous types of devices, the IMS Connect function provides transparent access to IMS applications and data from practically any application environment. Customers can use their storehouse of IMS applications from IMS Connect client applications to access their IMS and IBM DB2 data from the Internet. IMS Connect is now integrated with IMS and acts as a client of the Open Transaction Manager Access (OTMA), a transaction-based, connectionless client/server protocol that provides an access path and an interface specification for sending and receiving transactions and data from IMS.

Various IBM-supplied connectivity solutions, such as IMS Connector for Java or IMS SOAP Gateway, are based on the use of IMS Connect.

In this chapter we discuss solutions based on Web Services with SOAP Gateway as well as IMS Services through WebSphere.

The chapter contains these topics:

- Introduction
- IMS Connect
- Web Services with SOAP Gateway
- IMS Services through WebSphere
- Accessing DLI data

© Copyright IBM Corp. 2006. All rights reserved.

8.1 Introduction

Although we always mention IMS as one product, in reality, it has two distinct components (see Figure 8-1):

- Database Manager for DLI databases (hierarchical databases)

  Those databases can also be accessed by other subsystems like CICS, WebSphere Application Server through a database resource adapter (DRA). Recently IBM made also JDBC access available for those databases.

- Data Communication Manager

  Access via transactions over several communication protocols. Programs dispatched in IMSDC as a result of those transactions, can access the DLI databases but also DB2 databases.

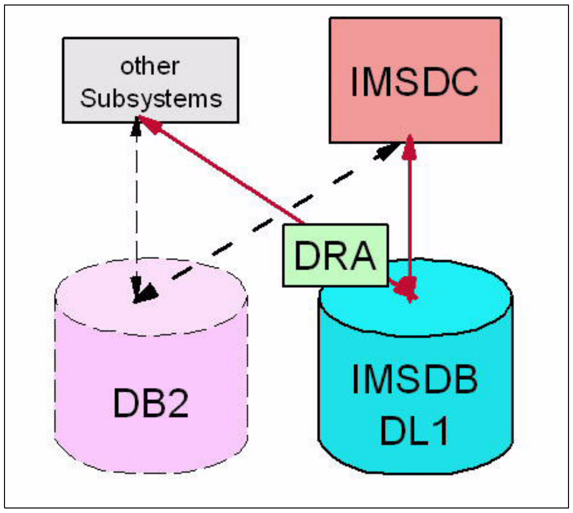


Figure 8-1 IMS composed of IMSDC and IMSDB

The IMS Open Database Access (ODBA) provides a callable interface to IMS databases from z/OS programs that are not managed by IMS, such as DB2 stored procedures or any z/OS application which uses the Resource Recovery Services (RRS) to manage their sync point processing.

Both DLI access and transactional access can be integrated in a service-oriented architecture (SOA). The ultimate goal is to have this integration through the Service Component Architecture (SCA) binding, which is supported by the WebSphere Integration Development (WID) tooling and deployed in the WebSphere Portal Server (WPS). A SCA binding describes the physical connection protocol of components. Services that can be accessed include Plain Old Java Objects (POJOs), EJBs, Web services, JMS messages, and adapters.

We introduce some technologies that make this integration possible for IMS, and also the tooling which can be used to quickly build the artifacts.

Besides the base product IMS V9, other software components are used to implement the solutions. Figure 8-2 on page 203 shows the products that we consider:

- IMS Connect is part of IMS V9, it is the gateway on z/OS
- IMS Connect for Java provides Java classes and JNI code to build IMS transaction connectors through the IMS connect gateway. This can be used in a non-J2EE and in a J2EE (WebSphere) environment.

- IMS JDBC Connector allows direct access to DLI databases using JDBC.
- IMS SOAP Gateway, only available on Windows, provides Web service access for IMS transactions through IMS Connect without WebSphere requirement.

Figure 8-2 IMS connectivity functions

More information about these components and downloads can be found at:

`http://www-306.ibm.com/software/data/ims/`

In the next sections we describe and implement the following solutions.

- Web Services (over SOAP) without WebSphere
- **Web Services** through the WebSphere Application Server
  - J2C Web service built with RAD wizard
  - J2C Import built with WID wizard, this is an SCA component
  - J2C EJB built with RAD wizard, and then used as an imported SCA component.
- **Direct** access to DLI data through DLI/JDBC adapter.

The first two solutions are both based on IMS Connect, which starting with IMS V9 is integrated in the IMS product. We start with a quick overview of IMS Connect.

8.2 IMS Connect

IMS Connect is the building block to be used when we want to have existing IMS transactions accessible from within intelligent clients, written in C, Java or Web services. We have to build a connector. Connector code takes the role of the good old terminal like 3270.

The integrated IMS Connect function in IMS Version 9 provides easy-to-install, easy to use, high-performance, high-volume, and secure transparent access to IMS applications and operations from any TCP/IP-supported environment. It provides commands to manage the network environment and assist with workload balancing, resulting in better resource utilization. It reduces the design and coding effort for client applications and provides easier access to IMS applications and operations, thereby improving programmer productivity. It can be used with IBM WebSphere and Rational development tools to quickly transform static Web sites into sources of dynamic Web content, improving marketing effectiveness and customer service, and to transform IMS transactions into Web services for service-oriented architecture (SOA), enabling quick response to new customer requirements, business opportunities, and competitive threats.

IMS Connect runs in its own address space. It connects to IMS via an high speed internal Open Transaction Manager Access (OTMA) interface, while clients connect to it via TCP/IP or via a local option if the client is on the same z/OS.

This local option has been deprecated, so we only consider the TCP/IP (socket) option. The redbook *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794, explains all details about the implementation of IMS Connect.

Clients drive IMS Connect and receive their responses through frames which are composed of headers and data. Composing and understanding those frames is rather complex. Although it can be done via C and Java client coding, it is much more efficient to use tools to build and guide the client artifacts (like connectors and proxies).

The header contains information related to the client-server exchange protocol, such as commit and synchronization level. The data layout is in a format as expected by, or sent from the IMS transactional program.

The setup used in our examples is very simple. It uses an IMS/V9 system, with subsystem name IMSH. See Figure 8-3.

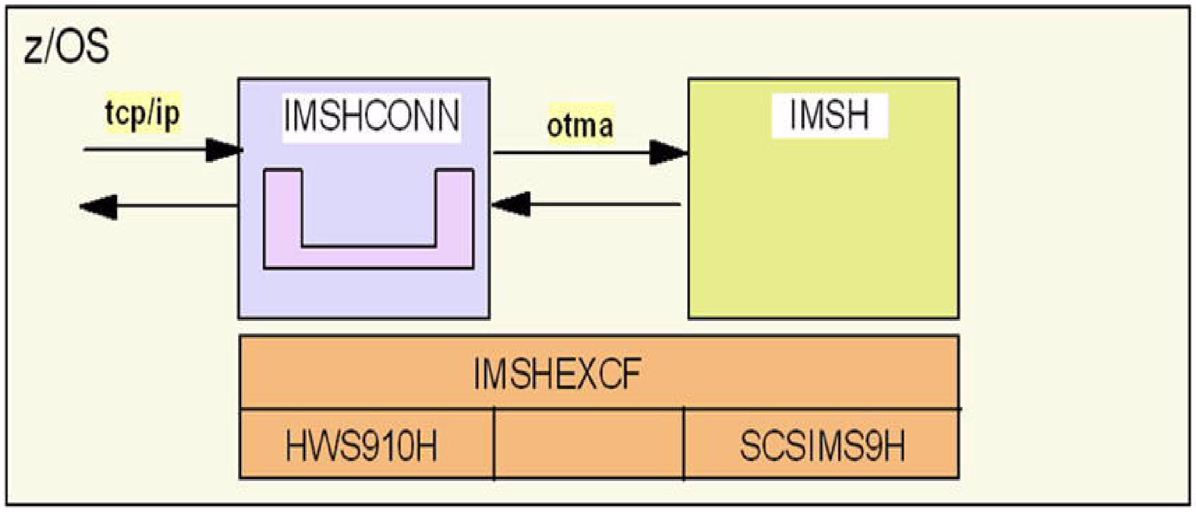


Figure 8-3 IMS Connect

This system is front-ended with IMS Connect IMSHCONN. Both have to be member of the same XCFOTMA group IMSHEXCF for communication. Within the XCFOTMA group IMS is called SCSIMS9H, and IMS Connect has member name HWS910H. We show the IMS Connect configuration statements in Example 8-1.

Example 8-1 IMS Connect configuration

```
HWS (ID=IMSHCONN,RACF=N,XIBAREA=20,RRS=Y)
TCPIP (HOSTNAME=TCPIP,PORTID=(6001,LOCAL),MAXSOC=2000,TIMEOUT=3000)
DATASTORE (ID=IMSH,GROUP=IMSHEXCF,MEMBER=HWS910H,TMEMBER=SCSIMS9H)
```

8.2.1 IMS example

In our example we tried to include as much as possible elements which are different from the traditional examples. We include:

- MPP(JMP) in Java
- Input/output message layouts (in C and Cobol (when required))
- DLI HDAM database: accessed by JDBC/DLI

The example is a simple customer query case (transaction **CQUERY**).

All the IMS definitions are listed in Appendix D, “Setting up IMS services” on page 665.

8.3 Web Services with SOAP Gateway

IMS SOAP Gateway is a Web services solution that enables IMS applications to interoperate outside of the IMS environment through Simple Object Access Protocol (SOAP) to provide and request services independently of platform, environment, application language, or programming model. This solution does NOT use the WebSphere Application Server, and is available for Windows and Unix (not z/OS).

IMS SOAP Gateway allows you to enable your IMS application to become a Web service. Different types of client applications, such as Microsoft .NET, Java and third party applications, can submit SOAP requests into IMS to drive the business logic of your IMS applications.

IMS SOAP Gateway is compliant with the industry standards for Web services, including SOAP/HTTP 1.1 and WSDL 1.1. Currently it only exists for Windows.

IMS SOAP Gateway consists of two main components:

- IMS SOAP Gateway deployment utility

  The end-to-end deployment utility enables you to set up properties and create runtime code that IMS SOAP Gateway uses to enable IMS applications as Web services.

- IMS SOAP Gateway server

  The IMS SOAP Gateway server processes SOAP messages. It receives the SOAP message from the client application, converts it to an IMS input message, and sends it to IMS through IMS Connect. It then receives the output message from IMS and converts it to a SOAP message to send back to the client.

Two important enhancements have been introduced in the new version of the SOAP gateway

- There is **no** need anymore to change the IMS application programs. The IMS SOAP gateway takes care of the XML [de]marshalling, so that the IMS application is unaware of the usage of SOAP.
- The connection between the Web service client and the Gateway can be over HTTPS (HTTP over SSL). This is independent of the user ID/password which is transported in the SOAP envelope and is passed to IMS Connect.

IBM WebSphere Developer for zSeries (WDz) is an application development tool that helps the development of traditional mainframe applications. WDz version 6 provides the XML Services for the Enterprise (XSE) feature that helps you to easily generate the artifacts needed to transform your IMS application into a Web Service to be used with the IMS SOAP Gateway runtime. With a step as simple as taking a COBOL copybook for your IMS application, describing the input and output message format, you can generate the following Web service artifacts:

- Web Services Description Language (WSDL) file, which provides a Web service interface of the IMS application so that the client can communicate with the Web service.
- COBOL converters and driver file, which help you to transform the XML message from the client into COBOL bytes for the IMS application and then back to XML.
- Correlator file, which contains information that enables IMS SOAP Gateway to set IMS properties and call the IMS application.

The solution is shown in Figure 8-4.

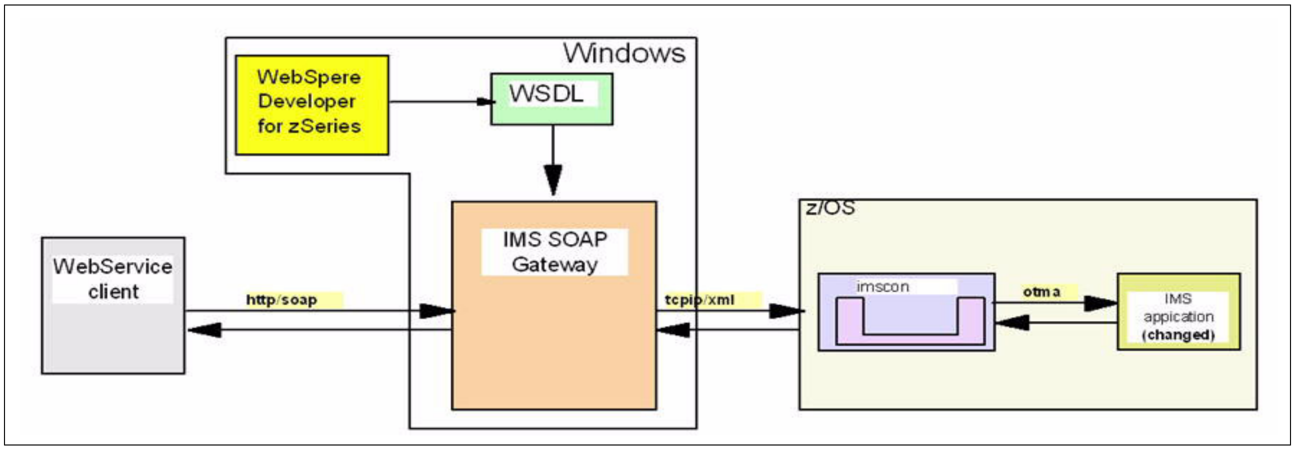


Figure 8-4 Soap Gateway

We now we give an overview of the tasks required to enable an IMS application as a Web service using IMS SOAP Gateway. After you create the WSDL file, modifying the IMS application, and deploy the IMS SOAP Gateway Web service, the IMS application is available as a Web service and the client application can send SOAP messages.

8.3.1 Creating the WSDL file

To deploy an IMS application as a Web service, you need to create a WSDL (Web service description language) file. A WSDL file is an XML document that describes a Web service. WSDL files are used by others (for example, the client that invokes the service) to discover the service and to understand how to invoke the service. It specifies the location of the service and the operations that the service exposes. To make your IMS application accessible as a Web service, you need to describe the functions provided by the IMS application and how the input and output message looks like for invoking the function in this WSDL file. The WSDL file serves as the Web service interface for the IMS application.

There are three ways to create the WSDL file:

- Generating the file by using IBM WebSphere Developer for zSeries
- Manually creating the file
- Modifying an existing WSDL file that is generated by IBM WebSphere Application Developer Integration Edition V5.1.1

To prepare the WSDL file, you need a COBOL copybook that describes the input and output messages for the IMS application.

The WSDL document, which in a general way is used to describe Services (not only SOAP) is composed of five parts, which, depending on the options, are located in three physical files or less. See Figure 8-5.

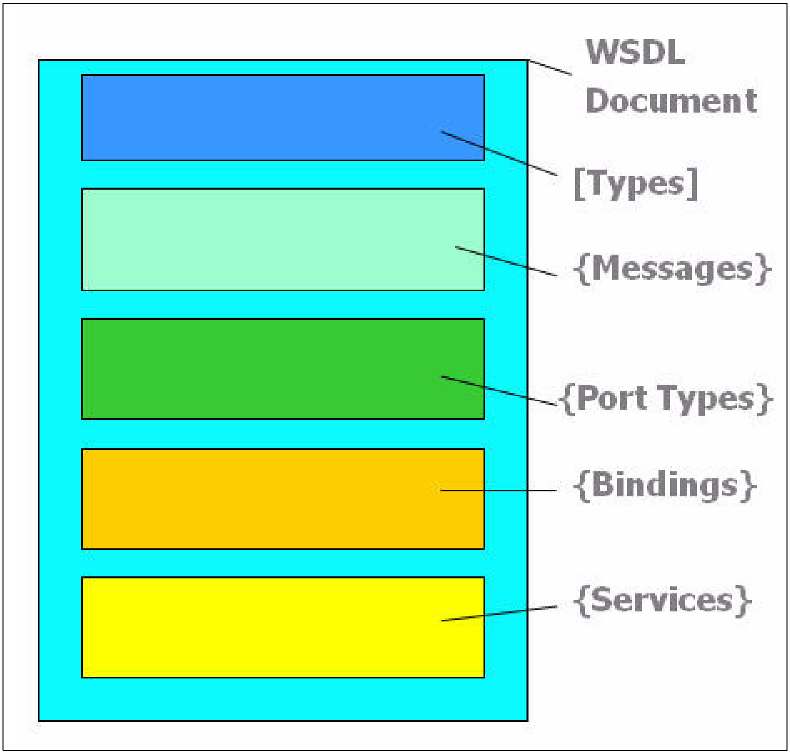


Figure 8-5 WSDL contents

The five parts are:

1. Definition of parameter types, if composed (structures, objects and so on.)
2. Definition of the input and output messages. A message means the aggregation of all parameters.
3. Abstract definition of the operations available on this port type, and input and output message used by them
4. Binding to a particular protocol implementation (HTTP/SOAP, JMS/SOAP, JMS, EJB, Plain Old Java Object (POJO))
5. Service Endpoint: the point where the service is available (like URL)

8.3.2 Enabling the z/OS Developer role

We assume that you have the product IBM WebSphere Developer for zSeries v6.0.1 installed on your workstation. If Rational Application Developer (RAD) is already installed then it installs in the same workbench.

Before doing any work, after starting the tool, you have to be sure that the **z/OS Modernization Developer** role has been enabled.

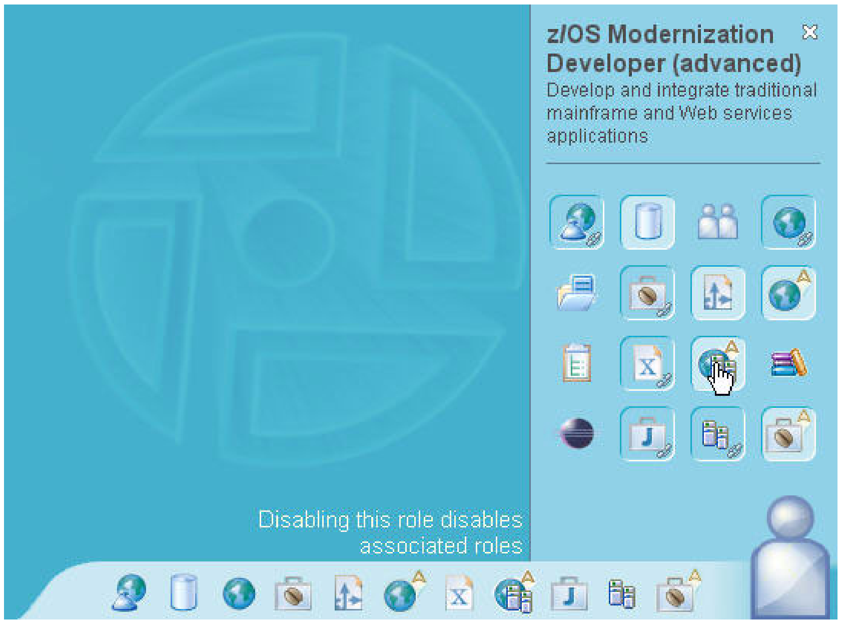

z/OS Modernization Developer (advanced)
Develop and integrate traditional mainframe and Web services applications
Disabling this role disables associated roles

Figure 8-6 Enable the z/OS Modernization Developer Role

8.3.3 Generating a WDSL file using WebSphere Developer for zSeries

With WebSphere Developer for zSeries, you can generate a WDSL file that you can use without modification for IMS SOAP Gateway.

As a prerequisite, to generate a WSDL file, you must have a COBOL copybook that describes the format of the input and output messages for the application.

To generate a WSDL file using WebSphere Developer for zSeries V6.0.1:

- Start WebSphere Developer for zSeries and open the z/OS Projects perspective.
- Select the z/OS Projects view.
- Create a local COBOL project.
  - Select **File** → **New** → **Project.**
  - Expand the Workstation COBOL or PL/I folder.
  - Select Local Project, as shown in Figure 8-7 and click **Next**.
  - Type a name for the project. For example, **IMSv92006SOAPExt**. Click **Finish**.

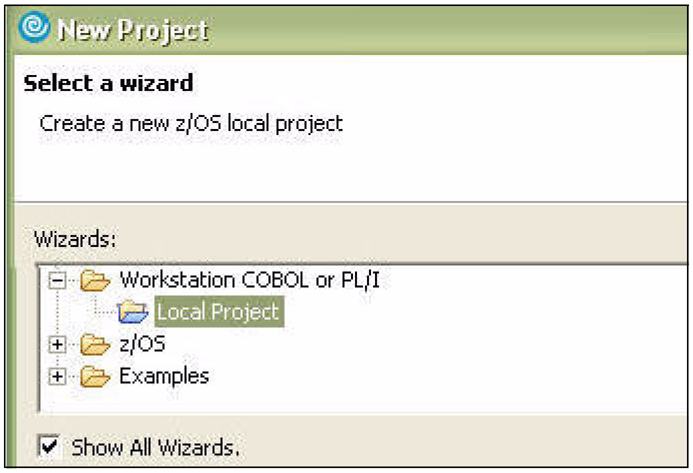


Figure 8-7 Selecting a z/OS project

A new project *IMSv92006SOAPExt* is now available in your workspace.

Import the COBOL copybook that describes the format of the input and output messages of your IMS application into the project you created:

- Select File → **Import** to open the Import wizard.
- Select File system to import the resources from the local file system. Click **Next**.
- Click Browse beside the directory field to go to the directory that contains the COBOL copybook. Click **OK**.
- Ensure IIMSv92006SOAPExt is the name of the destination folder for the imported resource.
- Click Finish to import the file and close the wizard.

The layout of the Cobol copybook, representing input/output IMS messages, is shown below.

Example 8-2 input/output IMS messages Cobol copybook

```
IDENTIFICATION DIVISION.
        program-id. pgm1.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        DATA DIVISION.
      *
      *    IMS Connector for Java, COBOL Transaction Message Source
      *
LINKAGE SECTION.

        01  INDATA.
            02  IN-LL          PICTURE S9(3) COMP.
            02  IN-ZZ          PICTURE S9(3) COMP.
            02  IN-TRCD        PICTURE X(8).
            02  CUSTOMERNR     PICTURE X(10).
            02  TRACEX         PICTURE X(1).

        01  OUTDATA.
            02  OUT-LL         PICTURE S9(3) COMP VALUE +0.
            02  OUT-ZZ         PICTURE S9(3) COMP VALUE +0.
            02  CUSTOMERNR     PICTURE X(10).
            02  SSN            PICTURE X(11) VALUE SPACES.
            02  FIRSTNME       PICTURE X(20) VALUE SPACES.
            02  MI             PICTURE X(2) VALUE SPACES.
            02  LASTNAME       PICTURE X(40) VALUE SPACES.
            02  SALUTAT        PICTURE X(10) VALUE SPACES.
            02  ADDRESS1       PICTURE X(40) VALUE SPACES.
            02  ADDRESS2       PICTURE X(40) VALUE SPACES.
            02  CITY           PICTURE X(30) VALUE SPACES.
            02  STATE          PICTURE X(4) VALUE SPACES.
            02  ZIPCD          PICTURE X(10) VALUE SPACES.
            02  PHONE          PICTURE X(20) VALUE SPACES.
            02  FAX            PICTURE X(20) VALUE SPACES.
            02  EMLADDR        PICTURE X(40) VALUE SPACES.
            02  MESSAGEX       PICTURE X(200) VALUE SPACES.

        PROCEDURE DIVISION.
```

Use the Web Service Runtime and Scenario Selection dialog to start the appropriate XML services for the Enterprise wizard.

- Highlight and right-click the COBOL copybook and select **Enable Enterprise Web Service**. A dialog opens where you can specify the options to start a Web service wizard.

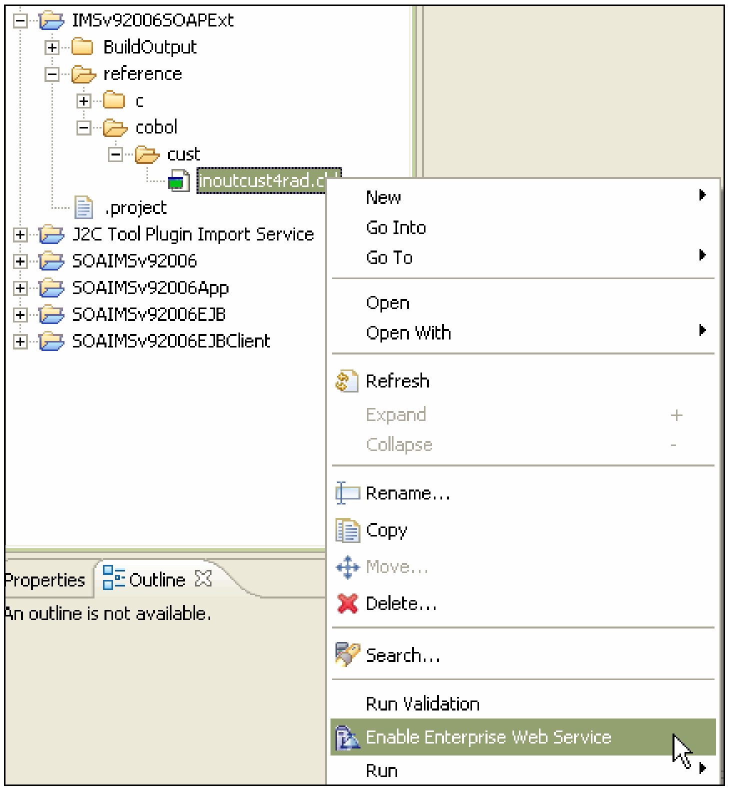
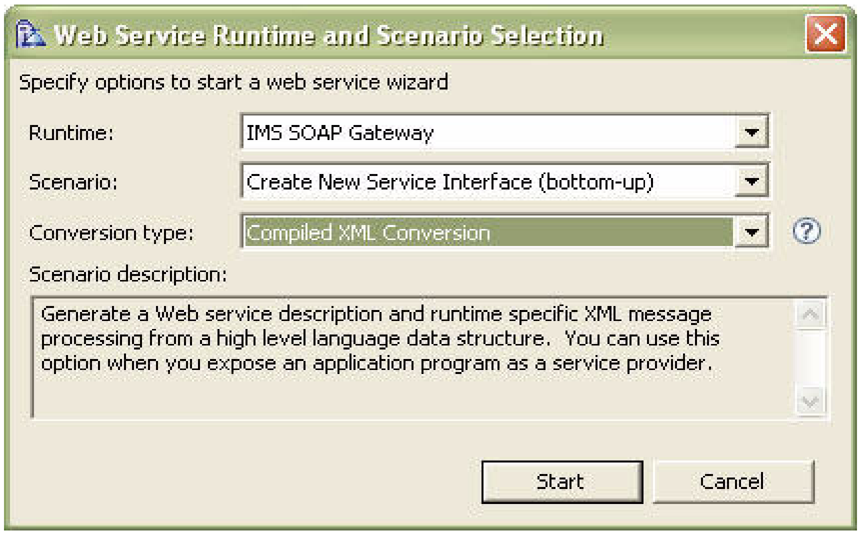


Figure 8-8 Enabling Web Service

- Select a Runtime, Scenario and Conversion as in Figure 8-9 on page 211:
  - **Runtime:** IMS SOAP Gateway
  - **Scenario:** Create New Service Interface (bottom-up)
  - **Conversion type:** Compiled XML Conversion.


Web Service Runtime and Scenario Selection

Specify options to start a web service wizard

Runtime: IMS SOAP Gateway

Scenario: Create New Service Interface (bottom-up)

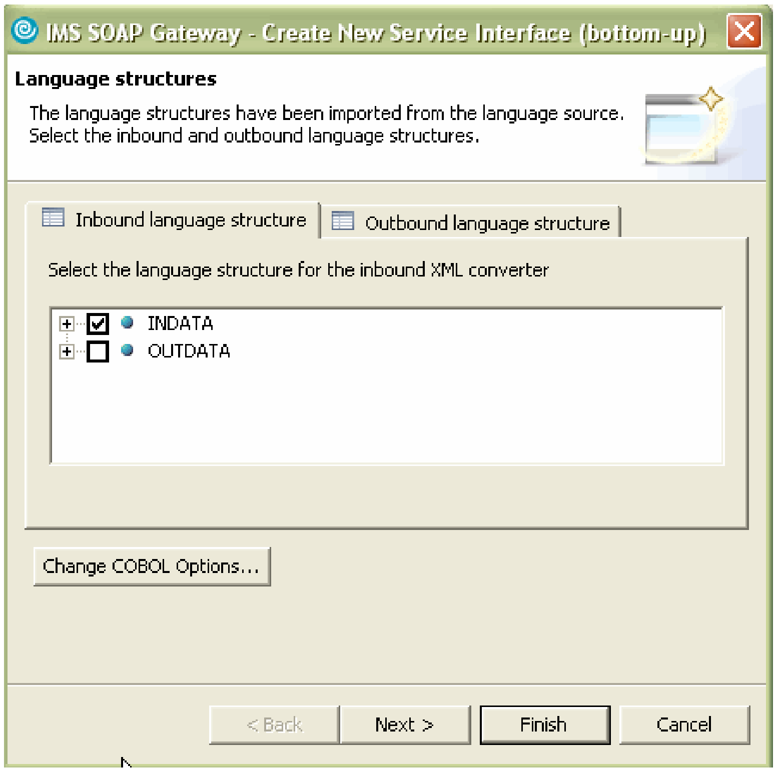
Conversion type: Compiled XML Conversion

Scenario description:

Generate a Web service description and runtime specific XML message processing from a high level language data structure. You can use this option when you expose an application program as a service provider.

Figure 8-9 Options for the Gateway definition

- Click **Start.**


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Language structures

The language structures have been imported from the language source. Select the inbound and outbound language structures.

Inbound language structure | Outbound language structure

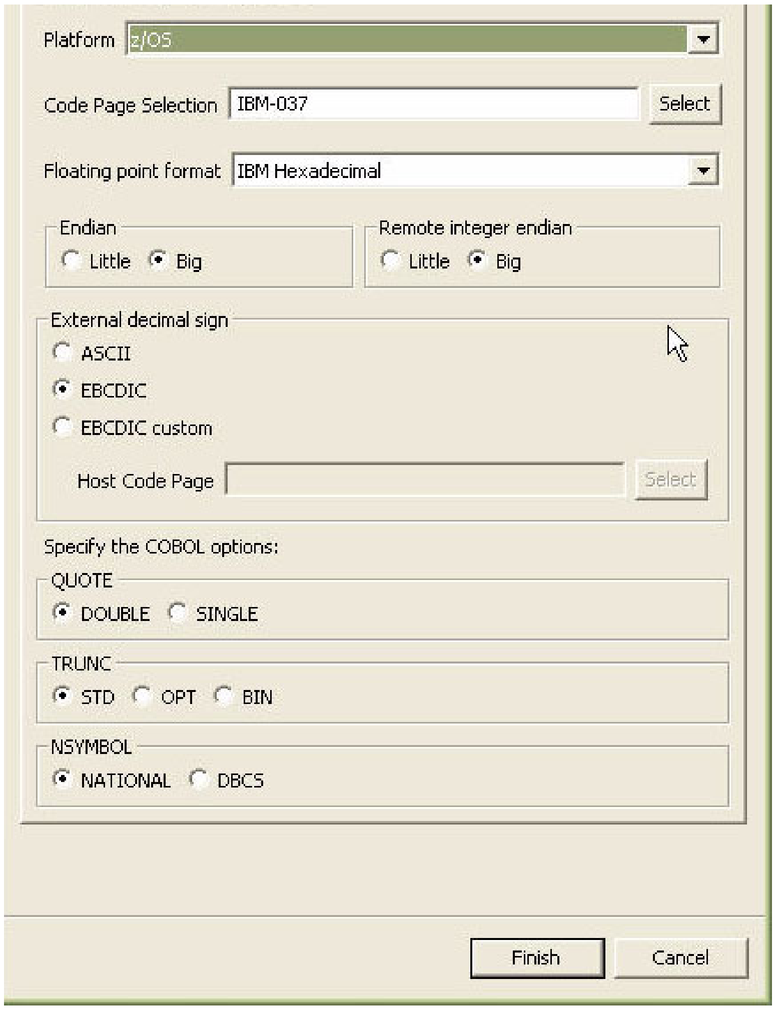
Select the language structure for the inbound XML converter

[✓] INDATA

[ ] OUTDATA

Figure 8-10 Change COBOL options and select Input, Output

- Set COBOL options first by clicking **Change COBOL options.**


Platform z/OS

Code Page Selection IBM-037

Floating point format IBM Hexadecimal

Endian: ( ) Little (•) Big

Remote integer endian: ( ) Little (•) Big

External decimal sign: ( ) ASCII (•) EBCDIC ( ) EBCDIC custom

Host Code Page

Specify the COBOL options:

QUOTE: (•) DOUBLE ( ) SINGLE

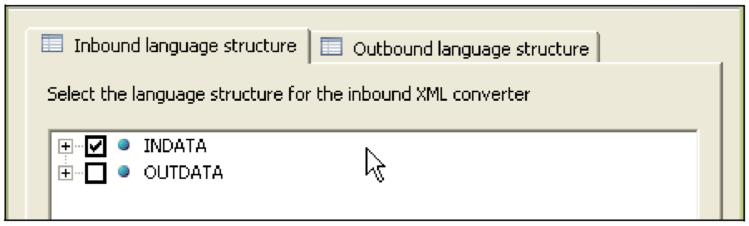
TRUNC: (•) STD ( ) OPT ( ) BIN

NSYMBOL: (•) NATIONAL ( ) DBCS

Figure 8-11 Target Platform Options

Set the options as indicated in Figure 8-11 and click **Finish.**

The Data structures panel appears.


Inbound language structure | Outbound language structure

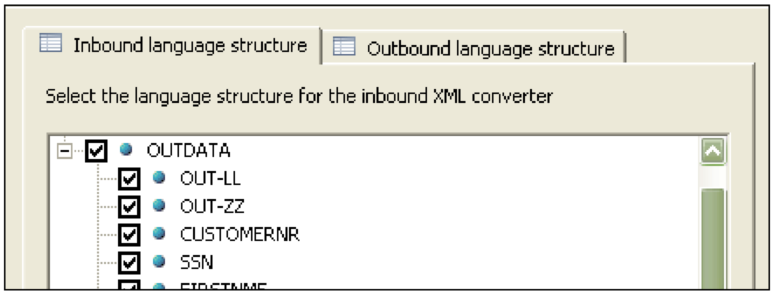
Select the language structure for the inbound XML converter

- [✓] INDATA
- [ ] OUTDATA

Figure 8-12 INDATA structure selection

For the Inbound data structure, select the COBOL data structure **INDATA** that corresponds to the input message of the IMS application.

Switch to the Outbound data structure tab.


Inbound language structure | Outbound language structure

Select the language structure for the inbound XML converter

- ☑ OUTDATA
  - ☑ OUT-LL
  - ☑ OUT-ZZ
  - ☑ CUSTOMERNR
  - ☑ SSN

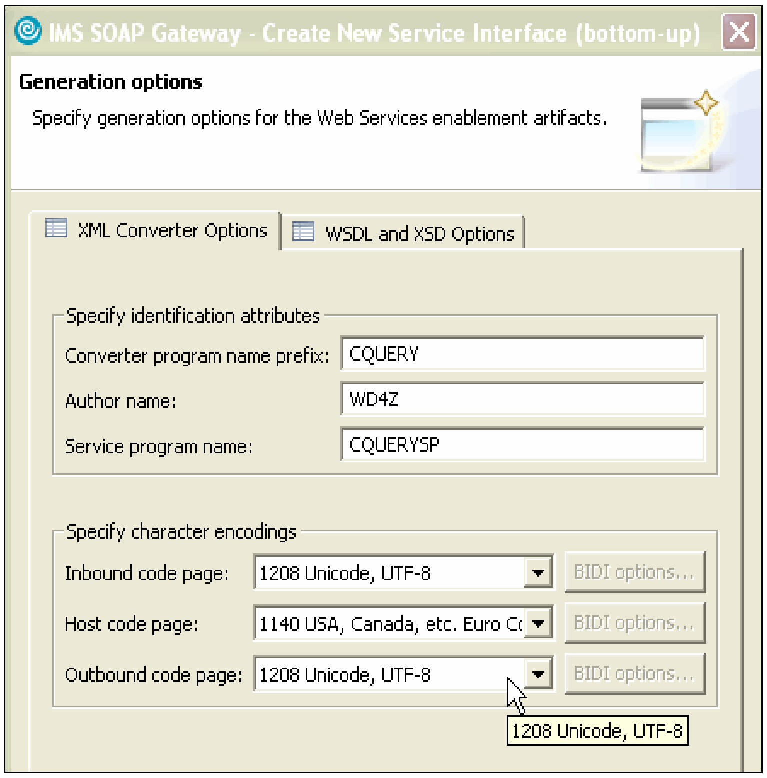
Figure 8-13 OUTDATA structure selection

Select the COBOL data structure **OUTDATA** that corresponds to the output message of the IMS application.

Click **Next** to continue.

In the panel of Figure 8-14, specify the generation options:

- In the Host code page field, select the code page that the host uses **1140**.
- Be sure to have **UTF-8** encoding for the inbound and outbound code pages.
- Specify any additional properties and names.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Generation options

Specify generation options for the Web Services enablement artifacts.

XML Converter Options | WSDL and XSD Options

Specify identification attributes

Converter program name prefix: CQUERY

Author name: WD4Z

Service program name: CQUERYSP

Specify character encodings

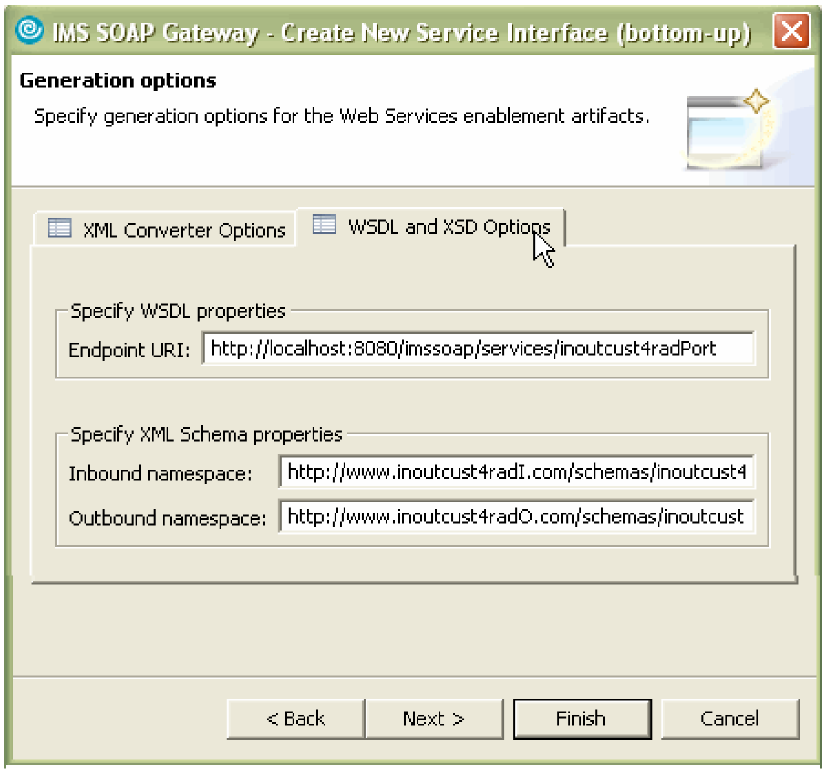
Inbound code page: 1208 Unicode, UTF-8

Host code page: 1140 USA, Canada, etc. Euro Cc

Outbound code page: 1208 Unicode, UTF-8

Figure 8-14 XML Converter options

Select the WSDL and XSD Options tab and the panel of Figure 8-15 on page 214 appears.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

Generation options

Specify generation options for the Web Services enablement artifacts.

Specify WSDL properties

Endpoint URI: http://localhost:8080/imssoap/services/inoutcust4radPort

Specify XML Schema properties

Inbound namespace: http://www.inoutcust4radI.com/schemas/inoutcust4

Outbound namespace: http://www.inoutcust4radO.com/schemas/inoutcust

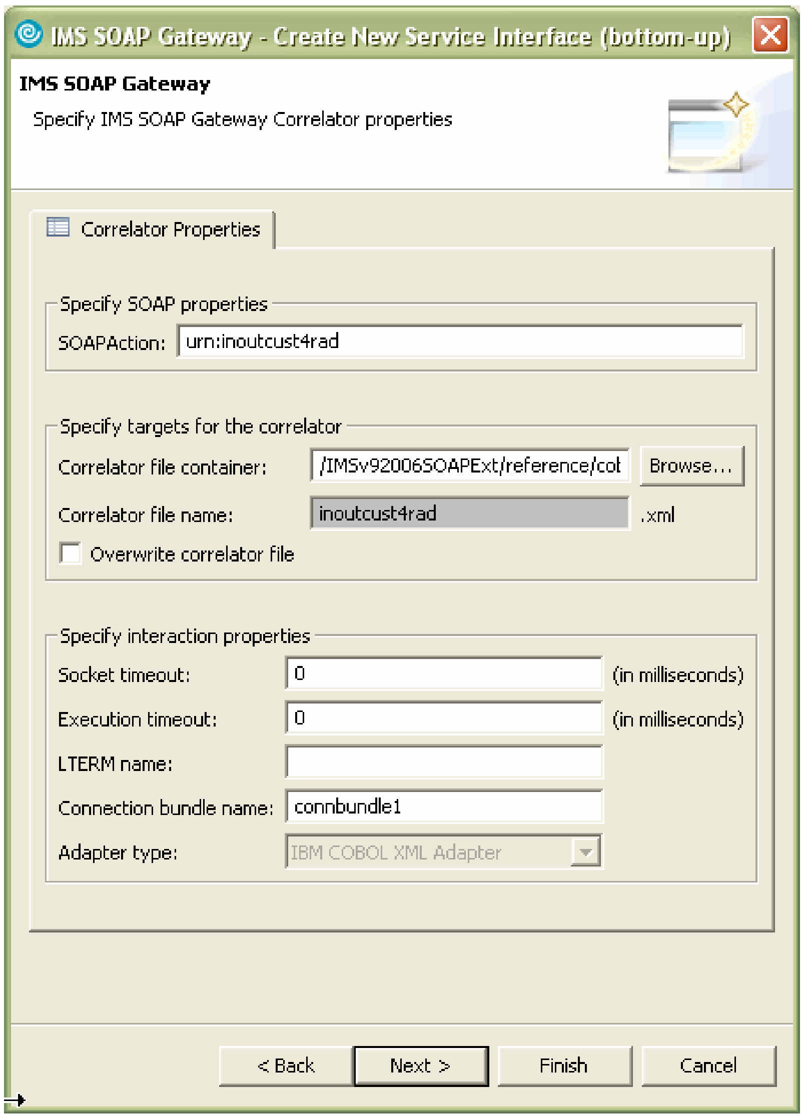
Figure 8-15 WSDL and XSD Options

- In the Endpoint URI field, change the host and port name to the location of IMS SOAP Gateway if needed.

  This URI specifies the address of the Web service. For this example, take the default values as IMS SOAP Gateway is going to run on the same machine as the client program.

Click **Next** to continue.

Specify in the screen of Figure 8-16 on page 215 the IMS SOAP Gateway correlator properties.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

IMS SOAP Gateway

Specify IMS SOAP Gateway Correlator properties

Correlator Properties

Specify SOAP properties

SOAPAction: urn:inoutcust4rad

Specify targets for the correlator

Correlator file container: /IMSv92006SOAPExt/reference/cot Browse...

Correlator file name: inoutcust4rad .xml

☐ Overwrite correlator file

Specify interaction properties

Socket timeout: 0 (in milliseconds)

Execution timeout: 0 (in milliseconds)

LTERM name:

Connection bundle name: connbundle1

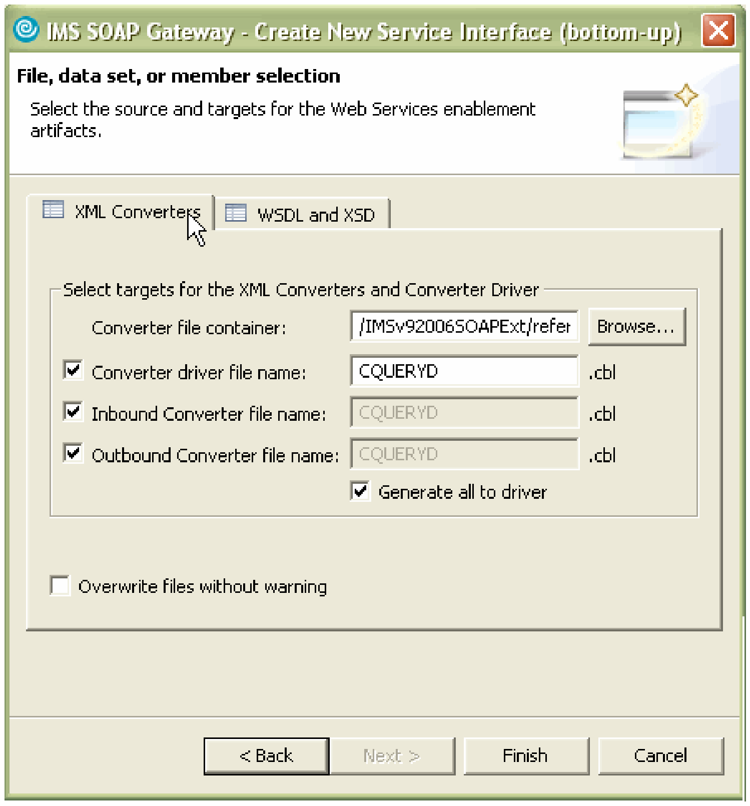
Adapter type: IBM COBOL XML Adapter

< Back | Next > | Finish | Cancel

Figure 8-16 Correlation Properties

- Take the default values as specified in Figure 8-16.
- Click **Next.**

On the screen of Figure 8-17 on page 216 you specify location and names of the Web service artifacts.



IMS SOAP Gateway - Create New Service Interface (bottom-up)

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the XML Converters and Converter Driver

Converter file container: /IMSv92006SOAPExt/refer

[✓] Converter driver file name: CQUERYD .cbl

[✓] Inbound Converter file name: CQUERYD .cbl

[✓] Outbound Converter file name: CQUERYD .cbl

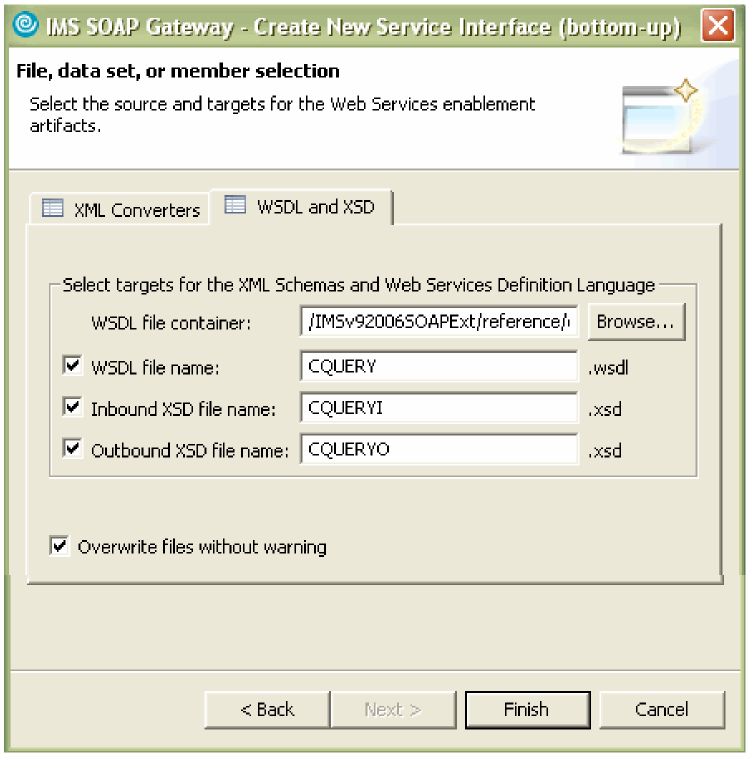
[✓] Generate all to driver

[ ] Overwrite files without warning

Figure 8-17 XML converter names

- Set the value as indicated for COBOL converters and driver.
- Ensure that Generate all to driver is selected.

Select the WSDL and XSD tab as in Figure 8-18 on page 217.


IMS SOAP Gateway - Create New Service Interface (bottom-up)

File, data set, or member selection

Select the source and targets for the Web Services enablement artifacts.

Select targets for the XML Schemas and Web Services Definition Language

WSDL file container:	/IMSv92006SOAPExt/reference/	Browse...
[✓] WSDL file name:	CQUERY	.wsdl
[✓] Inbound XSD file name:	CQUERYI	.xsd
[✓] Outbound XSD file name:	CQUERYO	.xsd

[✓] Overwrite files without warning

Figure 8-18 Member selection

- Set the names as shown in Figure 8-18.
- Ensure that WSDL file name is selected.
- Optionally, select the inbound and outbound XSD files to be generated. These files are not required by the IMS SOAP Gateway.

Click **Finish** and the generation of the required objects for the SOAPGateway is concluded.

Figure 8-19 on page 218 shows in the RAD project the files, artifacts which have been generated for the SOAP Gateway of this application.

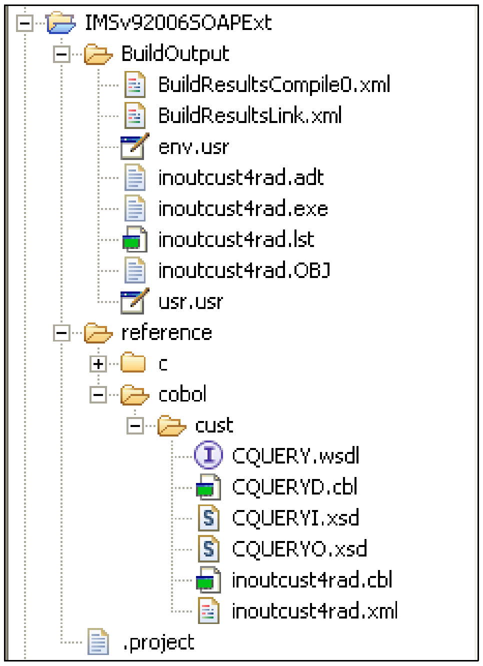

IMSv92006SOAPExt
BuildOutput
BuildResultsCompile0.xml
BuildResultsLink.xml
env.usr
inoutcust4rad.adt
inoutcust4rad.exe
inoutcust4rad.lst
inoutcust4rad.OBJ
usr.usr
reference
c
cobol
cust
CQUERY.wsdl
CQUERYD.cbl
CQUERYI.xsd
CQUERYO.xsd
inoutcust4rad.cbl
inoutcust4rad.xml
.project

Figure 8-19 Extended SOAP gateway Generated files

- CQUERYD.cbl – the COBOL XML Converter driver file
- CQUERY.wsdl - WSDL file
- inoutcust4rad.xml: the correlation file
- CQUERYI.xsd and CQUERYO.xsd - Inbound and outbound XSD files (optional)

The COBOL XML Converter driver program file (CQUERYD.cbl) has to be deployed to IMS Connect as described in 8.3.4, “Deploying to IMS Connect” on page 219.

You need to store the files in the appropriate IMS SOAP Gateway WSDL and XML directories before you start the deployment steps. You can use the RAD File export function to store the files as follows:

- Select the WSDL file (CQUERY.wsdl).
- From the top menu bar, select File → Export. The Export dialog box appears.
- Select File System and click **Next**.
- Browse to the IMS SOAP Gateway WSDL directory <installation_directory>\server\webapps\imssoap\wsdl.

  For example: c:\Program Files\IBM\IMS SOAP Gateway\server\webapps\imssoap\wsdl.
- Select the XML file (inoutcust4rad.xml). From the top menu bar, select **File** → **Export**. The Export dialog box appears.
- Select File System and click **Next.**
- Browse to the IMS SOAP Gateway XML directory <installation_directory>\server\webapps\imssoap\xml.

  For example: c:\Program Files\IBM\IMS SOAP Gateway\server\webapps\imssoap\xml

You can store the WSDL anywhere, but you must then specify the entire path when you are deploying the Web service.

The further action with the WSDL file and the correlator file for deployment is described in 8.3.5, “Deploying to IMS SOAP Gateway” on page 219.

8.3.4 Deploying to IMS Connect

1. Upload the COBOL XML Converter driver file (**CQUERYD.cbl**) generated by WebSphere Developer for zSeries using ASCII file transfer mode as a member in a newly allocated data set on the mainframe host. For example, transfer the file CQUERYD.cbl as member CQUERYD into a data set called *DRIVERS***.IMSXML.SOURCE**.
2. Compile the member CQUERYD from the source data set DRIVERS.IMSXML.SOURCE and linkedit it to a new data set that contains the object members (*DRIVERS*.IMSXML.OBJECT.
3. Update the IMS Connect startup JCL by concatenating the object data set *DRIVERS*.IMSXML.OBJECT to the STEPLIB DD statement. Another option is to linkedit the COBOL XML Converter driver directly into the IMS Connect XML Adapter reslib so that the additional object data set is not needed.

In this example, instead of compiling the COBOL XML Converter driver member into a new data set, we compile and linkedit the driver into the *IMS Connect XML Adapter reslib*. The JCL that compiles and link edits the driver is shown in Example 8-3. It assumes the IMS Connect XML Adapter reslib is in the ‘prefix1.HWS9A00.SHWSRESL data set:

Example 8-3 JCL for compiling COBOL XML converter

```
//userID JOB LINK,MSGLEVEL=1,REGION=640K,CLASS=G,MSGCLASS=H,
// NOTIFY=&SYSUID
//ORDER JCLLIB ORDER=IGY.SIGYPROC
//COMPILE EXEC IGYWCL,LNGPRFX=IGY,PARM.COBOL=LIST,
// PARM.LKED='LET,LIST,MAP,RENT,REUS,AMODE(31)'
//COBOL.SYSLIB DD DSN=CEE.SCEESAMP,DISP=SHR
// DD DSN=prefix1.HMK9A00.SHWSRESL,DISP=SHR
//COBOL.SYSIN DD DISP=SHR,
// DSN=DRIVERS.IMSXML.SOURCE(CQUERYD)
//LKED.SYSLMOD DD DSN=prefix1.HMK9A00.SHWSRESL,DISP=SHR
//*
//LKED.SYSIN DD *
 ENTRY CQUERYD
 ALIAS CQUERYXX
 NAME CQUERYD(R)
/*
```

8.3.5 Deploying to IMS SOAP Gateway

The last deployment step helps you to set up properties and create runtime code that will be used by IMS SOAP Gateway to make your IMS application accessible as Web services.

IMS SOAP Gateway provides you a deployment utility can help you to perform the following tasks:

- Easy end-to-end deployment
- Setup/modify connection/correlation information (for example, hostname/port, trancode, timeout)
- Generates runtime code from WSDL that will be used by IMS SOAP Gateway to enable an IMS application as a Web service

The deployment utility can step you through these tasks or you can perform each task individually either through the interactive mode or the command-line interface.

The following steps show you how to use the IMS SOAP Gateway deployment utility interactive mode to deploy to IMS SOAP Gateway and make the Customer query application as a Web service:

1. Before you start, make sure that the WSDL and correlator file generated by WDz are already stored in the appropriate IMS SOAP Gateway WSDL and XML directories as described in
   - The WSDL file (CQUERY.wsdl) should be stored in the IMS SOAP Gateway WSDL directory: installation_directory\server\webapps\imssoap\wsdl.
   - The correlator file (inoutcust4rad.xml) should be stored in the IMS SOAP Gateway XML directory: installation_directory\server\webapps\imssoap\xml.
2. Start → All Programs → IBM IMS SOAP Gateway Version 9 → Deployment Utility
   - The deployment utility interaction mode will start and the main menu of Example 8-4 is displayed.
   - The main menu lists the different tasks you can perform with the deployment utility.

Example 8-4 Main menu for deployment utility

```
*****************************************************************
* *
* Welcome to the IMS SOAP Gateway Deployment Utility *
* *
*****************************************************************
The IMS SOAP Gateway Deployment Utility provides an interactive
user interface with tasks to enable and maintain your IMS
applications as a Web service.
To get Help for a particular task, go to the IMS SOAP Gateway
documentation.
To return to the main menu, type "cancel" at any point.
================================================================
Enable your IMS application as a Web service:
Task 1: Enable your IMS application as a Web service from start to finish

Administrative tasks:
Task 2: Start IMS SOAP Gateway
Task 3: Stop IMS SOAP Gateway
Task 4: Update IMS SOAP Gateway properties
Task 5: Create, Update or View correlator properties for Web service
Task 6: Create, Update, Delete or View connection bundle
Task 7: Deploy the WSDL file
Task 8: Generate Java client code
Task 9: Undeploy Web service
Task 10: Exit deployment utility
================================================================
> Enter your selection here:
```

3. Your IMS SOAP Gateway **must be started** before you can deploy the Web service. To start IMS SOAP Gateway, you can use task 2 of the deployment tool by typing **2** and hit enter. In a Windows command window you see the output reported as in Example 8-5 on page 221.

Example 8-5 Start of Gateway server

```
[2006-04-13 01:07:03,233] main Catalina
 INFO : Initialization processed in 2361 ms
[2006-04-13 01:07:07,084] main Catalina
 INFO : Server startup in 3851 ms
[2006-04-13 01:07:07,084] main Catalina
 INFO : IMS SOAP Gateway server is now up and running.
```

4. To deploy the files and setup the properties to make your IMS application accessible as a Web service. You will use **Task 1**: Enable your IMS application as a Web service from start to finish to complete the deployment task.
   – To start task 1, type 1 and hit **enter**
5. The deployment utility now guides you through the following steps to complete the deployment task:
   – Specifying which WSDL file to use for the deployment.
   – Specifying the connection and security information for the Web service to create a connection bundle.
   – Specifying the interaction properties of the Web service to create a correlator file.
   – Deploying the Web service to IMS SOAP Gateway.
   – (Optional) Creating the Java client proxy code.

Example 8-6 shows the items you will be prompted for in input (the sample values are highlighted).

Example 8-6 Prompted input for deployment

```
> Enter your selection here: 1
Task 1 guides you through a series of steps to publish your IMS
application as a Web service.
Note: You must have IMS SOAP Gateway started and a WSDL file
created for your IMS application before you enter this task.
Step a: Provide the WSDL file for deployment
--------------------------------------------
To enable your IMS application as a Web service, you need a WSDL
file. You can create a WSDL file for your IMS application
manually or use a development tool.
At the prompt, specify the name and location of the WSDL file you
have created (e.g. c:\MyWSDLFile.wsdl).
If you had deployed the WSDL before or you have copied the WSDL
into the IMS SOAP Gateway wsdl directory, provide just the name
for the WSDL file (e.g. MyWSDL.wsdl), the full path is NOT
required.
> Provide the name and location of the WSDL file: CQUERY.wsdl

Step b: Provide Connection properties for connecting to IMS
------------------------------------------------------------
This step allows you to create or reuse a connection bundle that
consists of properties for connecting to IMS.
> Do you want to view all existing connection bundles? [Y/N]: N
> Do you want to create a new connection bundle? [Y/N]: y
- Provide the following connection bundle properties.
> Provide a name for the connection bundle: connbundlec
Warning: Connection bundle xml file is empty
```

```
> Provide IMS host name or IP address: WTSC63
> Provide IMS port number(default: 9999): 6001
> Provide IMS datastore name: IMSH
> Provide IMS userid (optional): PAOLOR7
The password will not appear.
> Provide IMS password (optional):
> Provide IMS group name (optional):
- The connection bundle 'connbundle1' has been created.
Step c: Provide Interaction (Correlator) properties for Web service
-----------------------------------------------------------------
This step allows you to create or reuse a correlator file that
consists of properties for interacting with the IMS application
for the Web service.
If you use the IBM WebSphere Developer for zSeries (WDz) tooling,
the correlator file is generated for you as part of the process.
Otherwise, you will have to create the correlator file.
> Do you want to use an existing Correlator file? [Y/N]: Y
At the prompt, specify the name and location of the correlator file.
If you had created the correlator file before or you have copied
the correlator file into the IMS SOAP Gateway xml directory,
provide just the name of the correlator file (e.g. MyCorr.xml).
The full path is NOT required.
Note: If you use a correlator file generated by WDz V6.0.1, you
may need to update the IMS transaction code value in the
correlator file after you deploy the Web service.
> Provide the name and full path to the Correlator file: inoutcust4rad.xml
> Do you want to view the correlator file ? [Y/N]: Y
The Correlator file values are :
Soap Action : inout4rad
Program Name : CQUERYD
Socket Timeout : 0
Execution Timeout : 0
Lterm Name :
Adapter type : IBM COBOL XML Adapter
Connection Bundle Name : connbundle1
Step d: Deploy the WSDL to IMS SOAP Gateway
--------------------------------------------
This step deploys the WSDL file and make your IMS application
accessible as a Web service.
Note: Ensure that IMS SOAP gateway is already started.
> Do you want to deploy the WSDL file to IMS SOAP gateway?[Y/N]:Y
Processing file C:\Program Files\IBM\IMS SOAP Gateway
V9.1\server\webapps\imssoap\temp\server\deploy.wsdd
- WSDL was successfully deployed.
```

The deployment is now complete.

In the following step the system asks whether you want Java client code to be generated. See Example 8-7. If you want to create a java client to invoke the IMS Customer Query Web service, this step can be useful. This java client code is proxy code that you can utilize in your client application to wrap the transaction data in a SOAP message and create a connection to send and receive SOAP messages with IMS SOAP Gateway.

Example 8-7 Generating Java client code

```
Step e: Generate java client code (optional)
```

```
--------------------------------------------
This step generates proxy code that you can use to build a Java application to
invoke the Web service.
> Do you want to generate java client code (optional)? [Y/N]: Y
> Please enter the location to output client files (Default : C:\Program
Files\IBM\IMS SOAP Gateway\server\webapps\imssoap\temp\client) :

Client code generation in directory "C:\Program Files\IBM\IMS
SOAP\Gateway\server\webapps\imssoap\temp\client" is complete.

Enable your IMS application as a Web Service :
Task 1: Enable your IMS application as a Web Service from start to finish

Administrative tasks :
Task 2: Start IMS SOAP Gateway
Task 3: Stop IMS SOAP Gateway
Task 4: Update IMS SOAP Gateway properties
Task 5: Create, Update or View correlator properties for Web Service
Task 6: Create, Update, Delete or View connection bundle
Task 7: Deploy the WSDL file
Task 8: Generate Java client code
Task 9: Undeploy Web Service
Task 10: Exit deployment utility
=================================================================
> Enter your selection here: 10
```

Restart IMS SOAP Gateway.

Stop IMS SOAP Gateway. You can stop IMS SOAP Gateway by using **option 2** from the deployment utility menu

Start IMS SOAP Gateway. You can start IMS SOAP Gateway by using **option 1**, from the deployment utility menu.

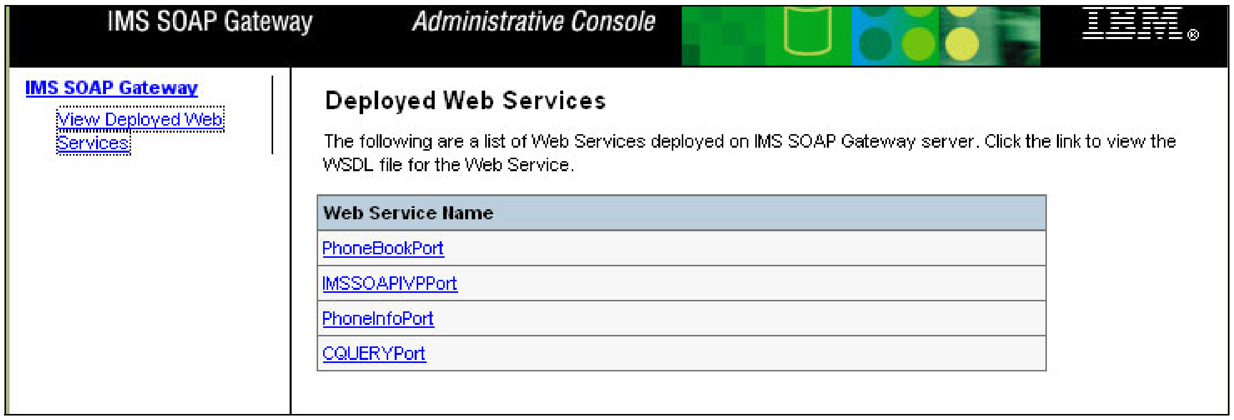
Note: In general, you do not need to restart IMS SOAP Gateway after you deploy a new Web Service. You only need to restart IMS SOAP Gateway if you create a new connection bundle during the deployment process.

Your IMS applications are ready to be accessible by a client as a Web Service. To verify the deployment has completed successfully, do the following:

- Start the IMS SOAP Gateway Administrative Console by

  Start → Programs → IBM IMS SOAP Gateway9.1 → Administrative Console.

  It will open a Web browser and display the Administrative Console.
- Click **View deployed Web Services** link. You should see the **CQUERYPort** listed in Figure 8-20 on page 224.


IMS SOAP Gateway Administrative Console

IMS SOAP Gateway

View Deployed Web Services

Deployed Web Services

The following are a list of Web Services deployed on IMS SOAP Gateway server. Click the link to view the WSDL file for the Web Service.

Web Service Name
PhoneBookPort
IMSSOAPIVPPort
PhoneInfoPort
CQUERYPort

Figure 8-20 SOAP Gateway Administrative Console

You could also look at the details of the WSDL file by clicking on it.

8.3.6 Accessing from a SOAP client

The HTTP Service EndPoint for the Web services is located in Windows. This information is also available in the WSDL file in the Service Endpoint section. See Figure 8-21.

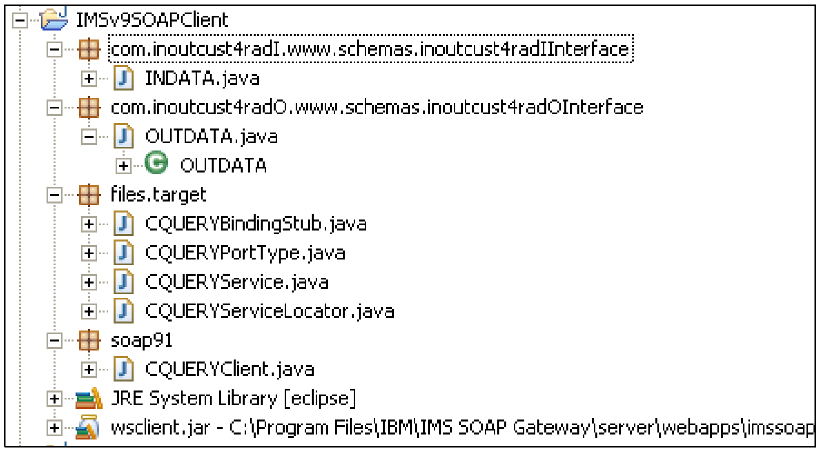

```
IMSv9SOAPClient
  com.inoutcust4radI.www.schemas.inoutcust4radIInterface
    INDATA.java
  com.inoutcust4radO.www.schemas.inoutcust4radOInterface
    OUTDATA.java
      OUTDATA
  files.target
    CQUERYBindingStub.java
    CQUERYPortType.java
    CQUERYService.java
    CQUERYServiceLocator.java
  soap91
    CQUERYClient.java
  JRE System Library [eclipse]
  wsclient.jar - C:\Program Files\IBM\IMS SOAP Gateway\server\webapps\imssoap
```

Figure 8-21 IMSClient project for SOAP Gateway client

To use the SOAP service we have to use the previously generated client code as a proxy. We have created a new Java project **IMSv9SOAPClient** in RAD and imported the client proxy code (classes and jar) in it.

In the package **soap91,** you find the java program CQUERYClient.java, which can interact via the GateWay with IMS. The program is listed in Example 8-8.

Example 8-8 Java CQuery client program

```
/*
 * Created on July 17, 2006
 */
package soap91;

import files.target.*;
import com.inoutcust4radI.www.schemas.inoutcust4radIInterface.INDATA;
import com.inoutcust4radO.www.schemas.inoutcust4radOInterface.OUTDATA;

public class CQUERYClient {
   public static void main(String[] args) {
      try {
         //  Populate the input data
         INDATA input = new INDATA();
         input.setIn_ll((short) 32);
         input.setIn_zz((short) 0);
         input.setIn_trcd("CQUERY");
         input.setCustomernr("006668");
         input.setTracex("Y");

         //  Invoke the service
         CQUERYService cqueryservice = new CQUERYServiceLocator();
         CQUERYPortType cquery = cqueryservice.getCQUERYPort();
         OUTDATA output = cquery.CQUERYOperation(input);
         //  Display the output data
         System.out.println("Customer: " + output.getCustomernr());
         System.out.println("Name: " + output.getSalutat() + " "
               + output.getFirstnme() + " " + output.getMi() + " "
               + output.getLastname());
         System.out.println("Address: " + output.getAddress1() + " "
               + output.getAddress2());
         System.out.println("City " + output.getZipcd() + " "
               + output.getCity());
         System.out.println("Country/State: " + output.getState());
         System.out.println("Phone: " + output.getPhone());
         System.out.println("Fax: " + output.getFax());
         System.out.println("EmlAddr: " + output.getEmladdr());
         System.out.println("Message: " + output.getMessagex());
      } catch (Exception exception) {
         exception.printStackTrace();
      }
   }
}
```

8.3.7 Preparation at the z/OS host

In this section we describe the installation and customization processes of IMS Connect XML Adapter. It requires IMS Version 9 with Integrated IMS Connect.

Note: The libraries that come with the IMS Connect XML Adapter should be installed on *different* data sets from your IMS Connect Version 9.1 installation. You should *never* replace your current IMS Connect Version 9 libraries with any of the IMS Connect XML Adapter libraries.

This IMS Connect XML Adapter distribution is intended for testing of the new XML Adapter function only. It is not for production use.

Installing IMS Connect XML Adapter

- Download IMS Connect XML Adapter (hwsadapterbeta.zip) from the IMS SOAP Gateway with XML Adapter Web site to your workstation.

- Unzip the hwsadapterbeta.zip file and it will extract one file called XMLADPT.
- Upload the XMLADPT file from your workstation to your host environment as follows:
  - Allocate a data set (for example, named **'userid.XMLADPT** with the following attributes:

```
1st extent cylinders: 15
Secondary cylinders : 1
Record format . . . : FB
Record length . . . : 1024
Block size . . . . .: 6144
```

- Use FTP to upload the XMLADPT file using BINARY transfer mode:
- Unpack the XMLADPT file back into data sets by using the JCL in Example 8-9.

Example 8-9 Populating the IMS Connect XML adapter data sets

```
//RESTORE JOB <job parameters>
//*
//***********************************************************/
//* JOB NAME: RESTORE */
//* */
//* LICENSED MATERIALS - PROPERTY OF IBM */
//* 5655-J38 (C) COPYRIGHT IBM CORP 1974,2003 */
//* ALL RIGHTS RESERVED. */
//* US GOVERNMENT USERS RESTRICTED RIGHTS - */
//* USE, DUPLICATION OR DISCLOSURE RESTRICTED */
//* BY GSA ADP SCHEDULE CONTRACT WITH IBM CORP. */
//* */
//* DESCRIPTION: THIS JCL WILL UNTERSE, ALLOCATE AND POPULATE */
//* THE DATASETS FOR THE IMS CONNECT XML ADAPTER */
//* CAUTION: THIS IS NEITHER A JCL PROCEDURE NOR */
//* A COMPLETE JOB. */
//* YOU WILL HAVE TO MAKE MODIFICATIONS BEFORE */
//* SUBMITTING THIS JOB. */
//* */
//* NOTES: */
//* 1) CHANGE THE JOB CARD TO MEET YOUR SYSTEM'S REQUIREMENTS. */
//* 2) CHANGE userid TO THE USERID FROM STEP 3 ABOVE. */
//* 3) CHANGE outdsn1 TO NAME OF THE OUTPUT DATASET OF UNTERSE.*/
//* 4) CHANGE volid TO THE VOLSER ID OF THE DASD THAT WILL */
//* CONTAIN THE RESTORED DATASETS. */
//* 5) CHANGE prefix1 TO THE PREFIX FOR THE RESTORED DATASETS. */
//* */
//* NOTES: */
//* Changes to this job are Case Sensitive. */
//* The FTP commands MUST be entered in lower case as shown. */
//* THE FOLLOWING ARE SUCCESSFUL RETURN CODES FOR THE */
//* JOB STEPS: */
//* UNTERSE RC=0 */
//* RESTR RC=0 */
//* */
//***********************************************************/
//UNTERSE EXEC PGM=TRSMAIN,PARM='UNPACK'
//SYSPRINT DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=12901,RECFM=FBA)
//INFILE DD DISP=SHR,DSNAME=userid.XMLADPT
//OUTFILE DD DISP=(,CATLG,DELETE),
// DSN=outdsn1,
// SPACE=(CYL,(15,2),RLSE),
// UNIT=SYSDA
//RESTR EXEC PGM=ADRDSSU,REGION=2048K
//SYSPRINT DD SYSOUT=*
//INDD1 DD DSN=outdsn1,
// DISP=(OLD,KEEP,KEEP)
//DASD1 DD UNIT=3390,VOL=(PRIVATE,SER=volid),DISP=OLD
//SYSIN DD *
   RESTORE DATASET( -
   INCLUDE( -
   ** -
   )) -
   INDDNAME(INDD1) -
   OUTDDNAME(DASD1) -
   NULLSTORCLAS -
   CANCELERROR -
   CATALOG -
   SHR -
   IMPORT -
   TGTALLOC(SOURCE) -
   RENAMEUNCONDITIONAL(prefix1) -
   WAIT(2,2)
```

You now have the following data sets:

- prefix1.HMK9A00.AHWSLOAD which contains the load library
- prefix1.HMK9A00.AHWSMAC which contains the macro library
- prefix1.HMK9A00.AHWSSMPL which contains the samples library
- prefix1.HMK9A00.SHWSRESL which contains the reslib (except the samples sources)
- prefix1.HMK9A00.AHWSJCL which contains sample JCL for your reference

IMS Connect with the XML Adapter has now been installed, and is ready for customization.

Configuring IMS Connect XML Adapter

1. Compile and linkedit the following samples from the samples library
   - prefix1.HMK9A00.AHWSSMPL into your reslib prefix1.HMK9A00.SHWSRESL
   - HWSUINIT (required by IMS Connect)
   - HWSJAVA0 (required by IMS Connect)
   - HWSSMPL1 (if being used)
   - HWSSMPL0 (if being used)
   - HWSYDRU (if being used)
   - HWSTECL0 (if being used)
2. Create a IMS Connect configuration file as described in the IMS Connect Version 9 documentation.
3. Add a new IMS Connect configuration file statement called ADAPTER

   ```
   ADAPTER (XML=Y)
   ```

   where XML=Y means Adapter supported and XML=N means Adapter *not* supported.
4. Add the User Exit Routine HWSSOAP1 to the IMS Connect configuration file for the TCPIP statement at EXIT= :

   ```
   TCPIP=(HOSTNAME=...,EXIT=(HWSSMPL0,HWSSMPL1,HWSSOAP1),...)
   ```

5. Build the IMS Connect Adapter list by creating a PROCLIB member named HWSEXIT0 with the following content:

   ```
   EXITDEF(TYPE=XMLADAP,EXITS=(HWSXMLC0),ABLIM=8,COMP=HWS)
   ```

6. Update the BPE CONFIG File by adding an EXITMBR statement

   ```
   #
   #DEFINITIONS FOR HWS EXITS
   #
   XITMBR=(HWSEXIT0,HWS) /* HWS USER EXIT DEF */
   ```

7. Create a new procedure for this IMS Connect XML Adapter. The new procedure should be as defined in the IMS Connect Version 9.1 documentation with prefix1.HMK9A00.SHWSRESL being the current reslib and with the IMS Connect Version 9 Reslib concatenated behind the prefix1.HMK9A00.SHWSRESL reslib.

   The IMS Connect Version 9 is required for the BPE environment when the IMS Connect XML Adapter is brought up. The HWS modules in IMS Connect Version 9 will not be loaded.

8. It will also be necessary to APF authorize your Prefix1.HMK9A00.SHWSRESL library.

In Example 8-10 you find the new HSW config member with the ADAPTER statement

Example 8-10 New HWS Configuration member

```
HWS (ID=IMSHCONN,RACF=N,XIBAREA=20,RRS=Y)
TCPIP (HOSTNAME=TCPIP,PORTID=(6001,LOCAL),MAXSOC=2000,TIMEOUT=3000,
EXIT=(HWSSOAP1))
ADAPTER (XML=Y)
DATASTORE (ID=IMSH,GROUP=IMSHEXCF,MEMBER=HWS910H,TMEMBER=SCSIMS9H)
```

This concludes the example with the Web Services Soap Gateway.

8.4 IMS Services through WebSphere

WebSphere Application Server (WAS) is a new type of *transactional environment*. A further description of WebSphere Application Server can be found in *IBM WebSphere Application Server for z/OS Version 6.0.1: Developing and Deploying Applications*, SA22-7959.

WebSphere Application Server is the container for several types of J2EE artifacts (modules), like Servlets and EJBs. Through these artifacts processing can be dispatched in the WebSphere Application Server via multiple transports from several types of clients, The transports include HTTP[S], SOAP over HTTP[S], IIOP[S], JMS, and SOAP over JMS. The J2EE modules in WebSphere Application Server can behave as a service for external clients and can also be a consumer of existing services. Within the SOA architecture, Web Services over SOAP are only one particular form of service. Other services are access to a CICS/IMS transaction through a J2C connector over TCP/IP or by an internal protocol (EXCI for CICS), or access over an adapter to a service located on another platform, it could also be a DB2 access over an EJB. Many other examples can be found.

Web services are using SOAP (Simple Object Access Protocol) messages. In theory these messages can flow over many transports. IBM supports Web services over HTTP and JMS (Java Message Services). In WebSphere Application Server, an access through Web services (SOAP) can be built for many types of components like Plain Old Java Objects (POJOs), Enterprise Java Beans (session).

The build of the required artifacts for a service is supported by tool builders like the **Rational Application Developer (RAD)** and the extension **WebSphere Integration Developer (WID).** The artifacts that are built by this tooling are all in Java. The use of wizards is straight forward. In this chapter we concentrate on the build of services for an existing IMS transaction. Good candidates are all non conversational transactions, using **NO** Scratch Pad Area (SPA) in their definition.

For IMS/CICS transactions, we always have to pass through a J2C connector bean. Starting from an existing implemented transaction, the *end to end* build requires the following information during the use of the wizards:

- Layout of the input message (Cobol copybook, c include)
- Layout of output messages (Cobol copybook, c include)
- Property values for the **connectionSpec**(ification)
  - IP address of IMS Connect
  - Port number of IMS Connect
  - Datastore
  - Security information

  This information can be made available in two ways:

  - **managed:** a ConnectionFactory definition has been made available in the WebSphere Application Server or in the WebSphere Test Environment (WTE in RAD). This definition can be addressed by a lookup.
  - **non-managed:** the data is passed at runtime.
- Data for the interactionSpec(ification)
  - commit protocol (send and commit, commit and send)\synclevel (0, 1, 3).

The combination of RAD and WebSphere Application Server provides almost everything that is needed to build service solutions for IMS transactions. The data that we use in our example is shown in Table 8-1.

Table 8-1 Our input for the Wizard

InputMessage	C include	inoutcust4rad.h(INDATA)
OutputMessage	C include	inoutcust4rad.h(OUTDATA)
CommectionSpec	IpAddres IpPort datastore userId pwd dedicated SSL	WTSC63 3001 IMSH PAOLOR7 ******** false NO
InteractionSpec	defaults	SYNC_SEND_RECEIVE SEND_THEN_COMMIT Request_type_transaction

The J2C implementation for IMS also requires the installation of the Resource Adapter for IMS. The latest distribution is v9.1.0.2.2. The distribution is available as a RAR file in a ZIP file for window platforms and a TAR file for UNIX (including z/OS) platform. The installation of a RAR is almost the same for all resource types and always explained together with the distribution. When testing from the RAD development then this resource adapter must also be installed in this tool.

For the IMSCON RAR, you find information and installation instructions at the URL:

http://www.ibm.com/software/data/db2imstools/imstools/imsjavcon.html

RAD and WID support the building of J2C connectors, under the condition that the J2C plug-in has been installed in the development tool. This can be achieved by updating options from the tool maintenance site. The plugin/wizard allows the build of J2C connectors for EIS like IMS, CICS and others. This J2C connector can then be used in several ways, via a simple command bean, a session EJB or Web Services.

8.4.1 HTTP Web Service built with RAD

An access facility to IMS/DC is always based on a J2C connector layer (layer 2, using the databinding). In a service-oriented architecture, using Web Services over SOAP is an option.

Several accesses are available. This in shown in Figure 8-22. On each of the indicated access points, indicated by an arrow, we have a J2C service available. Client code has to be adapted to the particular access type. In all solutions the layers 1 (databinding) and 2 (J2C bean) are common.

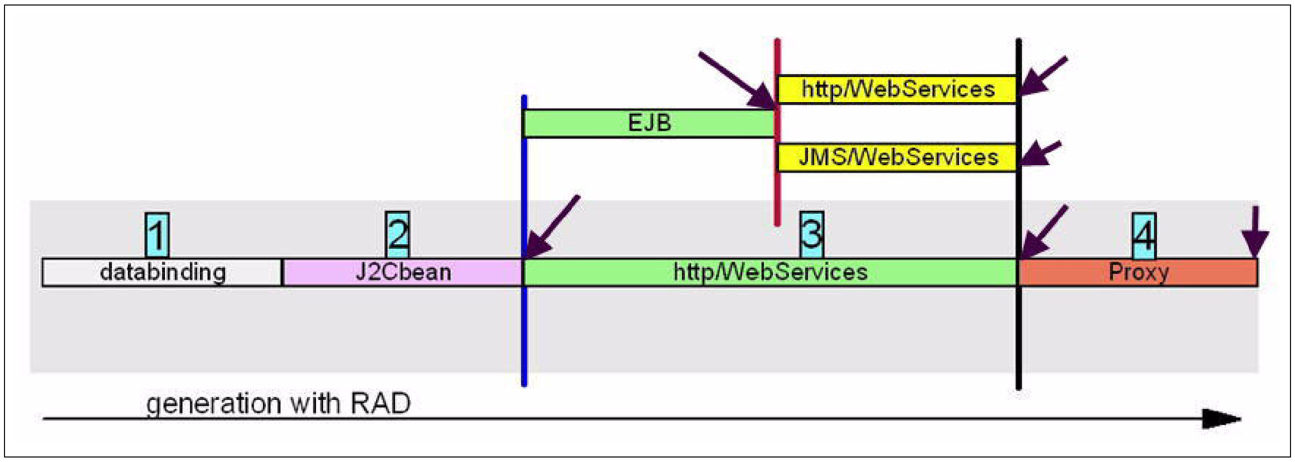


Figure 8-22 J2C services with RAD

Your choice of solution depends on the particular need of your environment.

In our example we selected a Web Service solution over HTTP. Everything including optional clientproxy code is prepared with RAD wizards. The path is shown in Figure 8-22.

A Web service solution is built in four phases

1. DataBinding
2. J2C Bean generation
3. Web services enablement
4. (optional) Client stubs, helpers, proxy

The final solution is shown in Figure 8-23 on page 231. It shows also the names of the projects used in the RAD workbench.

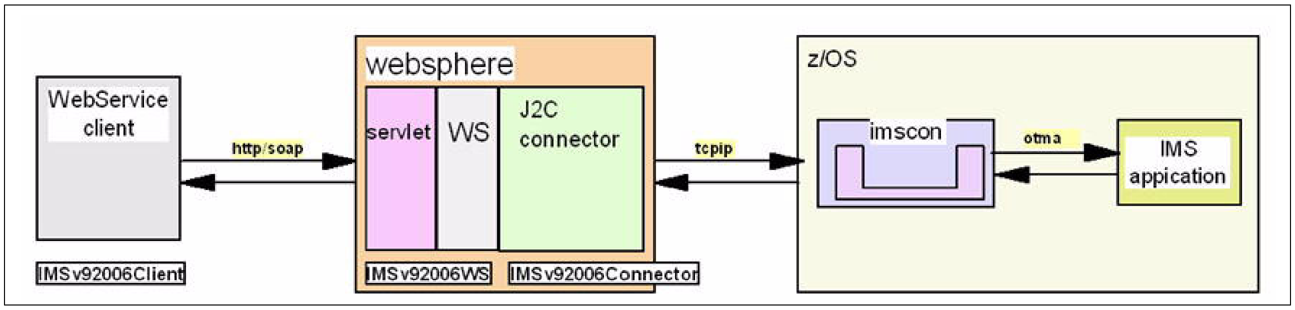


Figure 8-23 Web services solution built with J2C wizard

The generation and preparation is done as much as possible with wizards available in Rational Application Developer v6 (RAD). Only for the Web service client it is required to do some basic Java coding.

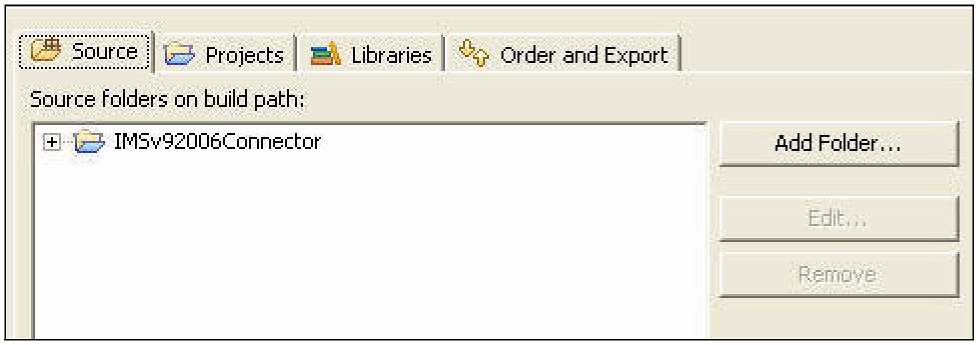
Preparation of the Connector project

In this Java project in the RAD workbench, we start by building the connector elements, but first we have to prepare it. Assume that RAD is started on Windows, and that currently we are in a J2EE perspective.

- In the left upper corner of the RAD window, in the menu bar, click **File** → **New** → **Project.**
- In the **New Project** pop-up, select the wizard for a Java Project, click **Next.**
- Give the new project a name, we suggest **IMSv92006Connector**, click **Next**.

The new project is created now, but some properties have to be adapted to allow a connector to be built in it.

In the new pop-up select the Projects menu option. See Figure 8-24.



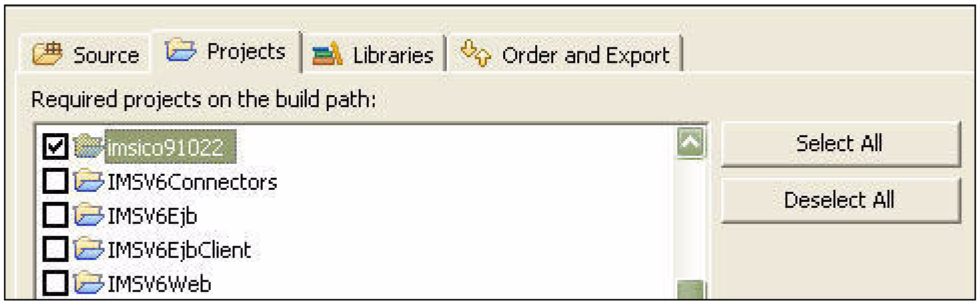
Source folders on build path:

IMSv92006Connector

Figure 8-24 Project references

In the project list, check the Connector project **imsico91022**, which contains the IMSv9 resource adapter previously installed. This is shown in Figure 8-25 on page 232.

Select the **Libraries** option.



Required projects on the build path:

- [✓] imsico91022
- [ ] IMSV6Connectors
- [ ] IMSV6Ejb
- [ ] IMSV6EjbClient
- [ ] IMSV6Web

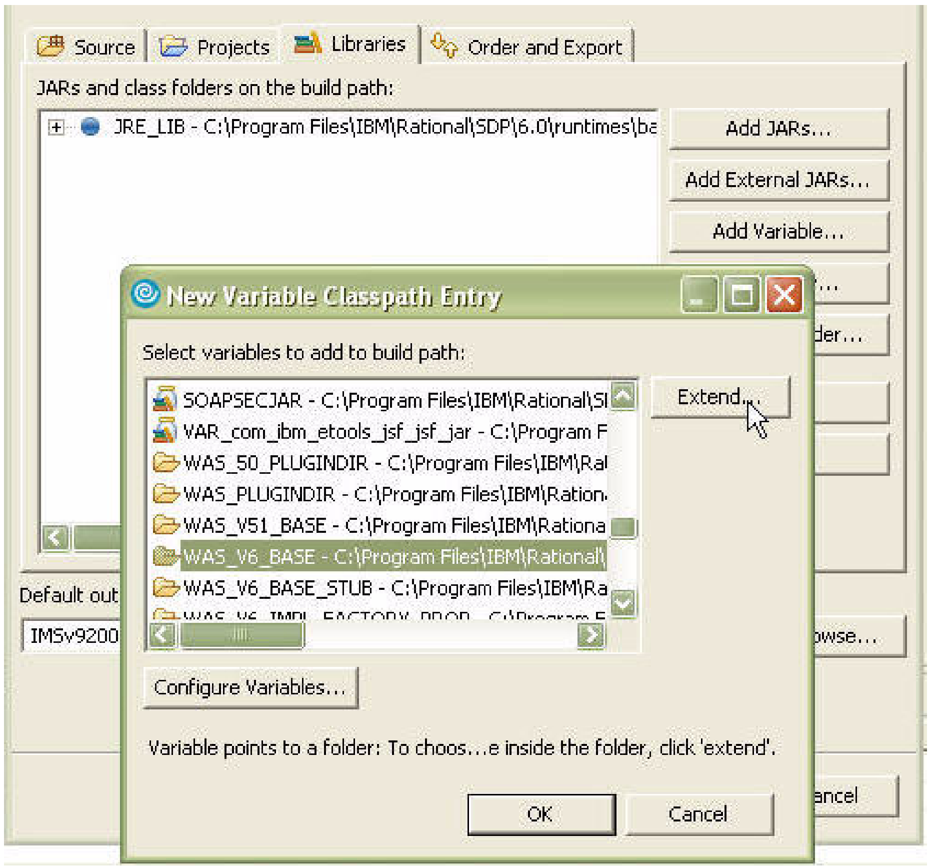
Figure 8-25 Project list

Select the **Libraries** option.

The project needs to have access to additional Java code (JAR files). We add here the J2EE library.

Click **Add Variable**, in the Variable Classpath Entry list that comes up, highlight WAS_V6_BASE, click **Extend** in the extensions navigate to the subdirectory **lib** and expand it.

From the new list select **j2ee.jar,** and click **OK,** as shown in Figure 8-26.



JARs and class folders on the build path:

JRE_LIB - C:\Program Files\IBM\Rational\SDP\6.0\runtimes\ba

Select variables to add to build path:

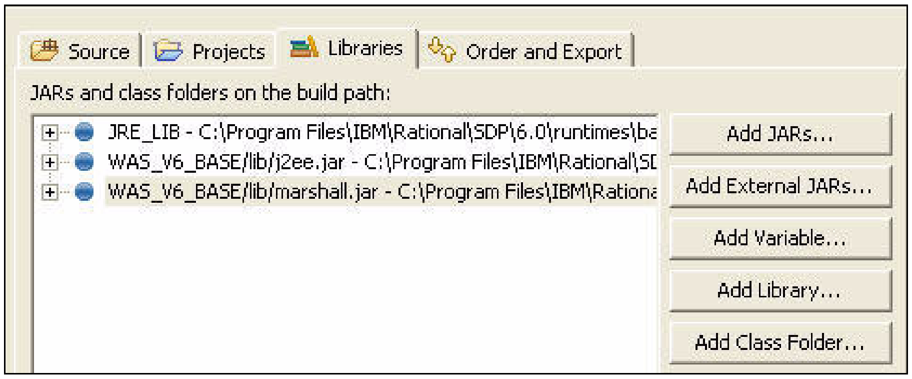
SOAPSECJAR - C:\Program Files\IBM\Rational\S
VAR_com_ibm_etools_jsf_jsf_jar - C:\Program F
WAS_50_PLUGINDIR - C:\Program Files\IBM\Ra
WAS_PLUGINDIR - C:\Program Files\IBM\Ration
WAS_V51_BASE - C:\Program Files\IBM\Rationa
WAS_V6_BASE - C:\Program Files\IBM\Rational\
WAS_V6_BASE_STUB - C:\Program Files\IBM\Ra

Variable points to a folder: To choos...e inside the folder, click 'extend'.

Figure 8-26 Libraries with J2EE jar

This operation has to be repeated for another file marshall.jar.

The pop-up with the Libraries selection is shown in Figure 8-27. Notice the two added libraries.



JARs and class folders on the build path:

```
JRE_LIB - C:\Program Files\IBM\Rational\SDP\6.0\runtimes\ba
WAS_V6_BASE/lib/j2ee.jar - C:\Program Files\IBM\Rational\SD
WAS_V6_BASE/lib/marshall.jar - C:\Program Files\IBM\Rationa
```

Figure 8-27 After adding libraries

Click **Finish.**

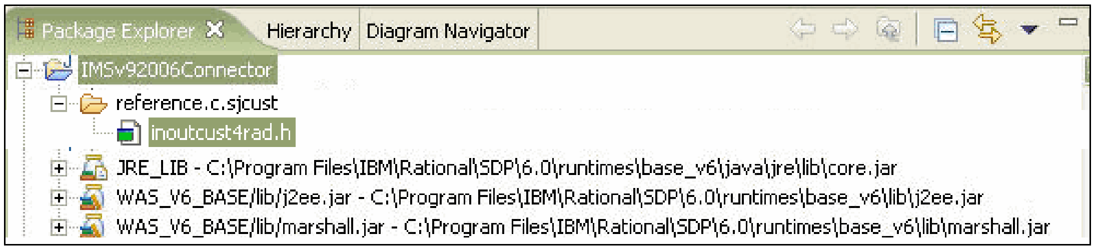
During the databinding phase we intend to use an existing C include with descriptions of the input/output messages for IMS. The cobol/copybook file *inoutcust4rad.h*, has to be copied in the hierarchy of the project. In the project, the file has been placed in folder *reference.c.sjcust*.

We distinguish the input and output section in the H file shown in Figure 8-28.

```
struct INDATA {
   short     llin;
   char      z1;
   char      z2;
   char      tran[8];       // transaction code
   char      custnr[10];    // customernr
   char      trace;         // trace y/n
   };
struct  OUTDATA {
   short     llout;
   char      z1;
   char      z2;
   char      custnr[10];    // customernr
   char      ssn[11];       //
   char      firstnme[20];  // first name
   char      mi[2];         // middle name
   char      lastname[40];  // last name
   char      salutat[10];   // salutation
   char      address1[40];  // address 1
   char      address2[40];  // address 2
   char      city[30];      // city
   char      state[4];      // state/department/kreiz
   char      zipcd[10];     // zip code/ postcode
   char      phone[20];     // phone
   char      fax[20];       // fax
   char      emladdr[40];   // email adddress
   char      message[200];  // message
   };
```

Figure 8-28 h include file inoutcust4rad

Now the project is ready for the next phase. An expansion of the project is shown in Figure 8-29.



```
IMSv92006Connector
  reference.c.sjcust
    inoutcust4rad.h
  JRE_LIB - C:\Program Files\IBM\Rational\SDP\6.0\runtimes\base_v6\java\jre\lib\core.jar
  WAS_V6_BASE/lib/j2ee.jar - C:\Program Files\IBM\Rational\SDP\6.0\runtimes\base_v6\lib\j2ee.jar
  WAS_V6_BASE/lib/marshall.jar - C:\Program Files\IBM\Rational\SDP\6.0\runtimes\base_v6\lib\marshall.jar
```

Figure 8-29 IMSv92006Connector project

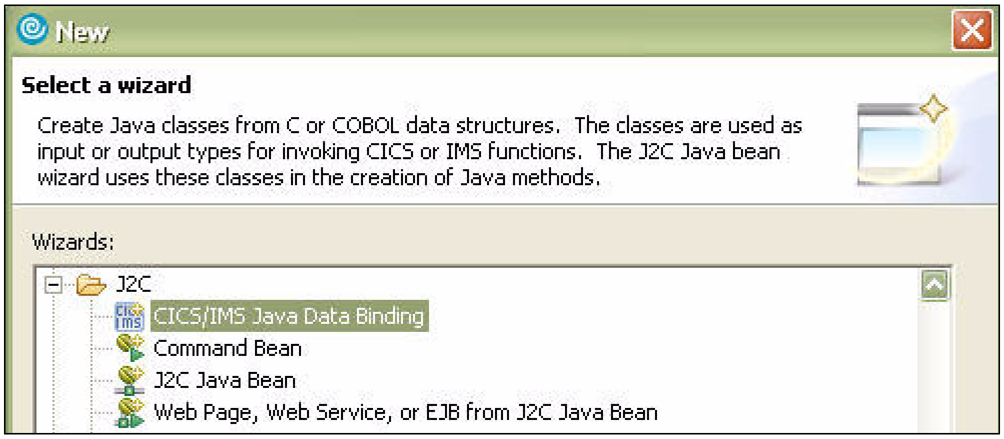
DataBinding

Although the messages exchanged with IMS are basically bytestrings (in EBCDIC) the J2C connector architecture uses as input/output for the CCI (Client Connector Interface), part of Java Connector Architecture (JCA1.5), special wrapping Java Beans (implementing the CCI interface). In those beans the input message is assembled and from here the output fields can be extracted on response. During this phase, the cobol copybooks or c includes can be imported into the wizard, which as a result builds those **databinding java** beans.

Select the previously created project **IMSv92006Connector.**

Right-click **New** → **Other.**

In the **New** pop-up, select in the wizard part for a J2C the option **CICS/IMS Java Data** Binding, as indicated in Figure 8-30.



Select a wizard

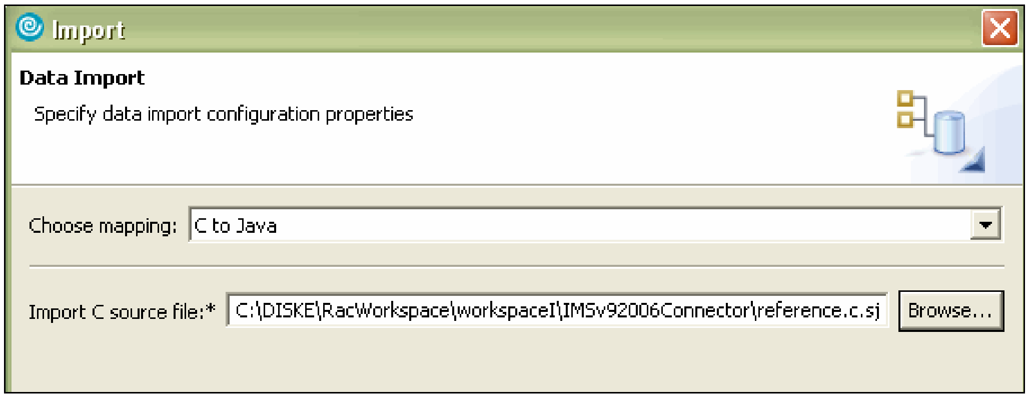
Create Java classes from C or COBOL data structures. The classes are used as input or output types for invoking CICS or IMS functions. The J2C Java bean wizard uses these classes in the creation of Java methods.

Wizards:
J2C
CICS/IMS Java Data Binding
Command Bean
J2C Java Bean
Web Page, Web Service, or EJB from J2C Java Bean

Figure 8-30 DataBinding phase selection

Click **Next.**

The following Data import screen requests a cobol/copybook or a c/include with the message mapping. Select the **Cobol to Java** option as shown, and with the Browse facility, navigate to file **Ex01.cbl** in this project, → **Open the file**, and the result is shown in Figure 8-31 on page 235.


Import

Data Import

Specify data import configuration properties

Choose mapping: C to Java

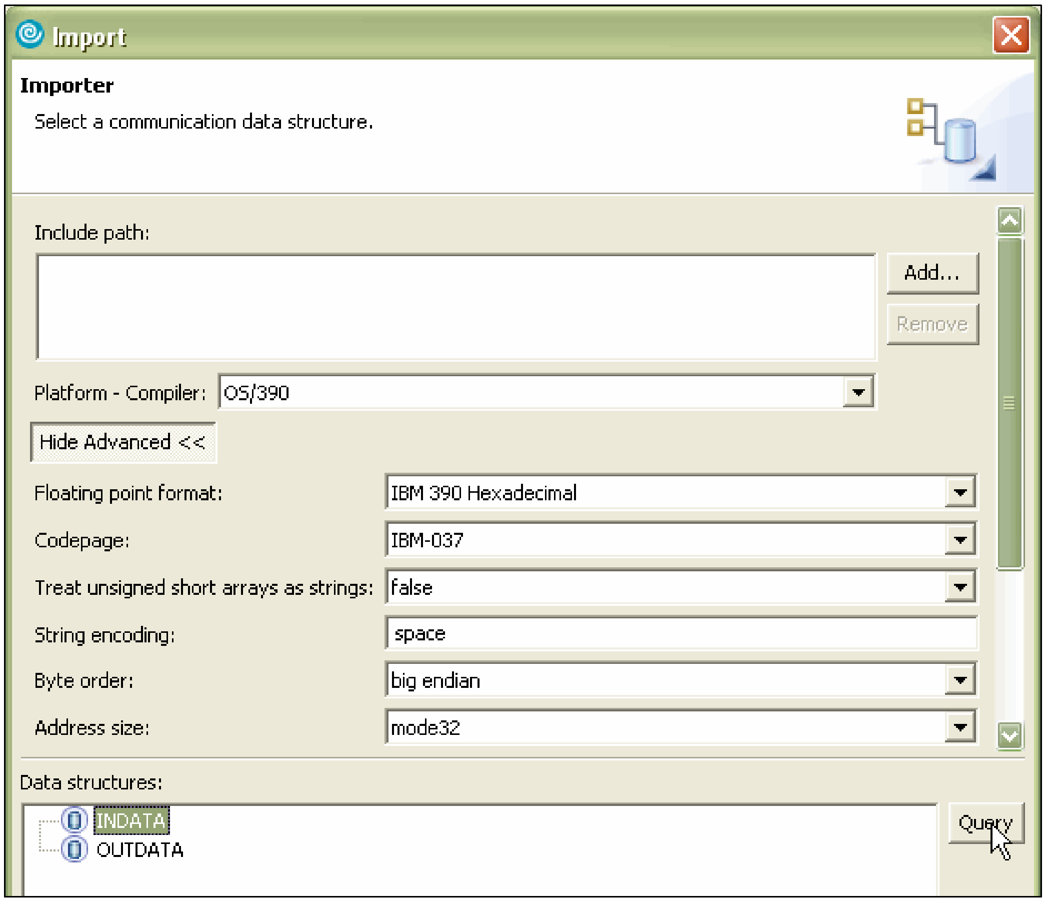
Import C source file:* C:\DISKE\RacWorkspace\workspaceI\IMSv92006Connector\reference.c.sj

Figure 8-31 Data import for inputmessage

Click **Next.**

In the new **Import** pop-up, set the platform value to **z/OS**, and codepage to **IBM-037**. If you do *not* see those options, click **Show Advanced**.

Click **Query**. This shows the available data sections in the c include. Select as input message layout **INDATA**; see Figure 8-32.


Import

Importer

Select a communication data structure.

Include path:

Platform - Compiler: OS/390

Floating point format: IBM 390 Hexadecimal

Codepage: IBM-037

Treat unsigned short arrays as strings: false

String encoding: space

Byte order: big endian

Address size: mode32

Data structures:

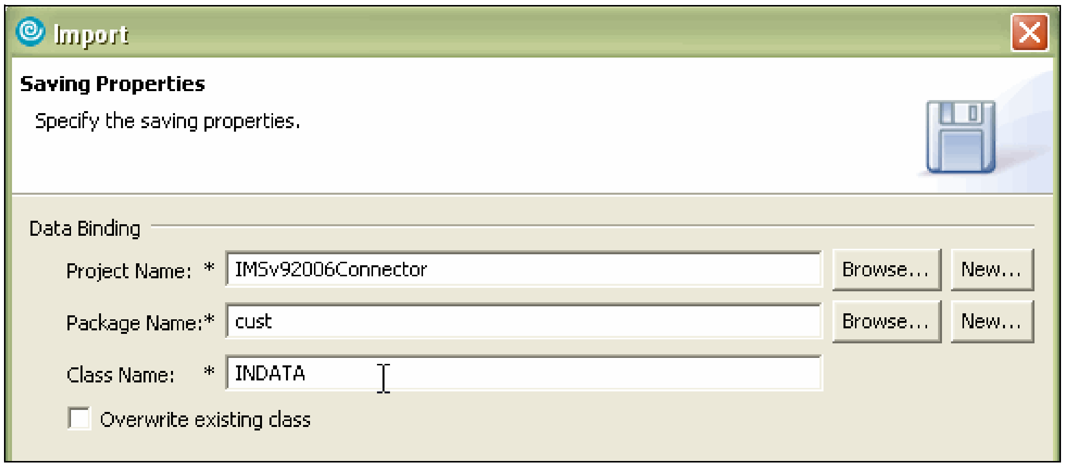
INDATA

OUTDATA

Figure 8-32 Input message properties

Click **Next.**

The input CCI resource bean is ready to be generated, the only missing information is the bean name and its package name. These are free selectable values, look at the example in Figure 8-33.


Import

Saving Properties

Specify the saving properties.

Data Binding

Project Name: * IMSv92006Connector

Package Name:* cust

Class Name: * INDATA

☐ Overwrite existing class

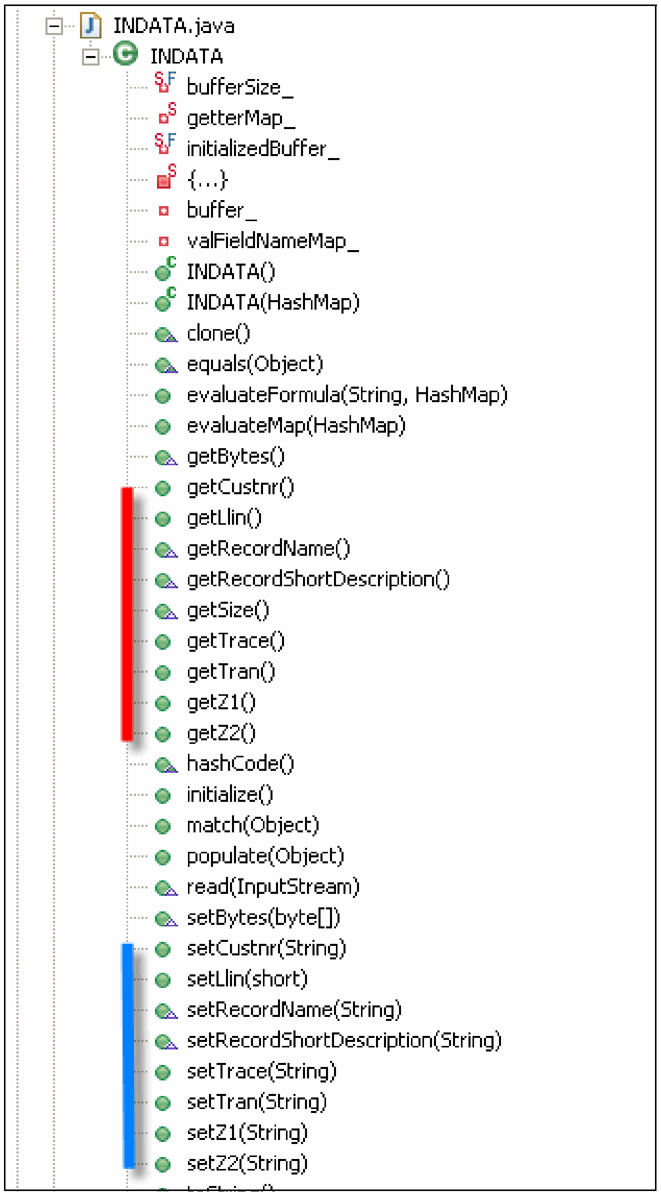
Figure 8-33 Input Saving properties

Click **Finish**

The databinding for the input message is generated now, and the java code is shown in the right frame of RAD. Instead of looking at the complete Java code, an excerpt of the outline of the code gives us a good impression of the methods that are available in this class.

For each field that was defined in the c include.h we have two accessors (getter/setter). The setter that is used for the inputmsg, while getters are used to extract the fields from the outputmsg bean. See Figure 8-34 on page 237.

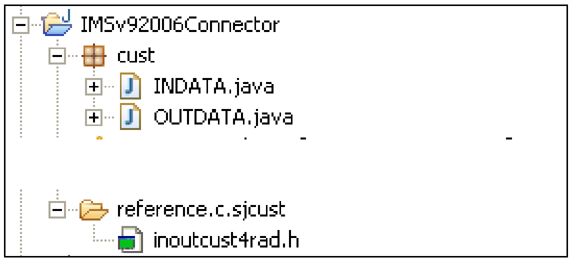
We still have to build the **outputmsg** bean, this happens exactly in the same way as for the inputmsg. We select the corresponding output section from the c include.h. One item which eventually could be different for output compared to input, is that we could have several possible output layouts, which is called **MPO** (multiple output). Here we keep it simple and stay with one possible output.



```
INDATA.java
  INDATA
    bufferSize_
    getterMap_
    initializedBuffer_
    {...}
    buffer_
    valFieldNameMap_
    INDATA()
    INDATA(HashMap)
    clone()
    equals(Object)
    evaluateFormula(String, HashMap)
    evaluateMap(HashMap)
    getBytes()
    getCustnr()
    getLlin()
    getRecordName()
    getRecordShortDescription()
    getSize()
    getTrace()
    getTran()
    getZ1()
    getZ2()
    hashCode()
    initialize()
    match(Object)
    populate(Object)
    read(InputStream)
    setBytes(byte[])
    setCustnr(String)
    setLlin(short)
    setRecordName(String)
    setRecordShortDescription(String)
    setTrace(String)
    setTran(String)
    setZ1(String)
    setZ2(String)
```

Figure 8-34 Outline for input CCI Databinding Javabean

The new look of the project after the databinding is complete is shown in Figure 8-35.



```
IMSv92006Connector
  cust
    INDATA.java
    OUTDATA.java
  reference.c.sjcust
    inoutcust4rad.h
```

Figure 8-35 Project with CCI input/ouput beans

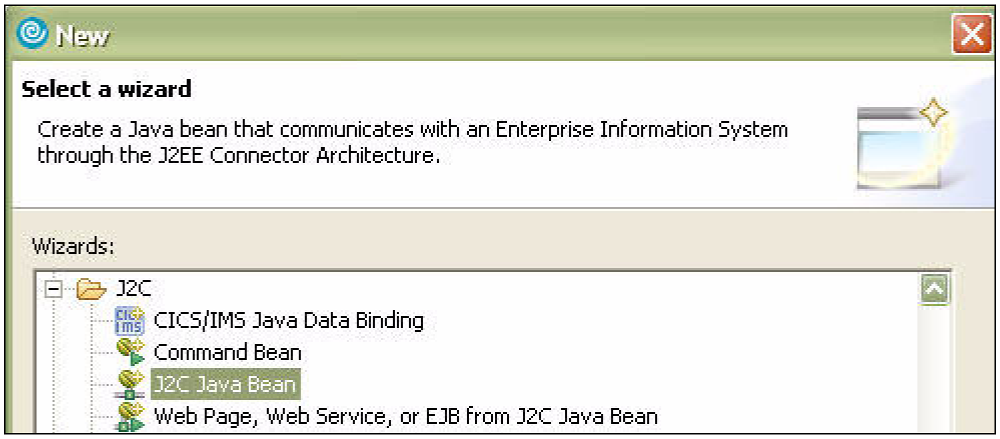
Notice the two databinding beans INDATA.java and OUTDATA.java.

J2C Bean

The next option of the RAD wizard generates the J2C Java bean. The J2C bean, which is the kernel of the J2C connector, uses the databinding beans as input and output, and based on ConnectionSpec and InteractionSpec information, it is generated by the wizard.

Select the previously created project **IMSv92006Connector** on Figure 8-36**.**

Right-click **New** → **Other.**


New

Select a wizard

Create a Java bean that communicates with an Enterprise Information System through the J2EE Connector Architecture.

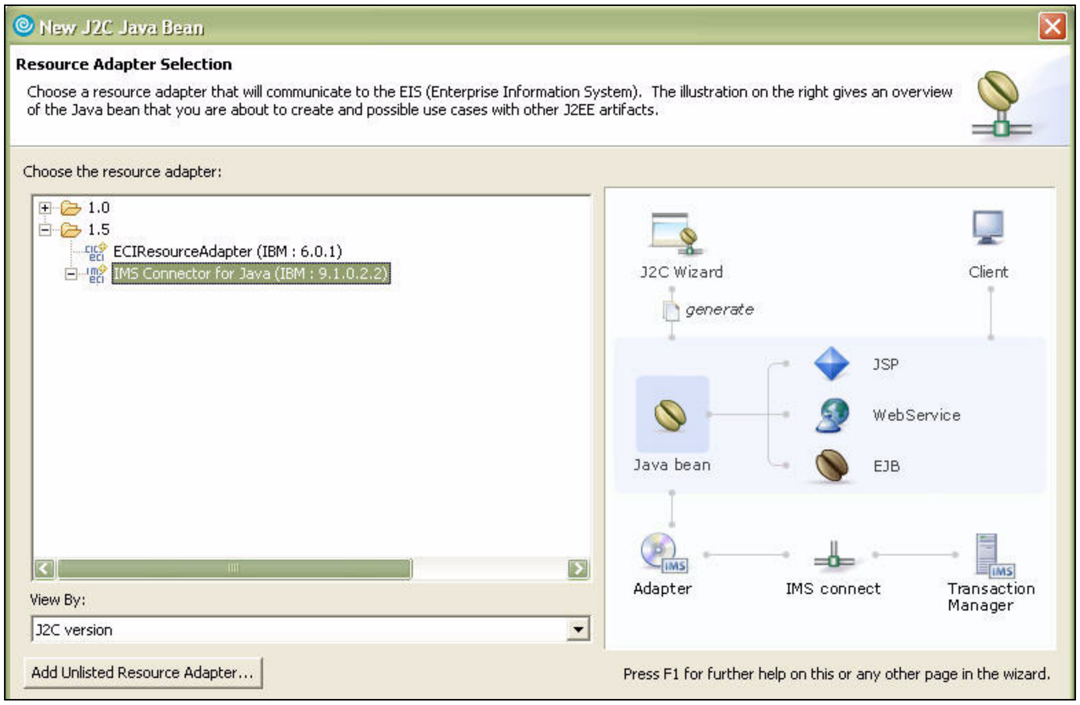
Wizards:

- J2C
  - CICS/IMS Java Data Binding
  - Command Bean
  - J2C Java Bean
  - Web Page, Web Service, or EJB from J2C Java Bean

Figure 8-36 J2C Bean phase selection

Click **Next.**

In the pop-up of Figure 8-37 we find, at the right a picture, the J2C bean that we are about to create and what interactions are possible with other artifacts.


New J2C Java Bean

Resource Adapter Selection

Choose a resource adapter that will communicate to the EIS (Enterprise Information System). The illustration on the right gives an overview of the Java bean that you are about to create and possible use cases with other J2EE artifacts.

Choose the resource adapter:

- 1.0
- 1.5
  - ECIResourceAdapter (IBM : 6.0.1)
  - IMS Connector for Java (IBM : 9.1.0.2.2)

View By: J2C version

Add Unlisted Resource Adapter...

Press F1 for further help on this or any other page in the wizard.

Figure 8-37 Resource Adapter selection

We also have to indicate what resource adapter has to be used. We select the previously installed **imscon** adapter, which is compliant with Java Connector Architecture (JCA) level 1.5.

Enter **Next.**

The new screen that comes up is related to the connection to IMS. It determines the properties of the **connectionFactory**. Connecting via TCP/IP to IMS requires following information:

- Host name (ipaddress)
- Port Number
- Commit Mode 0 dedicated (true/false)
- Datastore (IMSname)
- SSL enabled (true/false)
- Userid/Password

This information can be provided in two ways:

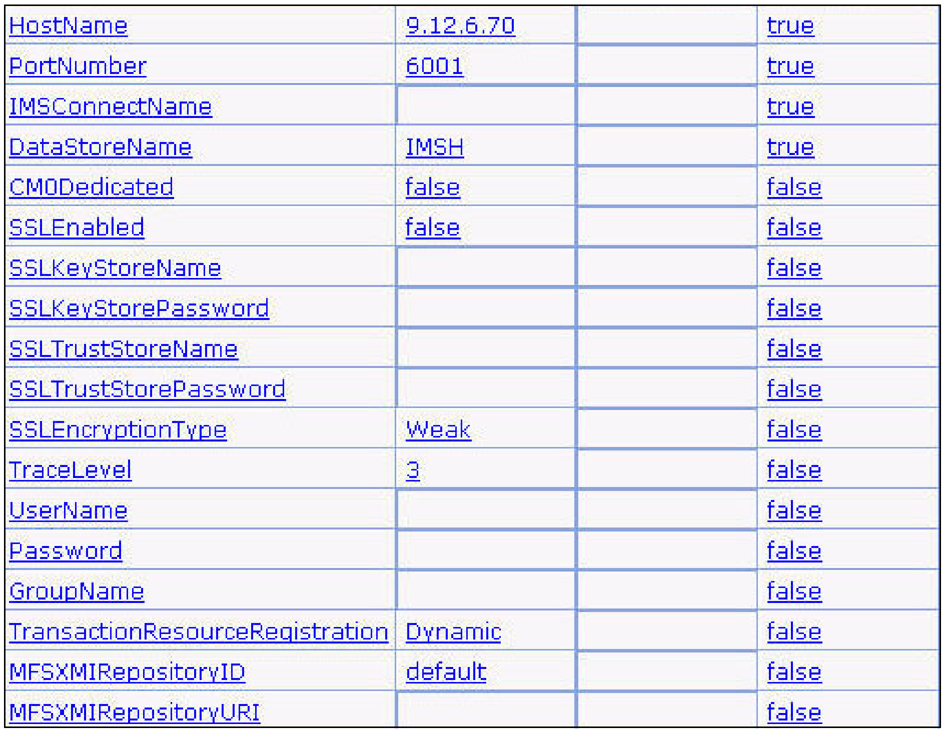
- **Managed**

  The required information has to be defined beforehand and stored in the WebSphere Application Server (WAS), from where it is retrievable via a JNDI lookup with name **ims/IMSSJ**. This retrieved information is used for connection with IMS via a connectionFactory.

- **Non Managed**

  All information is provided at runtime in the J2C bean.

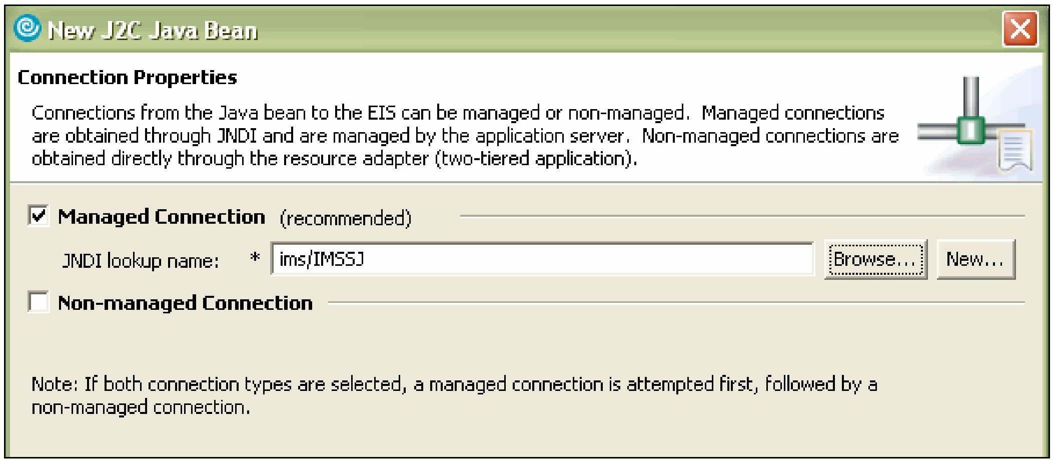
More information about it can be found in *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794. For this example we select **Managed.** The definition shown in Figure 8-38 on page 240 has been made in the WebSphere Test Environment under the IMSConnect resource. Once the application has been deployed on WebSphere Application Server on z/OS, this definition has to be made there as well.



HostName	9.12.6.70		true
PortNumber	6001		true
IMSConnectName			true
DataStoreName	IMSH		true
CM0Dedicated	false		false
SSLEnabled	false		false
SSLKeyStoreName			false
SSLKeyStorePassword			false
SSLTrustStoreName			false
SSLTrustStorePassword			false
SSLEncryptionType	Weak		false
TraceLevel	3		false
UserName			false
Password			false
GroupName			false
TransactionResourceRegistration	Dynamic		false
MFSXMIRepositoryID	default		false
MFSXMIRepositoryURI			false

Figure 8-38 Connection Factory definition with lookup ims/IMSSJ

In the managed case we must also choose a jndiName, which is used during the lookup of the connection resource. We give it the name **ims/IMSSJ**. See Figure 8-39.



Connection Properties

Connections from the Java bean to the EIS can be managed or non-managed. Managed connections are obtained through JNDI and are managed by the application server. Non-managed connections are obtained directly through the resource adapter (two-tiered application).

[✓] Managed Connection (recommended)

JNDI lookup name: * ims/IMSSJ

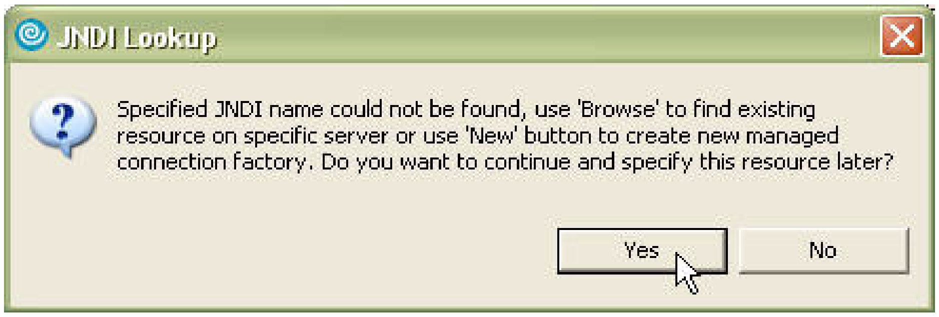
[ ] Non-managed Connection

Note: If both connection types are selected, a managed connection is attempted first, followed by a non-managed connection.

Figure 8-39 Managed Connection

Click **Next.**

The jndiName that we have chosen, does not exist yet in the WebSphere Test Environment in our development platform. We decide to continue and define it later. See Figure 8-40 on page 241.

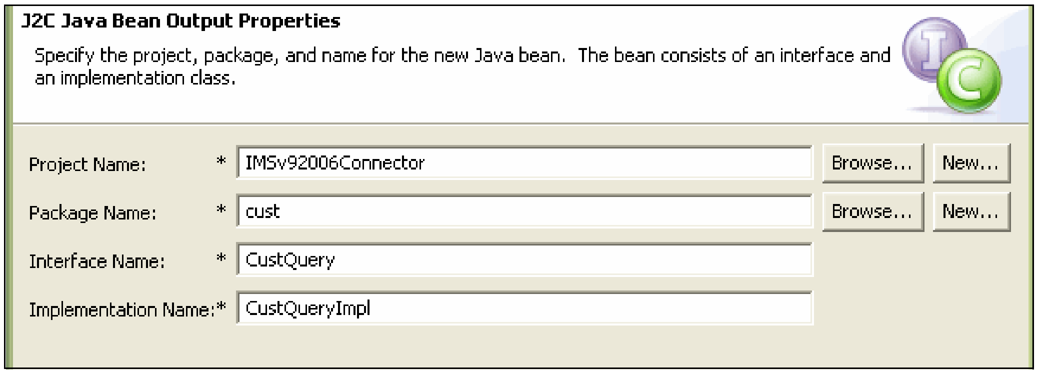


Specified JNDI name could not be found, use 'Browse' to find existing resource on specific server or use 'New' button to create new managed connection factory. Do you want to continue and specify this resource later?

Figure 8-40 Postpone managed connectFactory define

Click **Yes.**

We have to give a name for the **Interface** and the **Implementation** of the J2C bean, and also a name for the package it belongs to; see Figure 8-41.


J2C Java Bean Output Properties

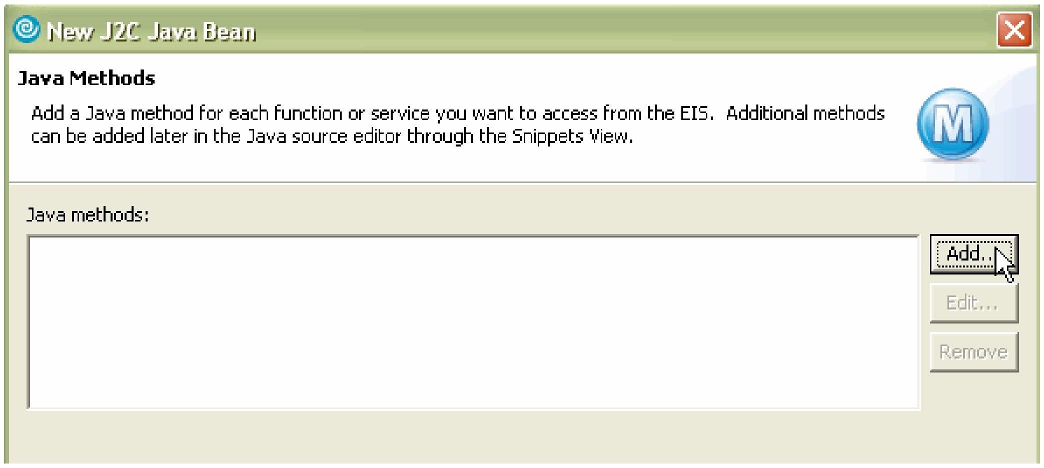
Specify the project, package, and name for the new Java bean. The bean consists of an interface and an implementation class.

Project Name: *	IMSv92006Connector	Browse...	New...
Package Name: *	cust	Browse...	New...
Interface Name: *	CustQuery		
Implementation Name:*	CustQueryImpl		

Figure 8-41 Package,Interface, Implementation

Click **Next.**

We have now to add names for the business methods mapped to the transactions that have to be executed via this J2C bean. Enter **cquery**. See Figure 8-42.



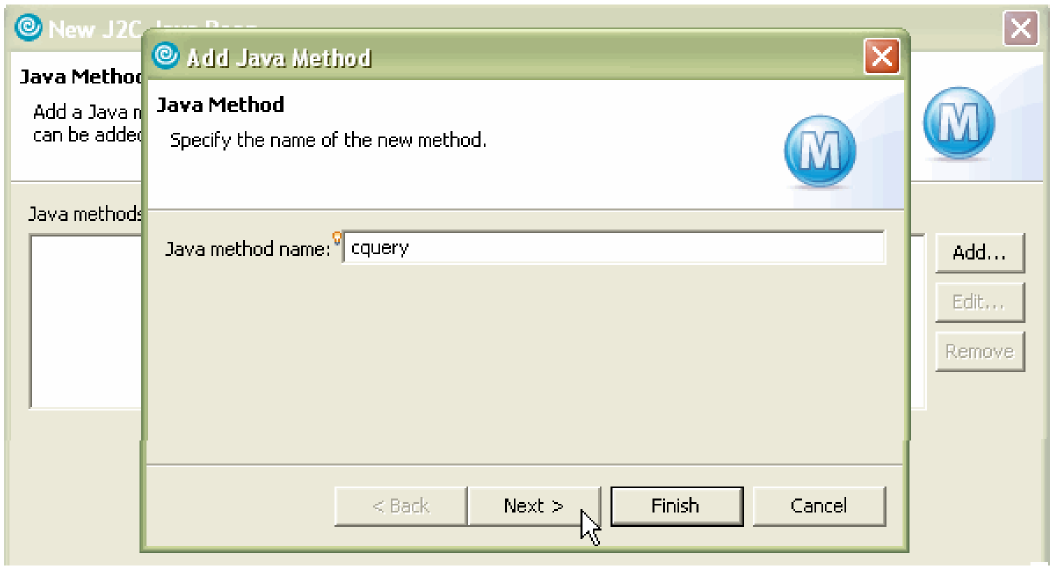
Java Methods

Add a Java method for each function or service you want to access from the EIS. Additional methods can be added later in the Java source editor through the Snippets View.

Java methods:

Figure 8-42 Add business method

Click **Add.**



Java Method

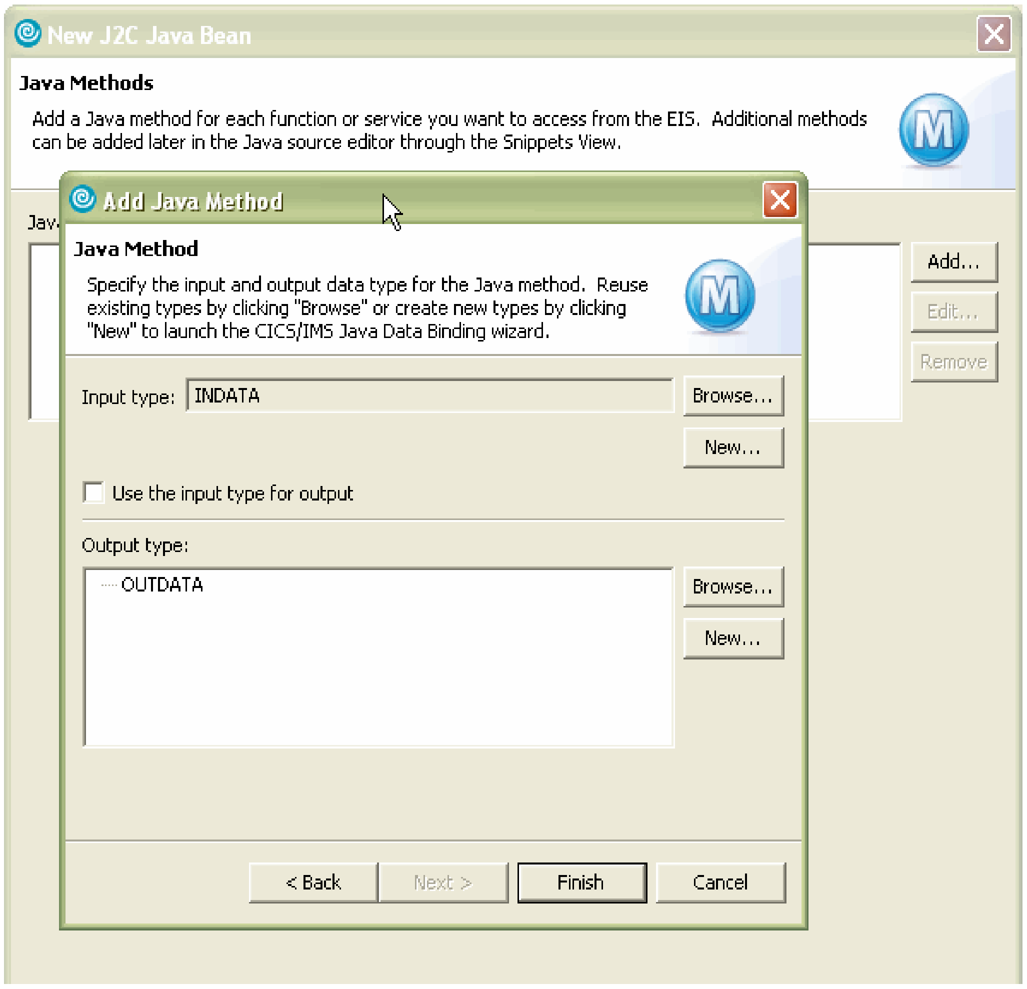
Specify the name of the new method.

Java method name: cquery

Figure 8-43   Add cquery method

Click **Next.**

The business method requires an input and output message. This is specified in the panel of Figure 8-44 on page 243.


New J2C Java Bean

Java Methods

Add a Java method for each function or service you want to access from the EIS. Additional methods can be added later in the Java source editor through the Snippets View.

Add Java Method

Java Method

Specify the input and output data type for the Java method. Reuse existing types by clicking "Browse" or create new types by clicking "New" to launch the CICS/IMS Java Data Binding wizard.

Input type: INDATA

[ ] Use the input type for output

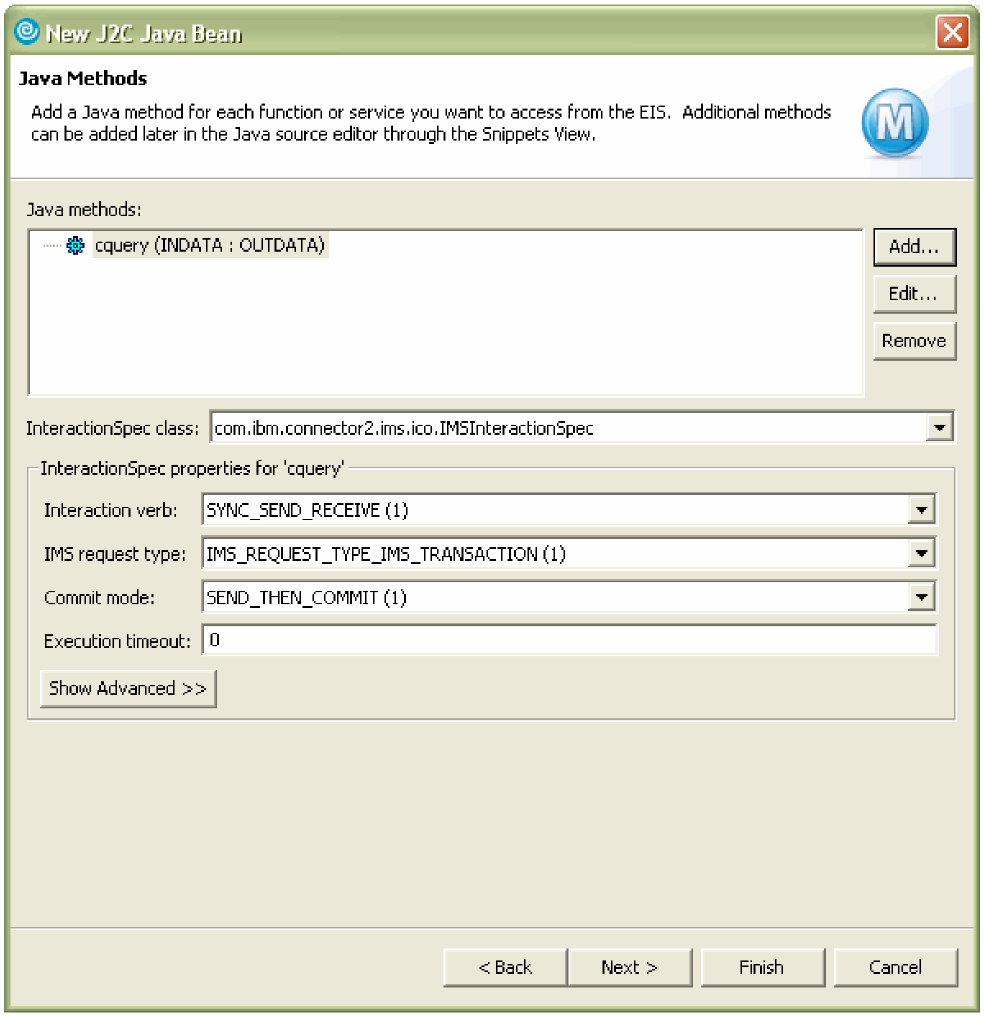
Output type: OUTDATA

Figure 8-44   Indicate input/output for business method

With the Browse button, lookup for input and output the databinding beans that were generated earlier on and select them.

Click **Finish.**

Each exchange with IMS (transaction, command) has its own interaction specifications, which are passed as an InteractionSpec object during the java call in the J2Cbean. Many combinations are possible within the boundary of the connection specification. More information about it can be found in *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-67944. For the transaction invocation corresponding with the above defined business method on the J2C bean, we select the options proposed by default. See Figure 8-45 on page 244.


New J2C Java Bean

Java Methods

Add a Java method for each function or service you want to access from the EIS. Additional methods can be added later in the Java source editor through the Snippets View.

Java methods:

cquery (INDATA : OUTDATA)

InteractionSpec class: com.ibm.connector2.ims.ico.IMSInteractionSpec

InteractionSpec properties for 'cquery'

Interaction verb: SYNC_SEND_RECEIVE (1)

IMS request type: IMS_REQUEST_TYPE_IMS_TRANSACTION (1)

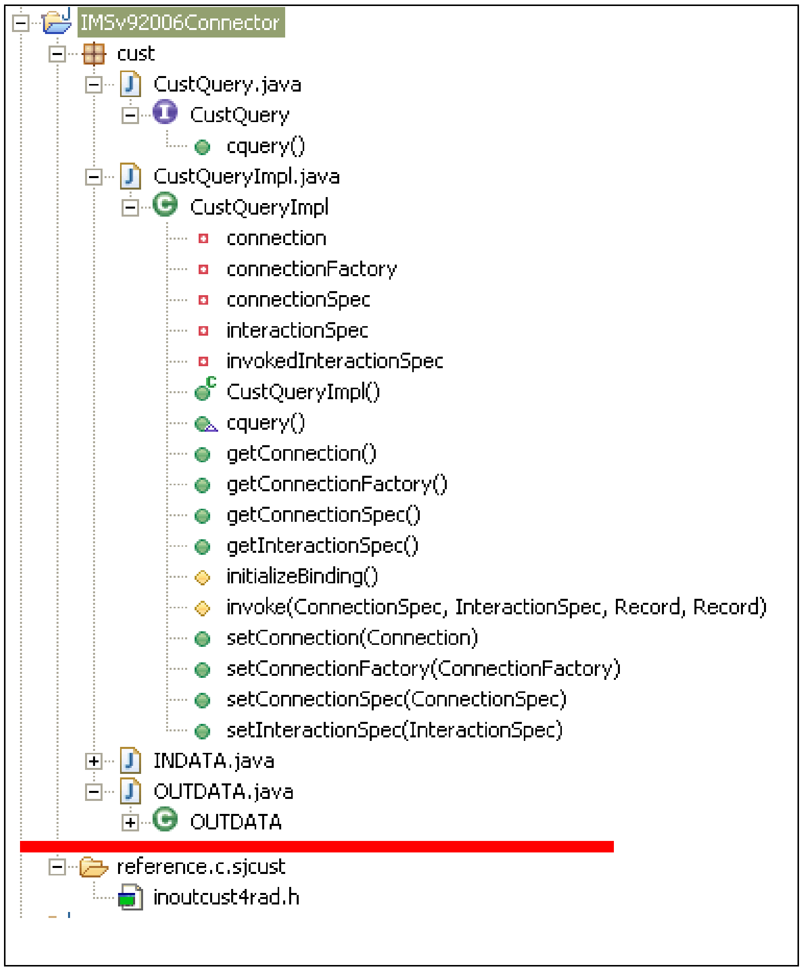
Commit mode: SEND_THEN_COMMIT (1)

Execution timeout: 0

Figure 8-45 InteractionSpec properties

Click **Finish.**

The J2C bean is now generated. The code should be alright, and we can look via the outline. See Figure 8-46 on page 245.



```
IMSv92006Connector
  cust
    CustQuery.java
      CustQuery
        cquery()
    CustQueryImpl.java
      CustQueryImpl
        connection
        connectionFactory
        connectionSpec
        interactionSpec
        invokedInteractionSpec
        CustQueryImpl()
        cquery()
        getConnection()
        getConnectionFactory()
        getConnectionSpec()
        getInteractionSpec()
        initializeBinding()
        invoke(ConnectionSpec, InteractionSpec, Record, Record)
        setConnection(Connection)
        setConnectionFactory(ConnectionFactory)
        setConnectionSpec(ConnectionSpec)
        setInteractionSpec(InteractionSpec)
    INDATA.java
    OUTDATA.java
      OUTDATA
  reference.c.sjcust
    inoutcust4rad.h
```

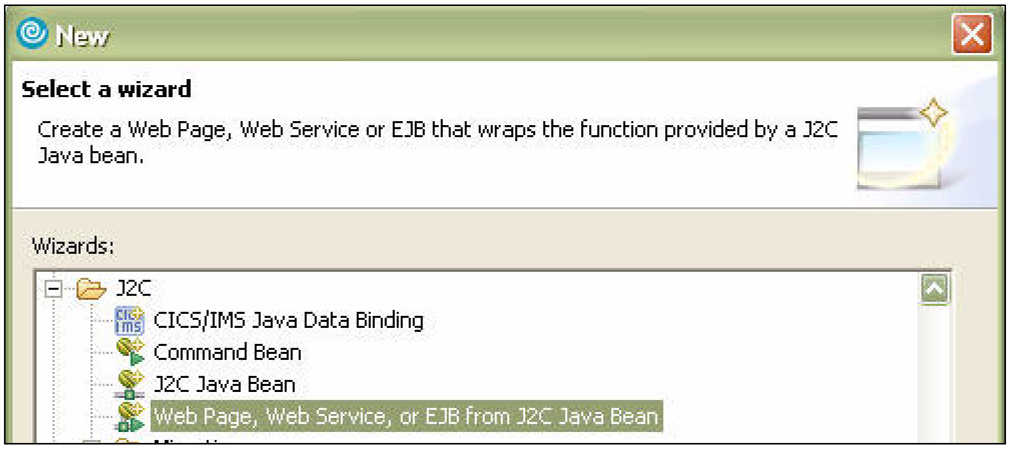
Figure 8-46 IMSv92006Connector project outline withJ2C bean and databindings

The new look of the **IMSv92006Connector** project is shown in Figure 8-47 on page 246. It now contains the J2C bean implementation. You recognize the added **cquery** method, with input and output.

Web services on the J2Cbean

To use this J2C bean (connector), several options are available (Java Server Pages (servlet by JSP), EJB (session), Web services. We could first build an intermediary EJB, and use this EJB as the Endpoint for the Web Services, or we can generate an HTTP based solution directly on the J2C bean. The method **cquery** is the service entrypoint for Web services.

Select the previously created project **IMSv92006Connector.** Right-click **New** → **Other.** You get to Figure 8-47 on page 246.


New

Select a wizard

Create a Web Page, Web Service or EJB that wraps the function provided by a J2C Java bean.

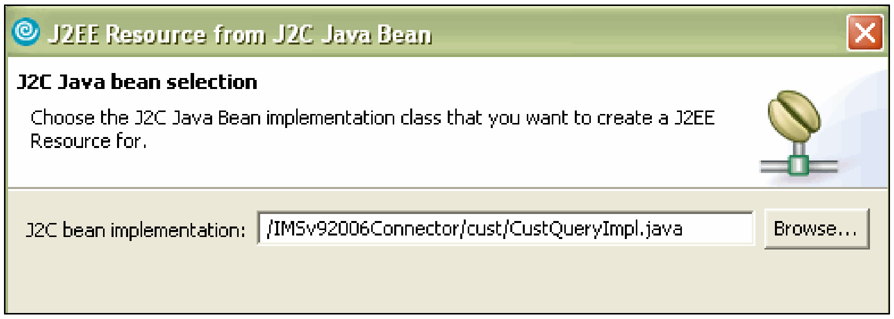
Wizards:

- J2C
  - CICS/IMS Java Data Binding
  - Command Bean
  - J2C Java Bean
  - Web Page, Web Service, or EJB from J2C Java Bean

Figure 8-47 Web Services phase selection

Click **Next.**

In the new screen of Figure 8-48, **Browse** to the appropriate J2C bean implementation.


J2EE Resource from J2C Java Bean

J2C Java bean selection

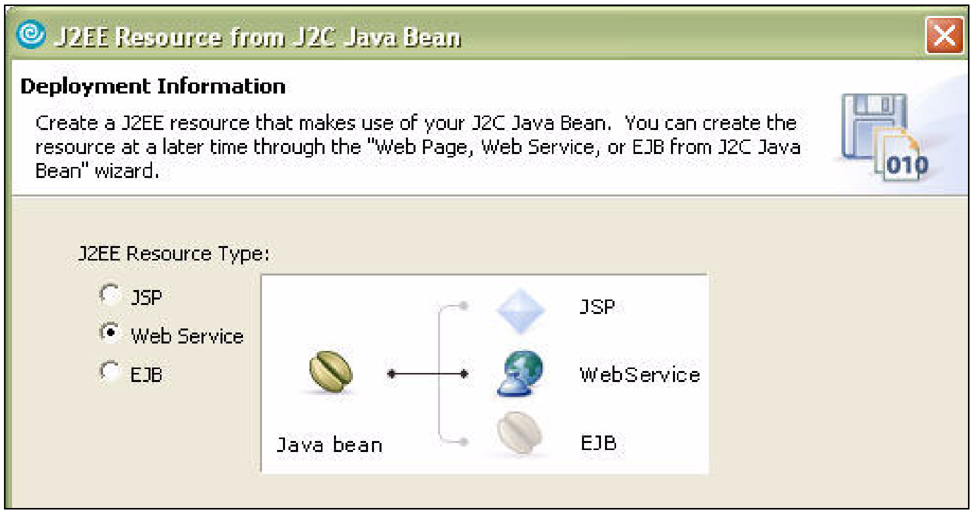
Choose the J2C Java Bean implementation class that you want to create a J2EE Resource for.

J2C bean implementation: /IMSv92006Connector/cust/CustQueryImpl.java

Figure 8-48 J2C bean selection for Web services

Click **Next.**

Continuing now requires that the connectionFactory be defined in the WebSphere Test Environment (WTE). We decided to use a managed connection, and as a result a predefined definition has to be available, retrievable by its jndiName, in our case **ims/IMSSJ.** This definition has to be made by the WTE Admin Console. The availability of the definition is checked by the wizard. If the condition is alright, the panel of Figure 8-49 on page 247 comes up, otherwise an error message indicates the missing resource.


J2EE Resource from J2C Java Bean

Deployment Information

Create a J2EE resource that makes use of your J2C Java Bean. You can create the resource at a later time through the "Web Page, Web Service, or EJB from J2C Java Bean" wizard.

J2EE Resource Type:

JSP [✓ Web Service] EJB

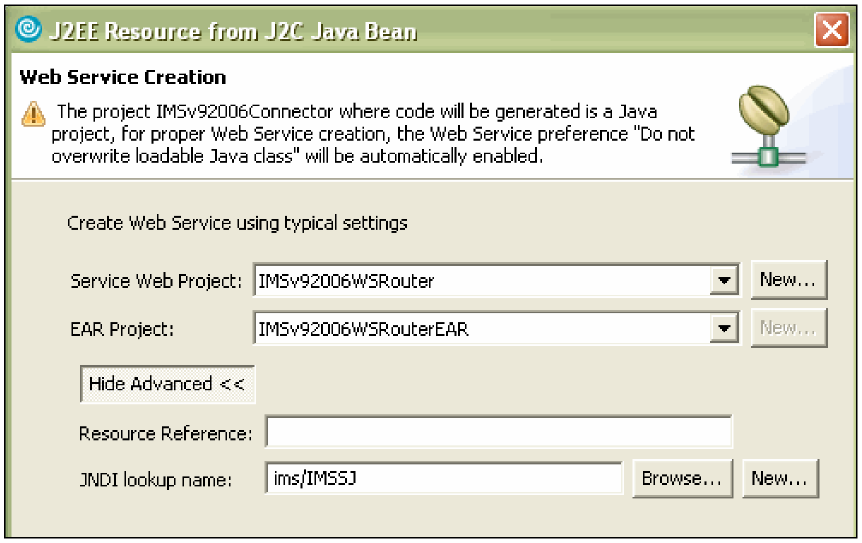
Figure 8-49 Web services deployment

Select in this screen the **Web Service** option.

Click **Next.**

On the next panel Web Service Creation, next to the label Service Web Project, click **New** if the Dynamic Web project does not exist otherwise select the project from the list. If creating a Dynamic Web Project, enter data in the pop-up as indicated in Figure 8-50.

The service endpoint for an HTTP Web Service is a servlet. A servlet resides in a WebApp and is built in a Dynamic Web Project.


J2EE Resource from J2C Java Bean

Web Service Creation

The project IMSv92006Connector where code will be generated is a Java project, for proper Web Service creation, the Web Service preference "Do not overwrite loadable Java class" will be automatically enabled.

Create Web Service using typical settings

Service Web Project: IMSv92006WSRouter

EAR Project: IMSv92006WSRouterEAR

Resource Reference:

JNDI lookup name: ims/IMSSJ

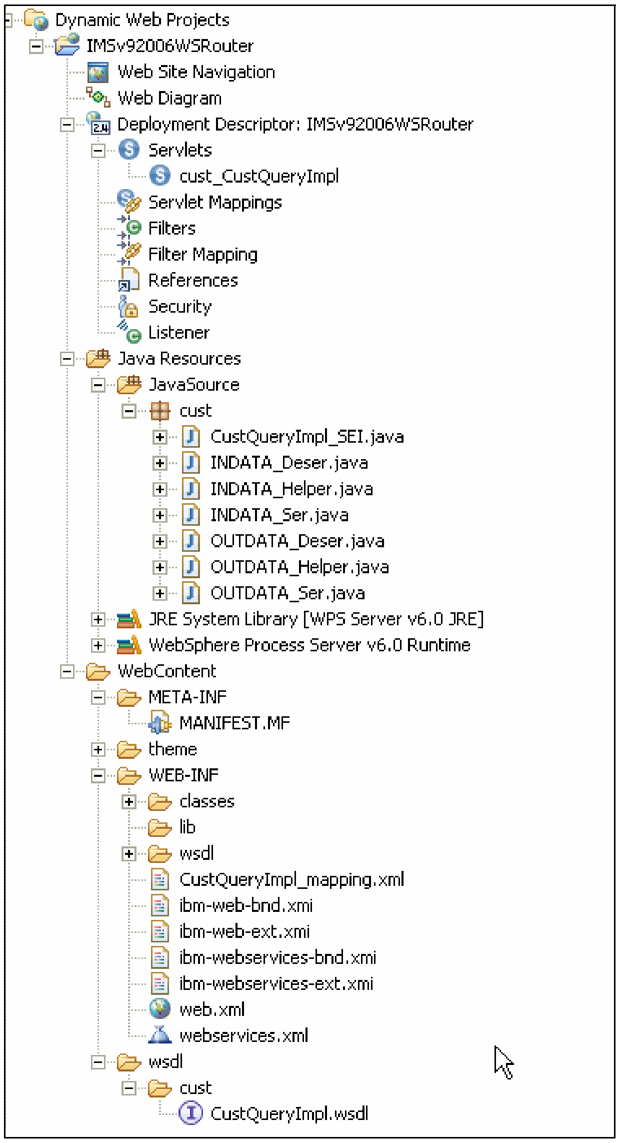
Figure 8-50 Dynamic Webproject for Web services

It is not required to click continue. Click **Finish** immediately.

It is *not* advisable now to switch to a Web Perspective. Click **No**, and verify the new proposed Web Service Creation parameters.

Click **Finish.**

RAD is now building the Web services, and also deploys it immediately into the WTE. The WTE server gets started. The entry point into this Web service is through HTTP. The layout of the new **IMSv92006WSRouter** project is shown in Figure 8-51.



```
Dynamic Web Projects
  IMSv92006WSRouter
    Web Site Navigation
    Web Diagram
    Deployment Descriptor: IMSv92006WSRouter
      Servlets
        cust_CustQueryImpl
      Servlet Mappings
      Filters
      Filter Mapping
      References
      Security
      Listener
    Java Resources
      JavaSource
        cust
          CustQueryImpl_SEI.java
          INDATA_Deser.java
          INDATA_Helper.java
          INDATA_Ser.java
          OUTDATA_Deser.java
          OUTDATA_Helper.java
          OUTDATA_Ser.java
      JRE System Library [WPS Server v6.0 JRE]
      WebSphere Process Server v6.0 Runtime
    WebContent
      META-INF
        MANIFEST.MF
      theme
      WEB-INF
        classes
        lib
        wsdl
        CustQueryImpl_mapping.xml
        ibm-web-bnd.xmi
        ibm-web-ext.xmi
        ibm-webservices-bnd.xmi
        ibm-webservices-ext.xmi
        web.xml
        webservices.xml
      wsdl
        cust
          CustQueryImpl.wsdl
```

Figure 8-51 WebProject IMSv92006WS for Web Services access

All contents under WebContent are packaged in a WAR file and will be deployed in the WebSphere container. Notice, under classes, the helper classes for XML marshalling and demarshalling, also the additional deployment descriptors and a WSDL file. This WSDL file can be used for testing the Web Service.

Testing the Web Services from RAD

Select the J2EE perspective, and look under the Project Explorer for Web services. expand the Services section, and then further expand the service CustQueryImplService, further expand it and select the WSDL. See Figure 8-52 on page 249.

Right-click **Test with Web Service Explorer.**

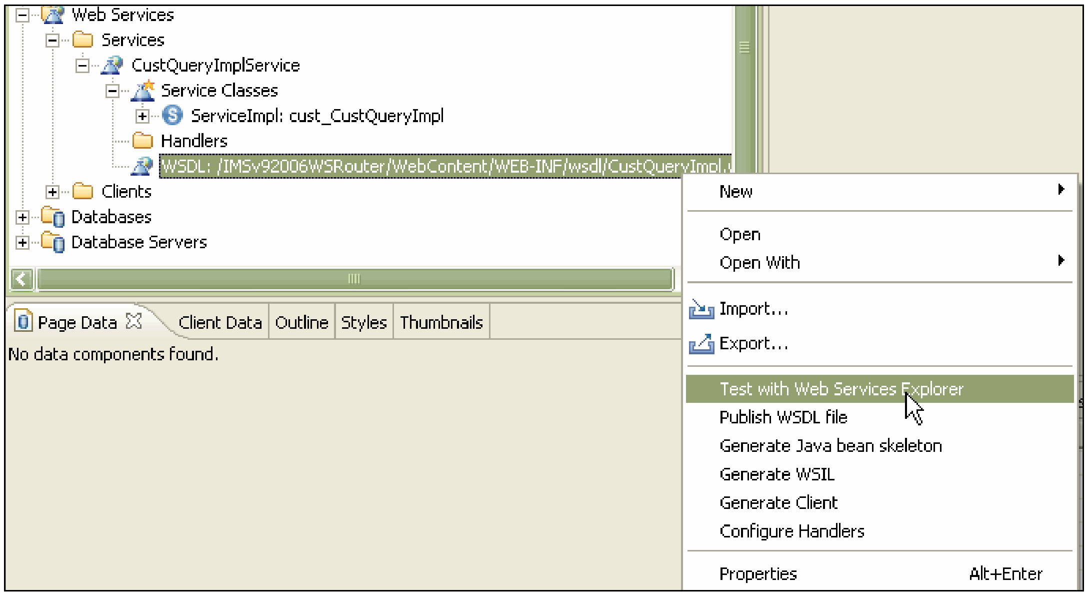
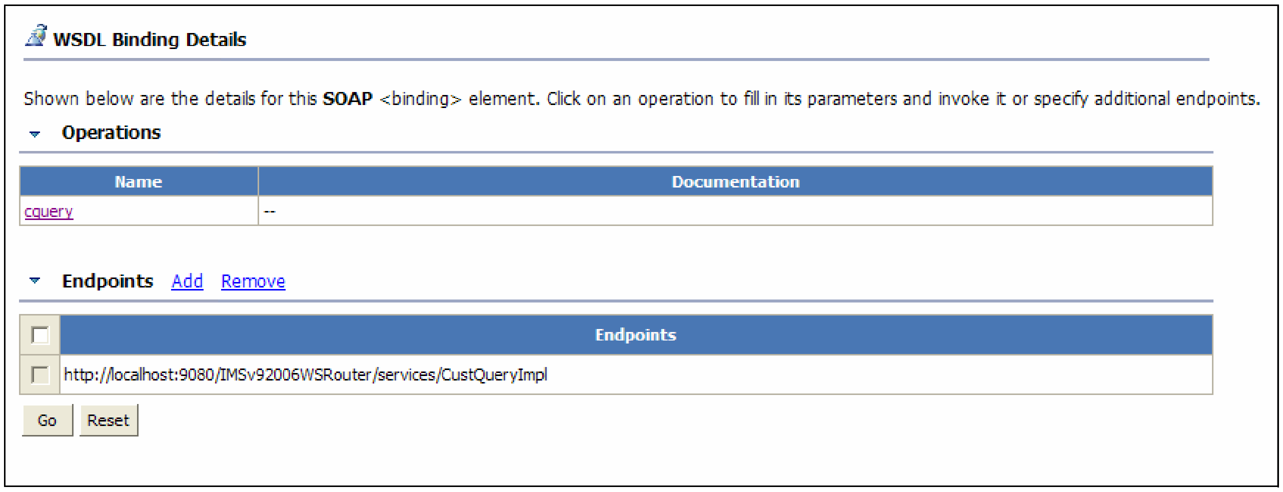


Figure 8-52 Testing the Web Service

The browser window of Figure 8-53 appears.



WSDL Binding Details

Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

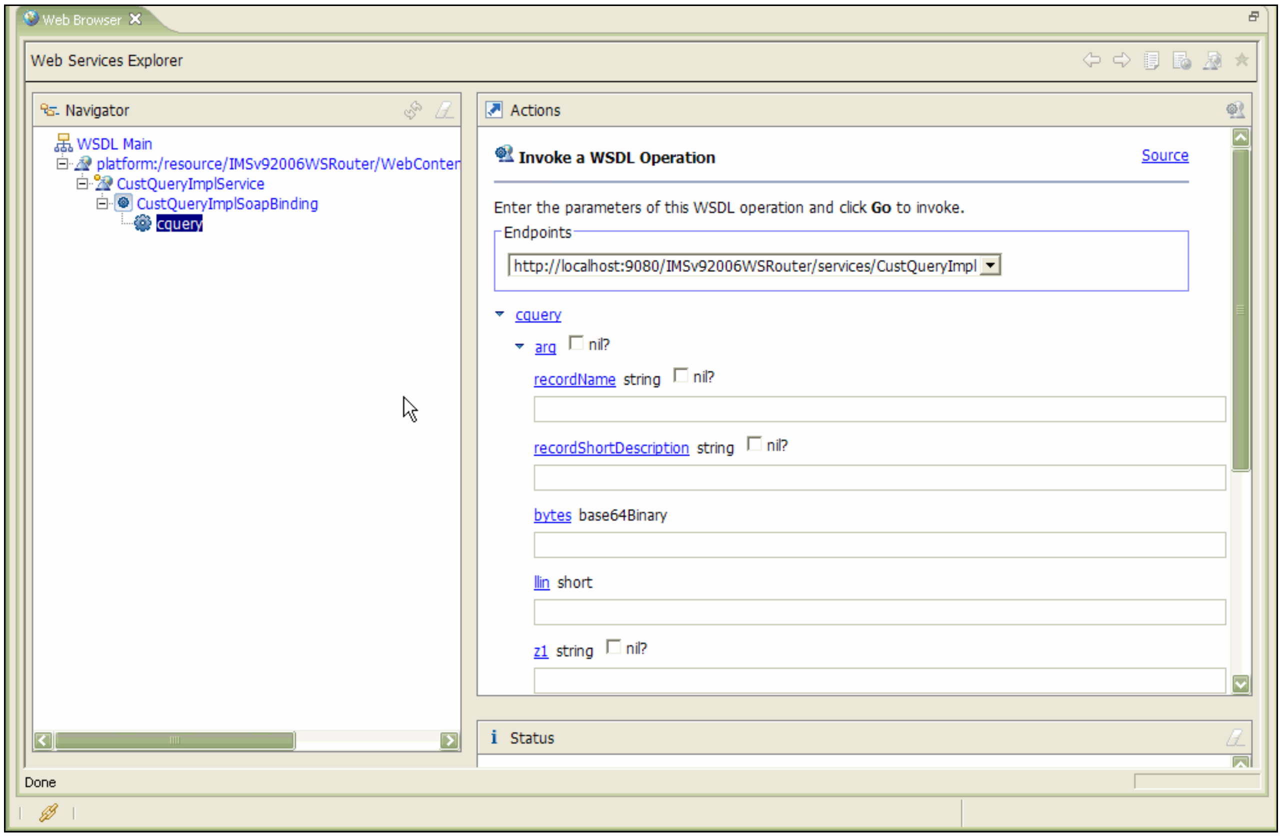
Name	Documentation
cquery	--

Endpoints Add Remove

	Endpoints
☐	http://localhost:9080/IMSv92006WSRouter/services/CustQueryImpl

Figure 8-53 Test Browser window

Within operations in the top/left of the window, click **cquery**, this brings up another browser screen from which you can test the J2C connector. Enter the values in the panel of Figure 8-54 on page 250 as required by the transaction.



Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://localhost:9080/IMSv92006WSRouter/services/CustQueryImpl

cquery

arg ☐ nil?

recordName string ☐ nil?

recordShortDescription string ☐ nil?

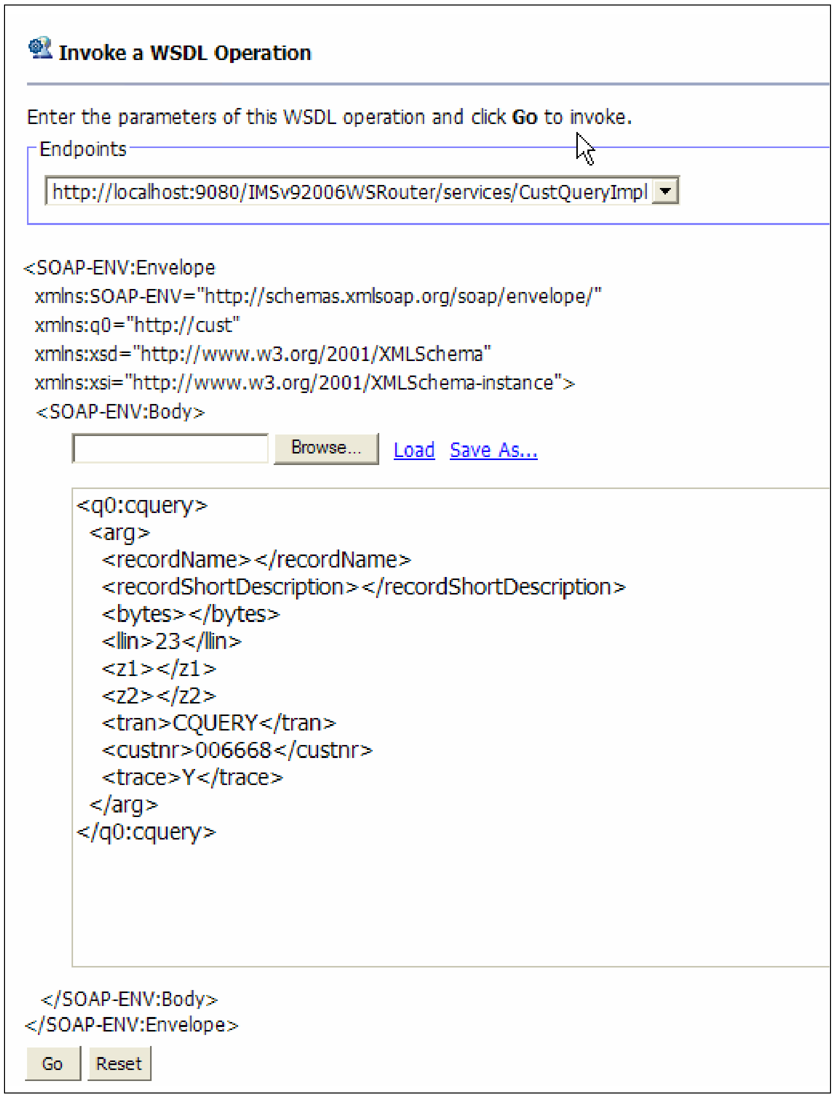
bytes base64Binary

llin short

z1 string ☐ nil?

Figure 8-54 Input browser screen

Enter data as expected by the program, click **Go**. This initiates the test sequence with the Web service deployed on Windows. The screen now shows the SOAP envelope as it is created by the test environment. See Figure 8-55 on page 251.


Invoke a WSDL Operation
Enter the parameters of this WSDL operation and click **Go** to invoke.
Endpoints
http://localhost:9080/IMSv92006WSRouter/services/CustQueryImpl
Browse...
Load
Save As...
Go
Reset

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:q0="http://cust"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
<q0:cquery>
  <arg>
    <recordName></recordName>
    <recordShortDescription></recordShortDescription>
    <bytes></bytes>
    <llin>23</llin>
    <z1></z1>
    <z2></z2>
    <tran>CQUERY</tran>
    <custnr>006668</custnr>
    <trace>Y</trace>
  </arg>
</q0:cquery>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 8-55 Inbound SOAP envelope

This window shows you the exact format of the SOAP frame that is transferred to the Web services. Notice the XML tokens for the SOAP envelope and subsections (only Body in this case) and also the XML transfer of all user data.

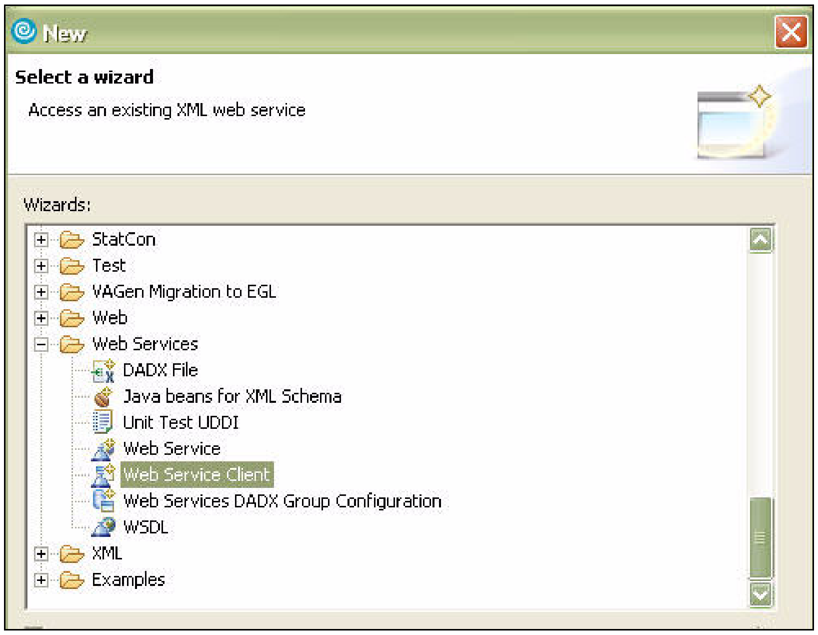
Web service client

A Web services is a client/server service. The client, in most of the cases is a programmed client. To help build this client in Java, the RAD developer has a wizard. With the help of this tooling artifacts, like proxies, helper classes for [de]marshalling, locator classes can be generated. Using those artifacts building a client becomes easy.

Verify that you are in a J2EE perspective in RAD, under Dynamic Web Projects, in project **IMSv92006WSRouter**, and select **WebContent/wsdl/cust/CustQueryImpl.wsdl.**

Right-click **New** → **Other.**

In the **New** pop-up, select in the wizard part for Web Services the option **Web Services Client,** see Figure 8-56 on page 252.


New

Select a wizard

Access an existing XML web service

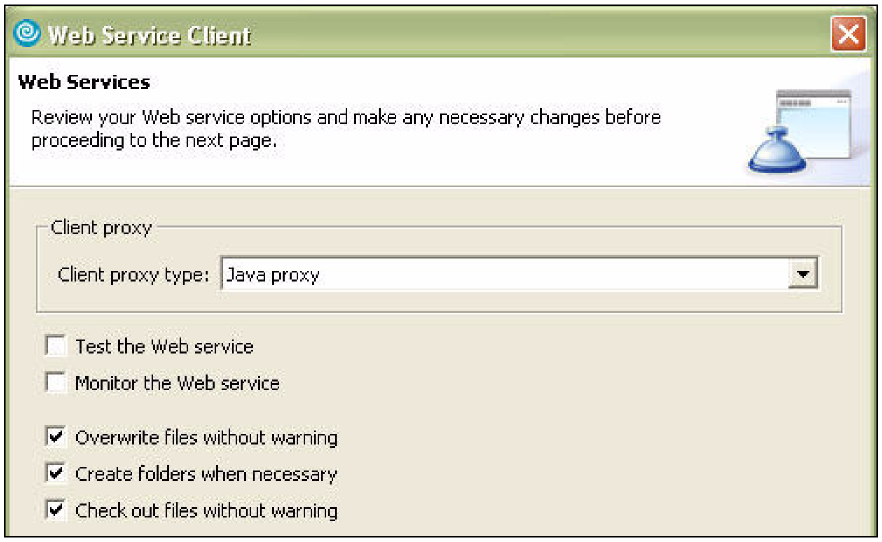
Wizards:

- StatCon
- Test
- VAGen Migration to EGL
- Web
- Web Services
  - DADX File
  - Java beans for XML Schema
  - Unit Test UDDI
  - Web Service
  - Web Service Client
  - Web Services DADX Group Configuration
  - WSDL
- XML
- Examples

Figure 8-56 Web Service Client selection

Click **Next.**

In the following pop-up we specify what kind of Java proxy we want to have. See Figure 8-57.


Web Service Client

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Client proxy

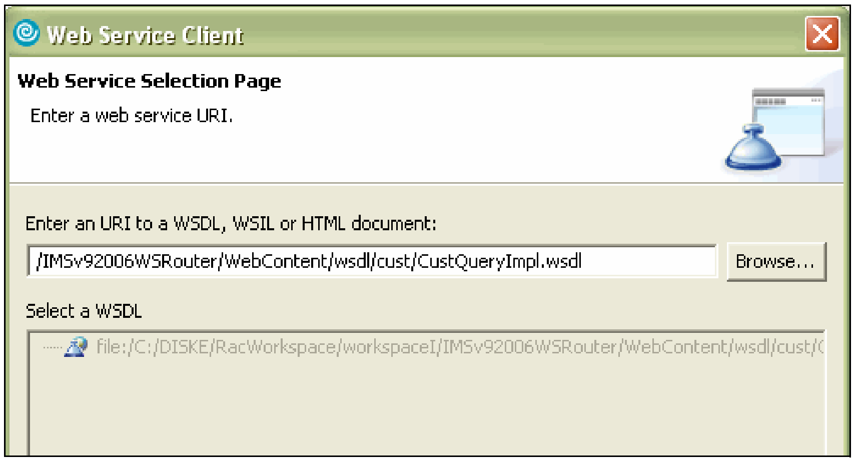
Client proxy type: Java proxy

- [ ] Test the Web service
- [ ] Monitor the Web service
- [x] Overwrite files without warning
- [x] Create folders when necessary
- [x] Check out files without warning

Figure 8-57 Client proxy options

Set the options as indicated, and click **Next.**

In the next panel (Figure 8-58 on page 253), verify the WSDL selection.



Web Service Client

Web Service Selection Page

Enter a web service URI.

Enter an URI to a WSDL, WSIL or HTML document:

/IMSv92006WSRouter/WebContent/wsdl/cust/CustQueryImpl.wsdl

Select a WSDL

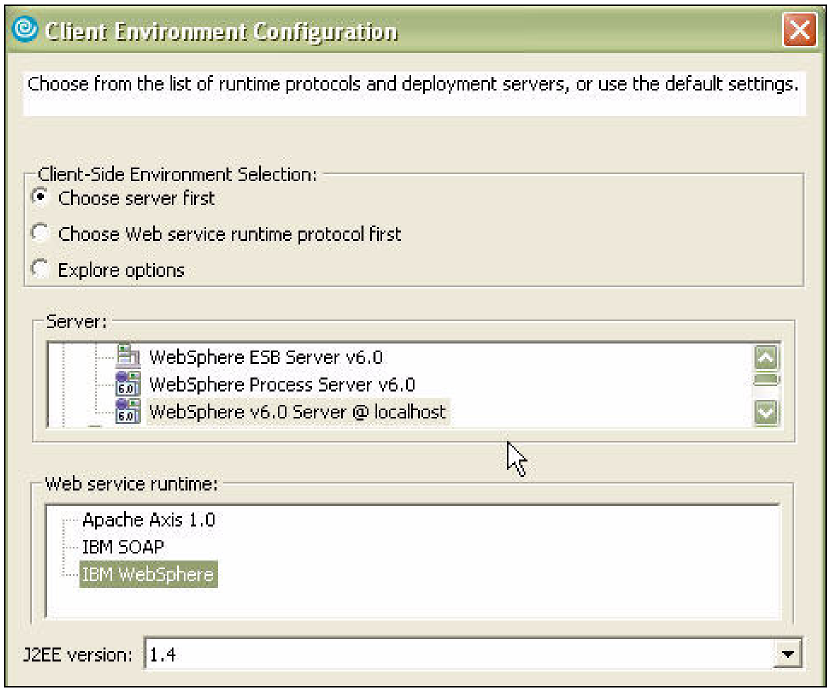
file:/C:/DISKE/RacWorkspace/workspaceI/IMSv92006WSRouter/WebContent/wsdl/cust/C

Figure 8-58 WSDL selection

If alright, click **Next**; otherwise fill in the correct **WSDL** file reference.

In the next window, by clicking **Edit**, set first the Web service run time and server to the values as shown in Figure 8-59.

- It could be that this panel does not come up, if the selection was implicitly made before.



Client Environment Configuration

Choose from the list of runtime protocols and deployment servers, or use the default settings.

Client-Side Environment Selection:
- [✓ Choose server first]
- Choose Web service runtime protocol first
- Explore options

Server:
- WebSphere ESB Server v6.0
- WebSphere Process Server v6.0
- WebSphere v6.0 Server @ localhost

Web service runtime:
- Apache Axis 1.0
- IBM SOAP
- [✓ IBM WebSphere]

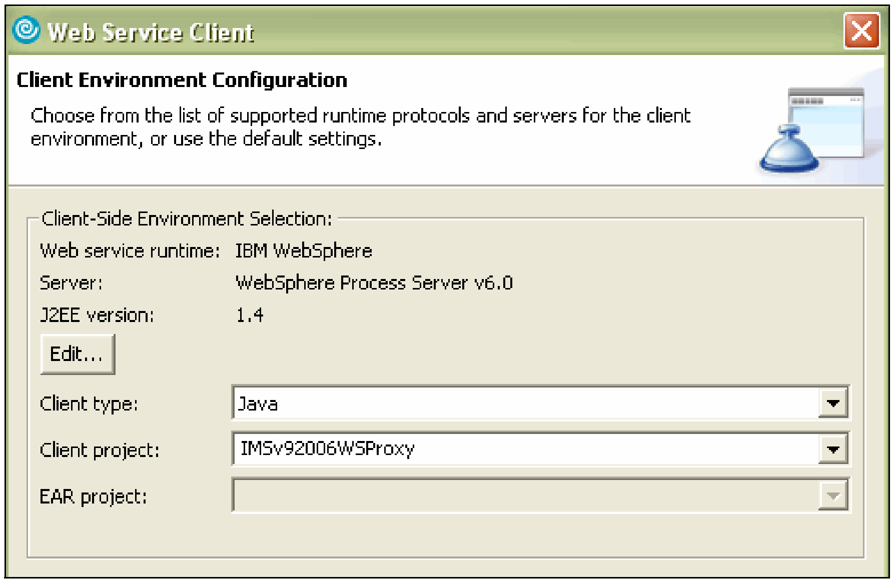
J2EE version: 1.4

Figure 8-59 Client-Side Environment server

Click **OK.** Look at Figure 8-59.

Verify the settings of the Web service runtime and Server, change the Client type and Client project as indicated in Figure 8-60 on page 254.

The client project, if *not* already existing yet, will be created.


Web Service Client

Client Environment Configuration

Choose from the list of supported runtime protocols and servers for the client environment, or use the default settings.

Client-Side Environment Selection:

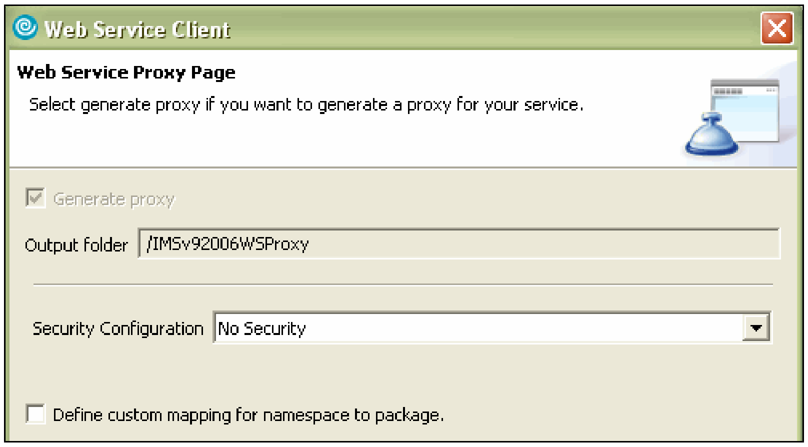
Web service runtime:	IBM WebSphere
Server:	WebSphere Process Server v6.0
J2EE version:	1.4

Edit...

Client type:	Java
Client project:	IMSv92006WSProxy
EAR project:	

Figure 8-60 Web service client project

Click **Next.** See Figure 8-61**.**


Web Service Client

Web Service Proxy Page

Select generate proxy if you want to generate a proxy for your service.

☑ Generate proxy

Output folder /IMSv92006WSProxy

Security Configuration No Security

☐ Define custom mapping for namespace to package.

Figure 8-61 Proxy

Verify the data in panel Web Service Proxy page see Figure 8-61, accept by clicking **Finish,** if correct.

The client artifacts are now generated in a newly created Java project. This takes a while as the server is also started. The final project is in Figure 8-62 on page 255.

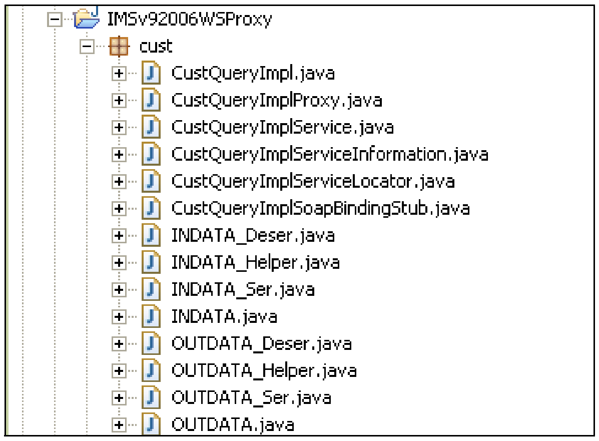

IMSv92006WSProxy
cust
CustQueryImpl.java
CustQueryImplProxy.java
CustQueryImplService.java
CustQueryImplServiceInformation.java
CustQueryImplServiceLocator.java
CustQueryImplSoapBindingStub.java
INDATA_Deser.java
INDATA_Helper.java
INDATA_Ser.java
INDATA.java
OUTDATA_Deser.java
OUTDATA_Helper.java
OUTDATA_Ser.java
OUTDATA.java

Figure 8-62 Client project

Notice the proxyclass that we will use in the client coding.

Building the Java Web services client

Web Services clients can be of two types:

1. **Managed clients**

   Run in a J2EE environment, this can be in the WebSphere Application Server container or in a J2EE client container.

2. **UnManaged clients**

   Are standalone Java Code.

In each category, we distinguish three techniques for building. The technique you choose depends on the dynamics to give to the client.

1. Static proxy

   We use the generated proxy, which contains also a preconfigured reference to the Web Service endpoint. We still can change it in the Java program.

2. Dynamic proxy

   The service locator is used, to change the service endpoint location at runtime

3. Dynamic Invocation Interface

   Those clients do not use pre generated proxy code, but use directly info from a WSDL, which eventually is acquired at runtime.

More info about this can be found in *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461.

We decided here to keep it simple and to write a Java standalone client using a static proxy, so that the endpoint URL can be changed.

The code **ws91.CQUERYClientStatic** located in project IMSClient is listed in Example 8-11 on page 256.

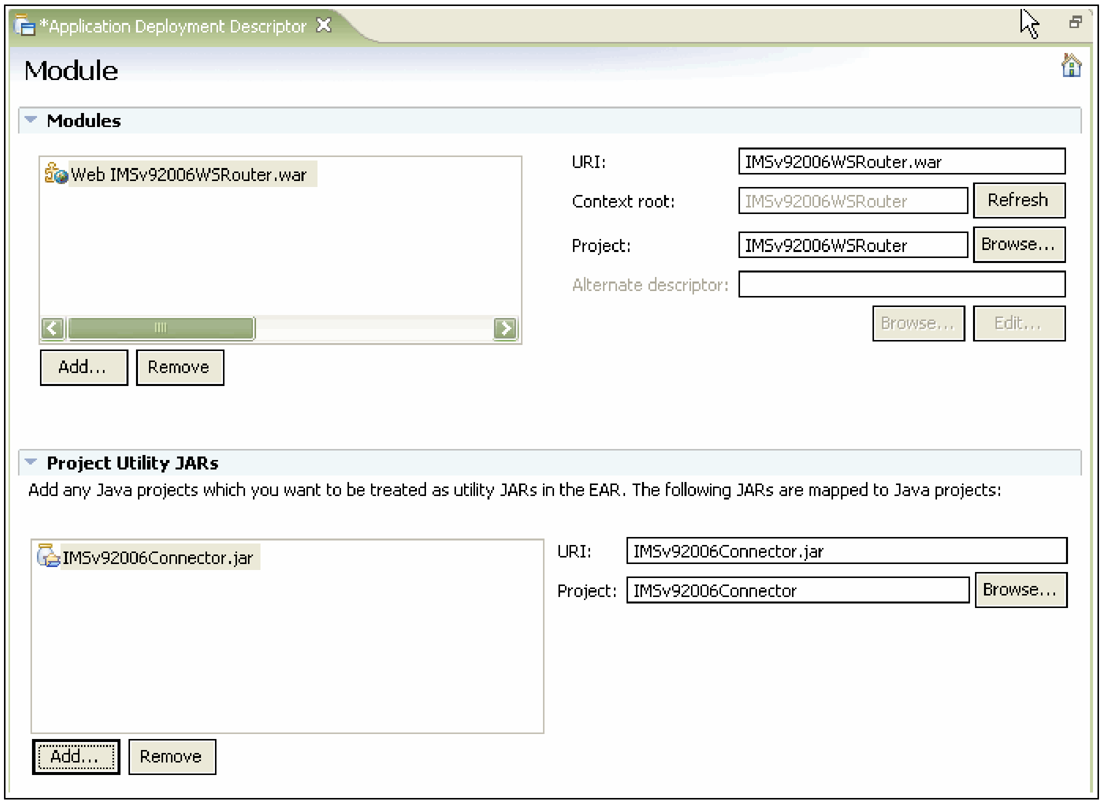
Example 8-11 Generated code with static proxy

```
package ws91;
import cust.*;
public class CQUERYClientStatic {
   public static void main(String[] args) {
        String ipaddr = "127.0.0.1";
        String iport = "9080";
        String custnr = "006668";
      try {
         //  Populate the input data
         INDATA input = new INDATA();
         input.setLlin((short) 23);
         input.setZ1("");
         input.setTran("CQUERY");
         input.setCustnr(custnr);
         input.setTrace("Y");
         //  Invoke the service
         CustQueryImplProxy cqueryproxy = new CustQueryImplProxy();
           String newendpoint = "http://" + ipaddr + ":" + iport
           + "/IMSv92006WSRouter/services/CustQueryImpl";
           cqueryproxy.setEndpoint(newendpoint);
         OUTDATA output = cqueryproxy.cquery(input);
         //  Display the output data
         System.out.println("Customer: " + output.getCustnr());
         System.out.println("Name: " + output.getSalutat() + " "
               + output.getFirstnme() + " " + output.getMi() + " "
               + output.getLastname());
         System.out.println("Address: " + output.getAddress1() + " "
               + output.getAddress2());
         System.out.println("City " + output.getZipcd() + " "
               + output.getCity());
         System.out.println("Country/State: " + output.getState());
         System.out.println("Phone: " + output.getPhone());
         System.out.println("Fax: " + output.getFax());
         System.out.println("EmlAddr: " + output.getEmladdr());
         System.out.println("Message: " + output.getMessage());
      } catch (Exception exception) {
         exception.printStackTrace();
      }
   }
}        System.out.println("Message: " + output.getOut__msg());
         System.out.println("Last name: " + output.getOut__name1());
         System.out.println("First name: " + output.getOut__name2());
         System.out.println("Extension: " + output.getOut__extn());
         System.out.println("Zip: " + output.getOut__zip());
      } catch (Exception exception) {
         exception.printStackTrace();
      }
   }
}
```

This code can be invoked as a Java stand alone, the service_address possibly has to be adapted.

Deploying in WebSphere Application Server

The enterprise application project IMSv92006WSRouterEAR assembles all the artifacts and has to be deployed in WTE in Windows or in WebSphere Application Server on z/OS. This deployment location defines the Web service endpoint address. The content of the EAR that has to be extracted from the RAD Workbench for deployment is shown in Figure 8-63.



Web IMSv92006WSRouter.war

URI: IMSv92006WSRouter.war
Context root: IMSv92006WSRouter
Project: IMSv92006WSRouter
Alternate descriptor:

Add any Java projects which you want to be treated as utility JARs in the EAR. The following JARs are mapped to Java projects:

IMSv92006Connector.jar

URI: IMSv92006Connector.jar
Project: IMSv92006Connector

Figure 8-63 Modules in IMSv92006WSRouterEAR with Web services

Explanation, how to deploy in a WebSphere environment, is beyond the scope of this chapter. After deployment the Web Service can be tested from the client.

Conclusion

With the RAD wizards we have built an HTTP Web Service for an existing IMS transaction. The service endpoint for this service is located in a WebApp router (servlet) in WebSphere Application Server. This service can be used in several ways through clients.

This concludes this example.

8.4.2 A Service built with WebSphere Integration Developer (WID)

Integrating applications on Enterprise Information Systems (EIS) with those already developed locally is a key feature of WebSphere Integration Developer. Examples of EIS systems include IMS, CICS and database systems. In the previous example all components have been prepared with the RAD wizard. Recently IBM introduced a new evolution in the way services can be bound into a SOA. This new architecture, System Component Architecture (SCA) requires the WebSphere Process Server, which is an extension of WebSphere Application Server. On the development side a RAD extension, WebSphere Integration Developer (WID) provides the development ingredients for importing existing services. In Figure 8-64 on page 258, we see an **import** component at work. The **import** component, created by the **enterprise service discovery** wizard in WID represents an application in IMS. The **import** component looks like a local service. Appropriate resource adapters and binding information provide the means of interaction between the **import** and the applications on the EIS systems.

A component exposes business-level interfaces to its application business logic so that the service can be used or invoked. The interface of a component defines the operations that can be called and the data that is passed, such as input arguments, returned values, and exceptions. An import also has interfaces so that the published service can be invoked. The SCA defines several types of components, this is NOT the scope of this book. We discuss here one particular type **import** that makes EIS service available as a component to be used later in a process assembly.

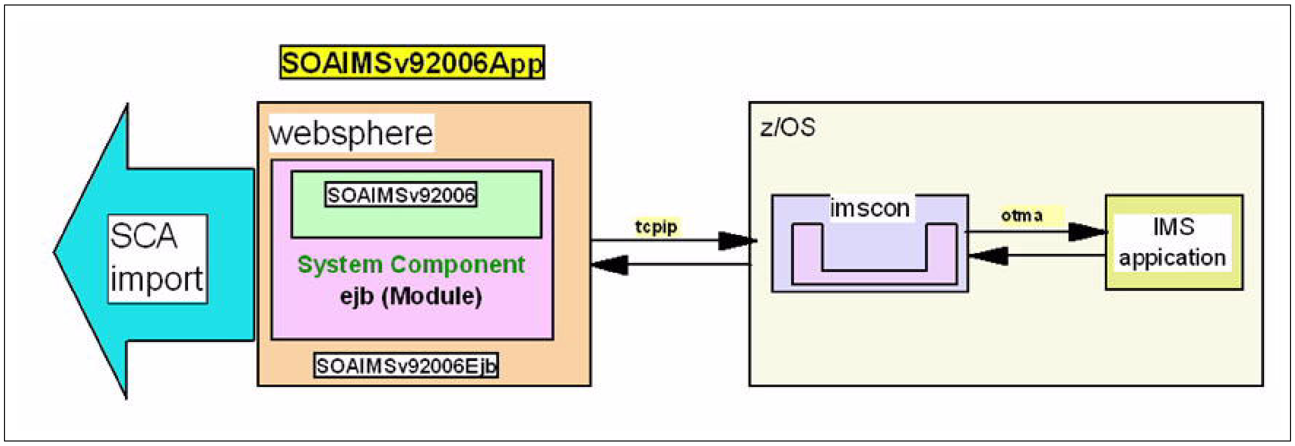


Figure 8-64 SOA built with enterprise service discovery

You create EIS *import* components and their interfaces with the enterprise data discovery and the enterprise service discovery wizards. The interface of the import uses inputs and outputs which are *business objects* (BO), also called *System Data objects* (SDO).You create business objects from data structures with the enterprise data discovery wizard. Business objects (BO) are the input and output structures that go with a particular interface. Interfaces an Business objects could be transformed, mapped by interface and data mapping components, also part of SCA.

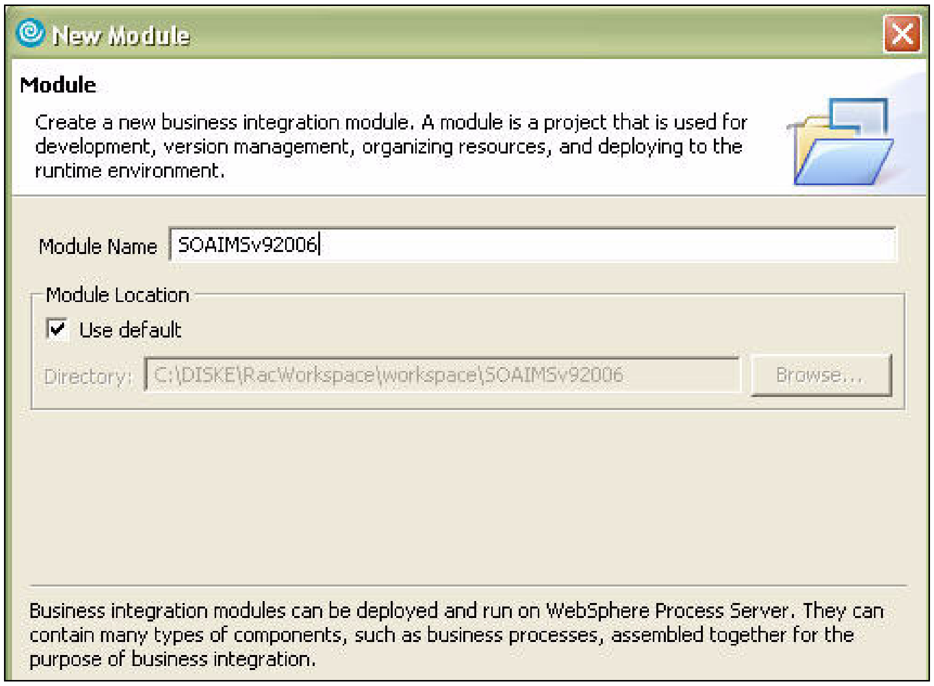
Using the enterprise service discovery wizard is similar no matter what resource adapter you use and EIS system you access.

Preparation of the Connector Business Integration module

In this module, which is the equivalent to a project, in the RAD workbench, we will start building the **import** component, but first we have to prepare the project(module). Assuming that RAD (WID) is started on Windows, and that currently we are in a Business Integration perspective, in the left upper corner of the RAD window, in the menu bar, click **File** → **New** → **Module.**

This starts the create of a new business integration module.

Click **Next.**



Module

Create a new business integration module. A module is a project that is used for development, version management, organizing resources, and deploying to the runtime environment.

Module Name: SOAIMSv92006

Module Location

☑ Use default

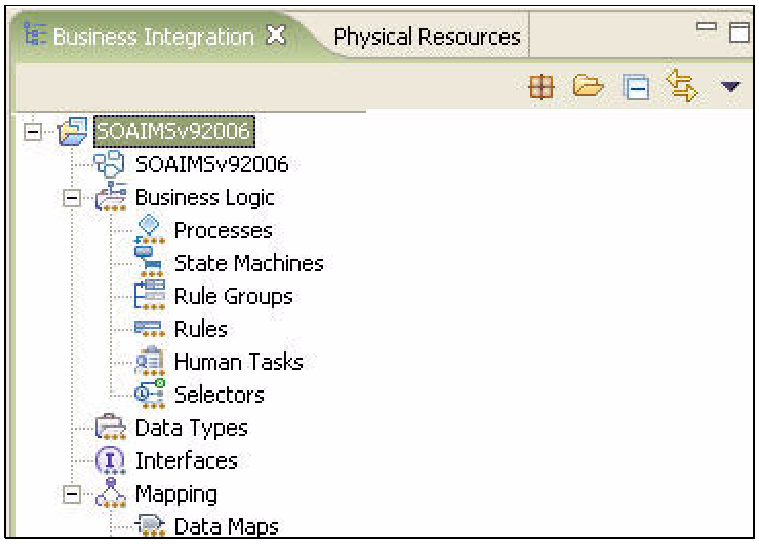
Directory: C:\DISKE\RacWorkspace\workspace\SOAIMSv92006

Business integration modules can be deployed and run on WebSphere Process Server. They can contain many types of components, such as business processes, assembled together for the purpose of business integration.

Figure 8-65 Business Module creation

Give the new module a name, we propose **SOAIMSv92006** as shown in Figure 8-65. Click **Finish**. Note that this module will have to run in a **WebSphere Process Server** container, WebSphere Application Server (WAS) is *not* sufficient.

The project is created, Figure 8-66 shows the layout of the new empty Business module. The module is the basic unit of deployment in the WebSphere Process Server container.



```
SOAIMSv92006
  SOAIMSv92006
  Business Logic
    Processes
    State Machines
    Rule Groups
    Rules
    Human Tasks
    Selectors
  Data Types
  Interfaces
  Mapping
    Data Maps
```

Figure 8-66 Business Module SOAIMSv92006

The names that you discover in this module layout belong to the **WPS domain**. They give an idea of the different elements that can be created as part of a module.

While in the WebSphere Integration Developer (WID), the tooling, which is used, is different from the one used in RAD.

We distinguish two steps:

1. Build of the input/output Data Objects
2. Build of the SCA component, which uses the previous generated Data Objects

The application that we will use, Customer query, is the same as before.

Creating the data objects for input/output of the IMS transaction

We first copy the c include(inoutcust4rad..h) in the module(project); see Figure 8-67.

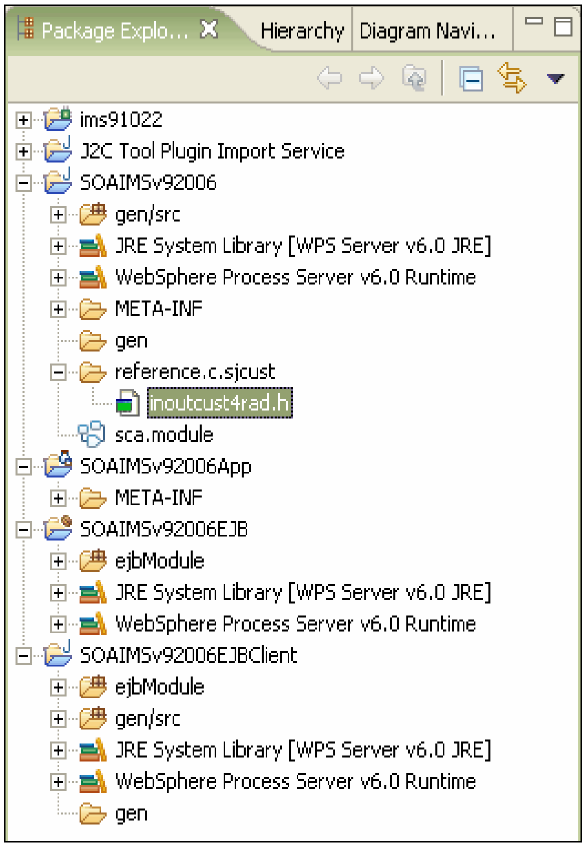
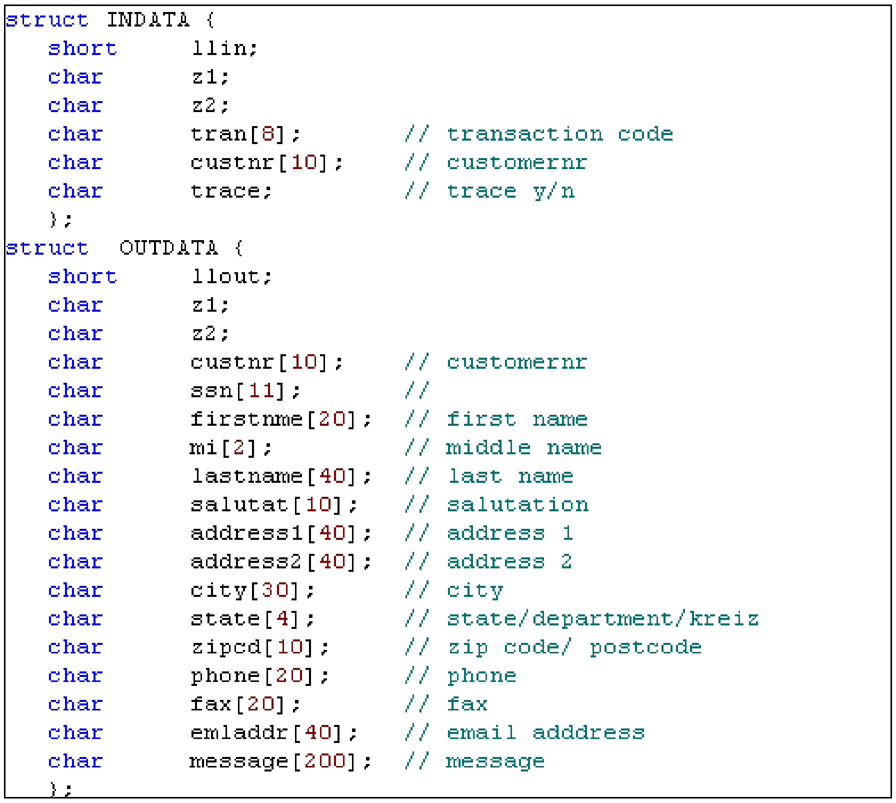

Package Explo...
Hierarchy
Diagram Navi...
ims91022
J2C Tool Plugin Import Service
SOAIMSv92006
gen/src
JRE System Library [WPS Server v6.0 JRE]
WebSphere Process Server v6.0 Runtime
META-INF
gen
reference.c.sjcust
inoutcust4rad.h
sca.module
SOAIMSv92006App
META-INF
SOAIMSv92006EJB
ejbModule
JRE System Library [WPS Server v6.0 JRE]
WebSphere Process Server v6.0 Runtime
SOAIMSv92006EJBClient
ejbModule
gen/src
JRE System Library [WPS Server v6.0 JRE]
WebSphere Process Server v6.0 Runtime
gen

Figure 8-67 Module with copy of inoutcust4rad.h

Let us look at this “h” file; Figure 8-68 on page 261.



```
struct INDATA {
   short      llin;
   char       z1;
   char       z2;
   char       tran[8];        // transaction code
   char       custnr[10];     // customernr
   char       trace;          // trace y/n
   };
struct  OUTDATA {
   short      llout;
   char       z1;
   char       z2;
   char       custnr[10];     // customernr
   char       ssn[11];        //
   char       firstnme[20];   // first name
   char       mi[2];          // middle name
   char       lastname[40];   // last name
   char       salutat[10];    // salutation
   char       address1[40];   // address 1
   char       address2[40];   // address 2
   char       city[30];       // city
   char       state[4];       // state/department/kreiz
   char       zipcd[10];      // zip code/ postcode
   char       phone[20];      // phone
   char       fax[20];        // fax
   char       emladdr[40];    // email adddress
   char       message[200];   // message
   };
```

Figure 8-68   C include (h file)

Using the **Enterprise Data Discovery** wizard, we will create the business objects.

To start the wizard, select the module, right-click → **New** → **Enterprise Data Discovery**.

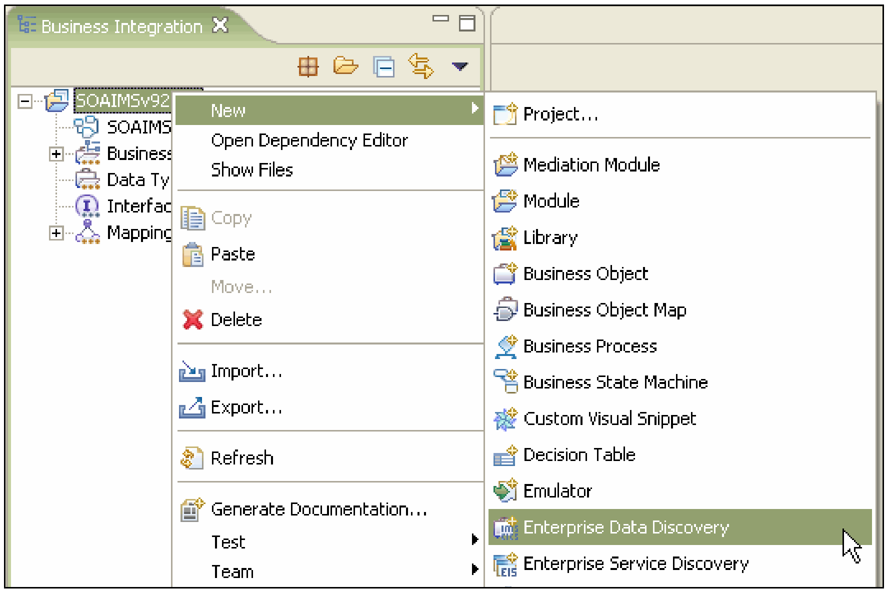


Figure 8-69   Starting the Enterprise Data Discovery wizard

The next pop-up window is a request for data import. Select **C to Business Object**.

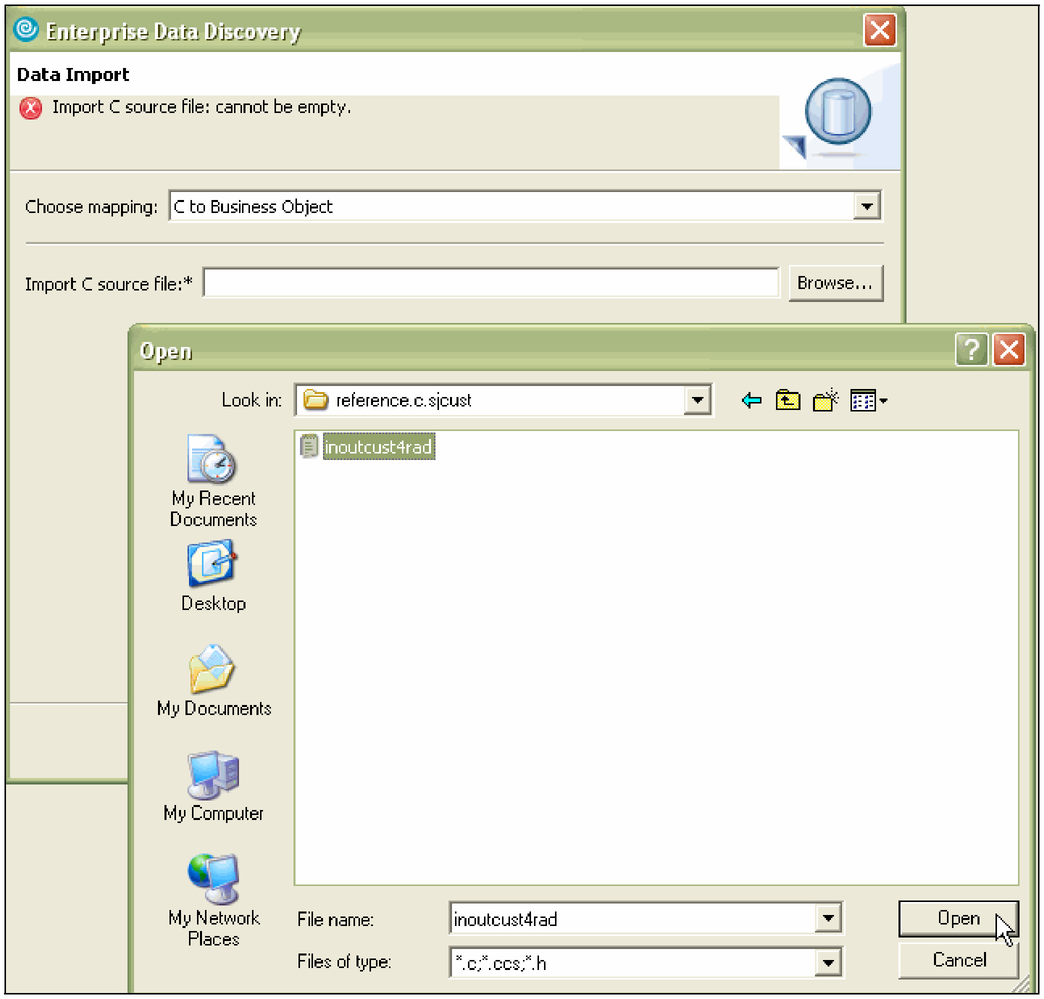


Figure 8-70 Select of the C include file

With the **Browse** option navigate to the inoutcust4rad.h file, Figure 8-70, click **Open.**

Click **Next**.

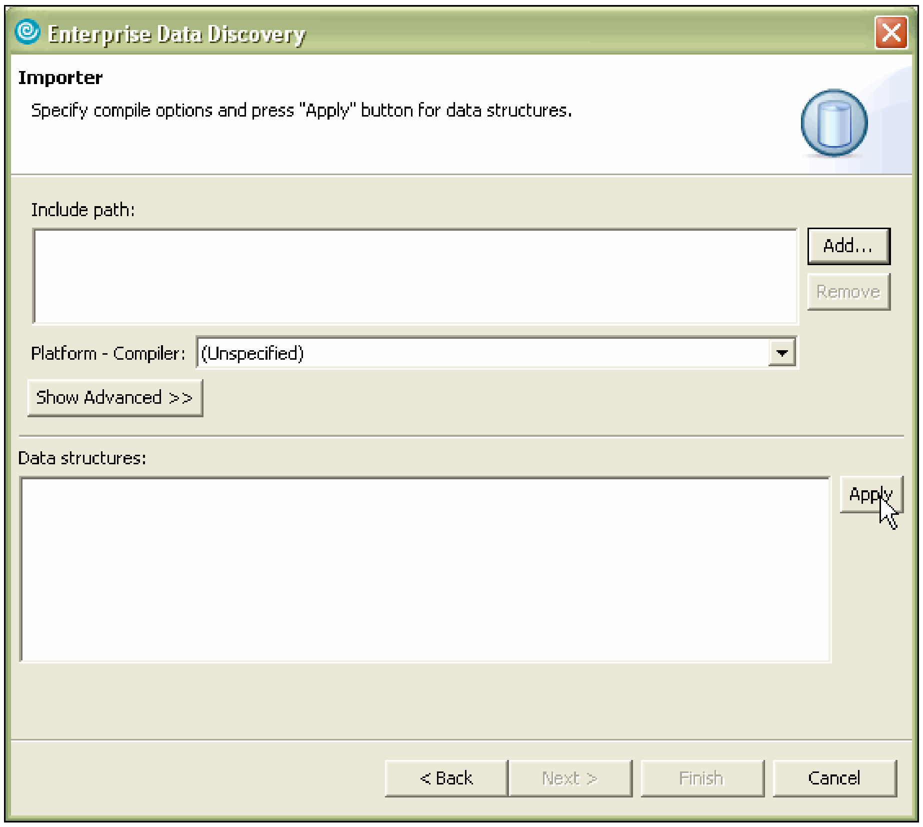


Figure 8-71 Initial Data Discovery panel

Click the option **Apply** as show in Figure 8-71 to see the available structures in the C include file.

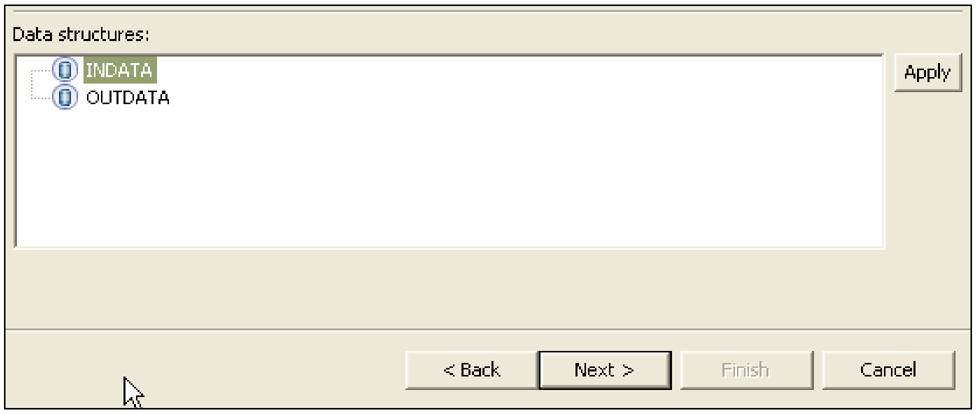
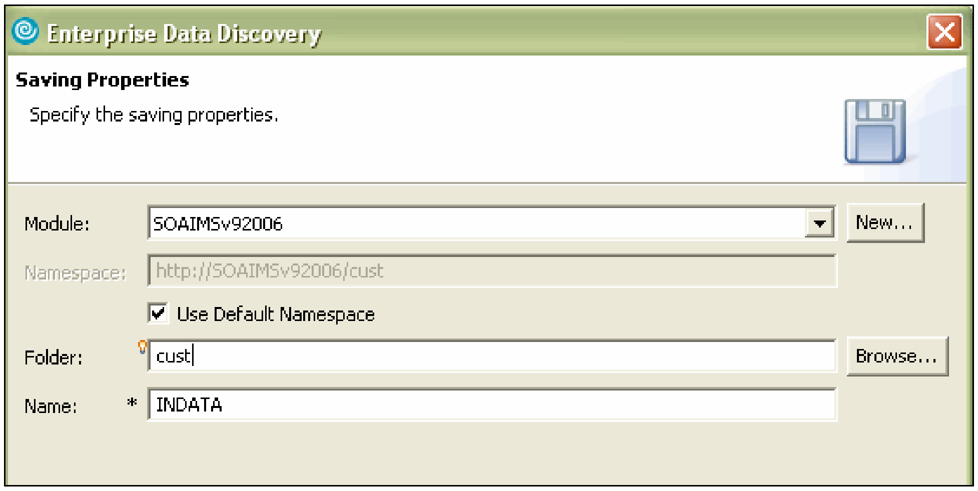


Figure 8-72 Contents of the C include

Select INDATA. This is the layout of the input message; see Figure 8-72. Click **Next**.


Enterprise Data Discovery

Saving Properties

Specify the saving properties.

Module: SOAIMSv92006

Namespace: http://SOAIMSv92006/cust

[✓] Use Default Namespace

Folder: cust

Name: * INDATA

Figure 8-73 Select input structure and folder(package)

Complete the screen information with the folder name, **cust**; see Figure 8-73.

Click **Finish**, the data binding is now generated.

Repeat the previous **wizard cycle** for the output message with the OUTDATA structure. The new look of the business integration module is shown in the following Figure 8-74.

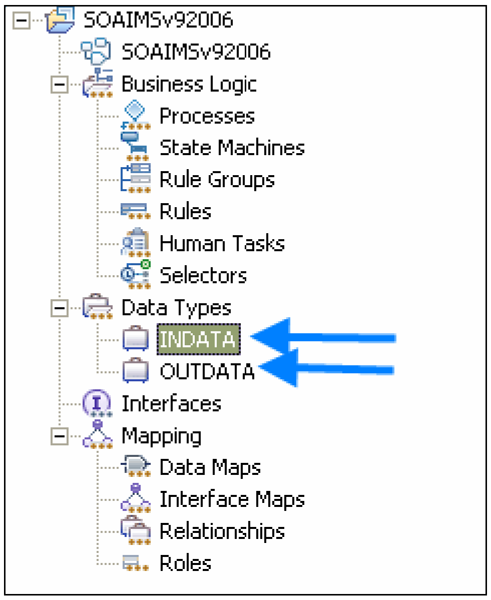


Figure 8-74 Look of module after preparation of Data Objects

You recognize the new generated Data Types for input and output. Both are XML schema files (XSD), describing the data objects.

Creating an import for the IMS transaction

Using the **enterprise service discovery** wizard, we create a **SCA component** of type **import,** based on an existing IMS application. We will use the datatypes, generated in the previous step, as input and output.

To start the wizard, select the module, right-click → **New** → **Enterprise Service Discovery.** See Figure 8-75.

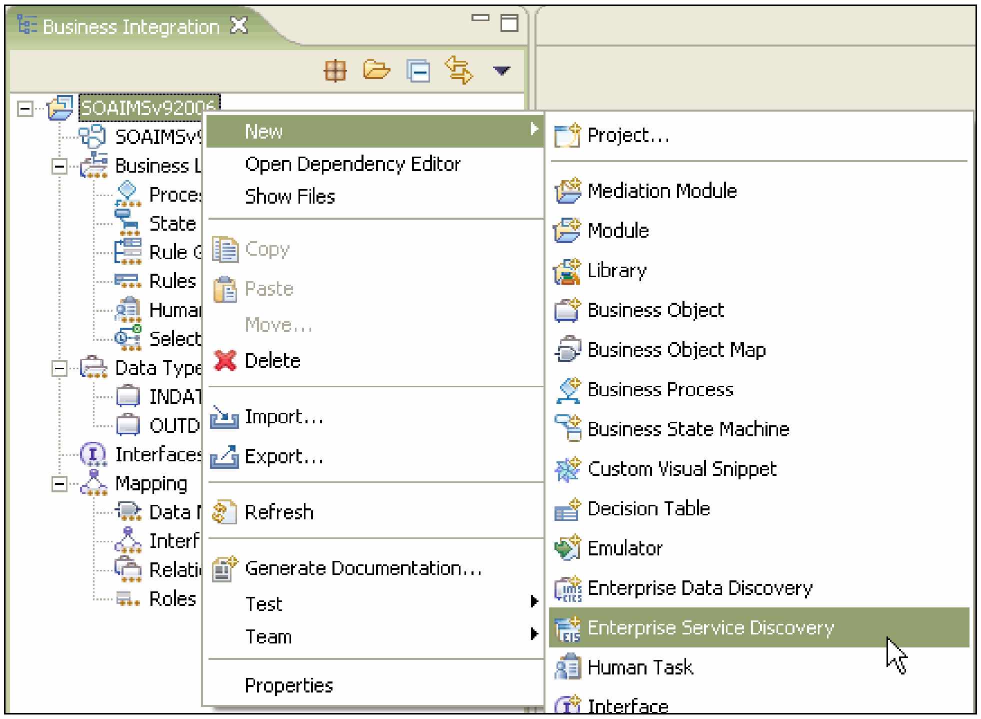
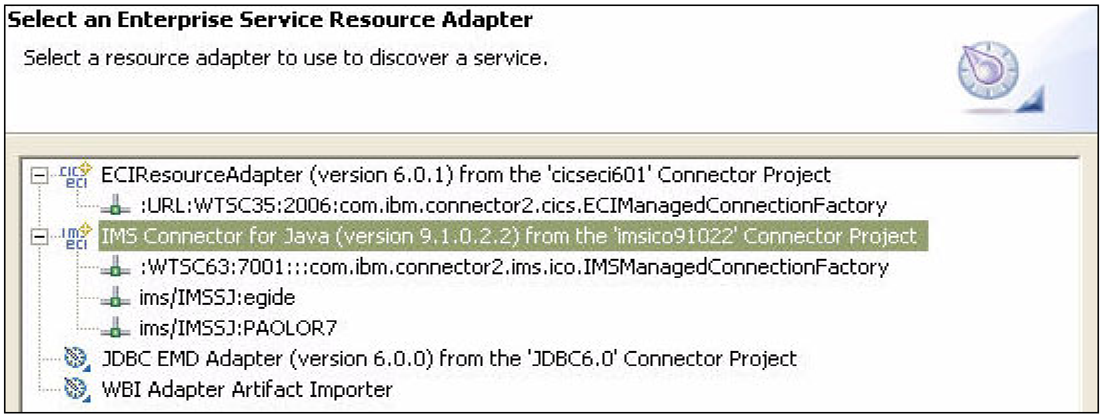


Figure 8-75 Starting the Enterprise Service Discovery

The panel of Figure 8-76 is presented with a list of installed resource adapters. Here you indicate for which resource adapter you create this **import**.



Select an Enterprise Service Resource Adapter

Select a resource adapter to use to discover a service.

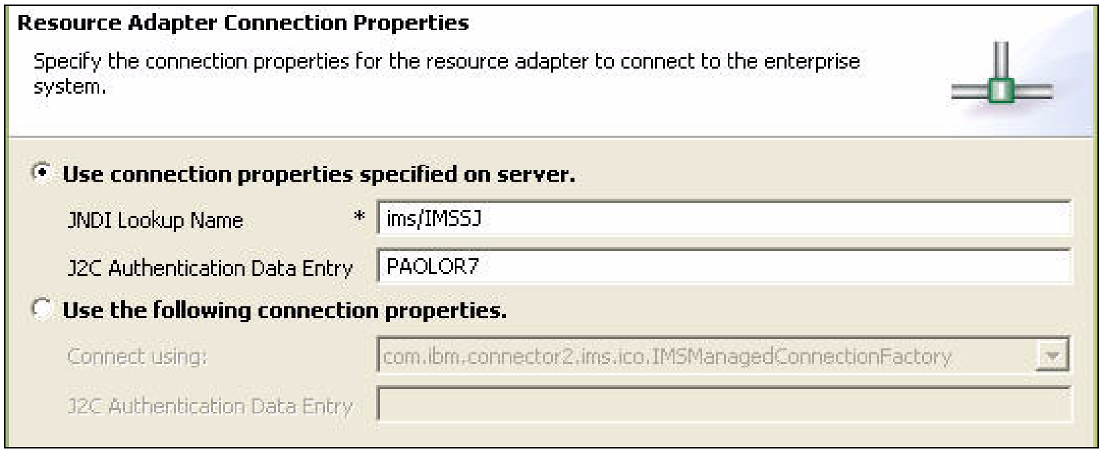
- ECIResourceAdapter (version 6.0.1) from the 'cicseci601' Connector Project
  - :URL:WTSC35:2006:com.ibm.connector2.cics.ECIManagedConnectionFactory
- IMS Connector for Java (version 9.1.0.2.2) from the 'imsico91022' Connector Project
  - :WTSC63:7001:::com.ibm.connector2.ims.ico.IMSManagedConnectionFactory
  - ims/IMSSJ:egide
  - ims/IMSSJ:PAOLOR7
- JDBC EMD Adapter (version 6.0.0) from the 'JDBC6.0' Connector Project
- WBI Adapter Artifact Importer

Figure 8-76 Selection of Resource Adapter

Select the IMS Connector for Java as shown in Figure 8-76. This is the adapter distributed with IMSV9, which previously had to be installed in RAD.

Click **Next.**

The next panel, in Figure 8-77 on page 266, asks for the connection properties. These can be specified here, or we can use a connection factory definition inside the WebSphere Process Server, which is retrievable via a JNDI lookup. The latter is the advisable solution, so we have to specify its JNDIname. It is assumed that previously this definition was made via the Admin Console of the WebSphere Test Environment (WTE). In the previous screen capture Figure 8-76 on page 265 you can see that already some connection factories have been defined. Referring to an explanation in the previous chapter, these are **managed** connection definitions.


Resource Adapter Connection Properties

Specify the connection properties for the resource adapter to connect to the enterprise system.

(●) **Use connection properties specified on server.**

JNDI Lookup Name * ims/IMSSJ

J2C Authentication Data Entry PAOLOR7

( ) **Use the following connection properties.**

Connect using: com.ibm.connector2.ims.ico.IMSManagedConnectionFactory

J2C Authentication Data Entry

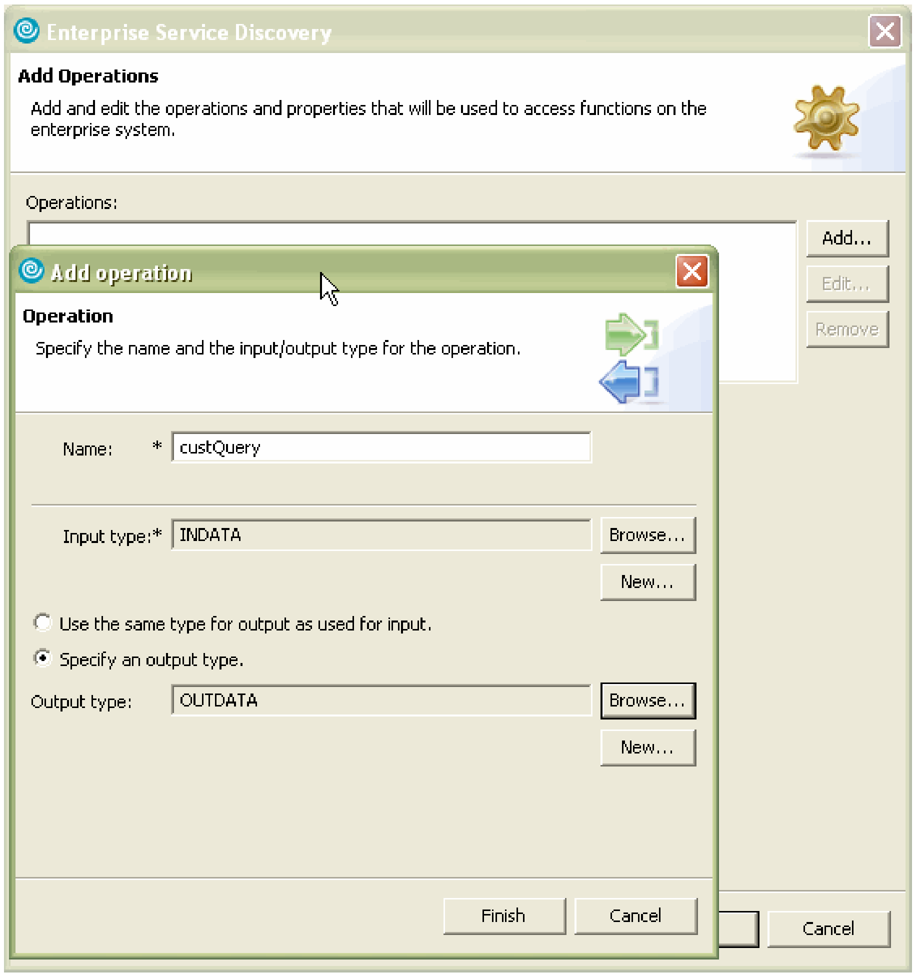
Figure 8-77 Managed Connection reference

Click **Next.**

The **import** component has to be configured with methods, comparable to an API, which will be part of its interface. Basically a method has a name, takes an input message and returns an output message. A method is not linked implicitly to an IMS transaction. This link is established by the transaction code in the contents of the input message.

On the new panel, that comes up, click **Add**, and complete the information in the **Add operation** pop-up with a methodname, input and output messages. The messages, in System Data Object (SDO) format, do exist from the data discovery phase and are used. Specify as methodname **custQuery**.

The wizards, see Figure 8-78 on page 267, will guide you through several self-explanatory steps, in which you will browse to the generated input/output datatypes and specify them as input and output.



Enterprise Service Discovery

Add Operations

Add and edit the operations and properties that will be used to access functions on the enterprise system.

Operations:

Add operation

Operation

Specify the name and the input/output type for the operation.

Name: * custQuery

Input type:* INDATA

( ) Use the same type for output as used for input.

(●) Specify an output type.

Output type: OUTDATA

Figure 8-78 Adding operation and importing C to business objects

Click **Finish.**

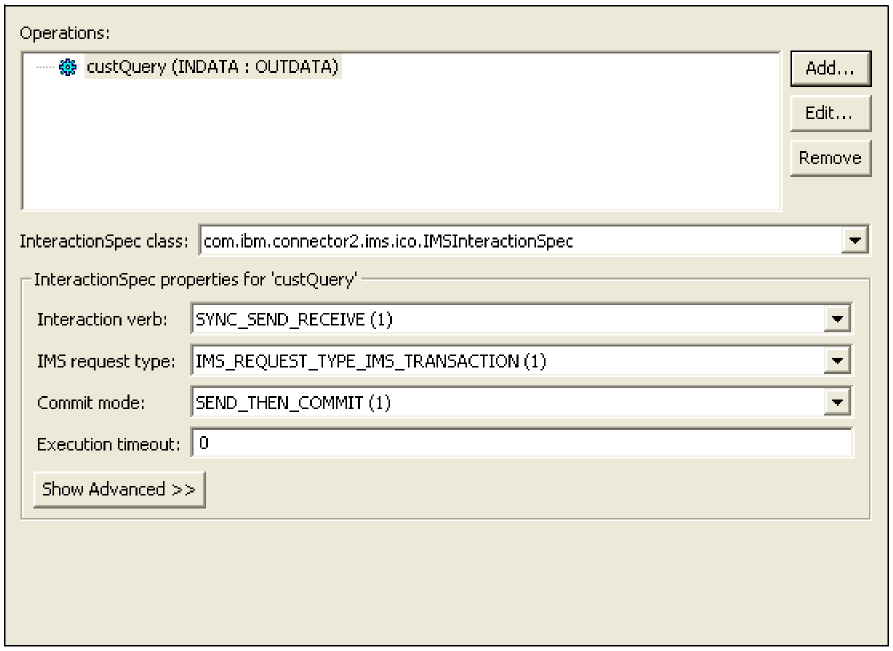
Most of the definitions for the **service discovery** have been done, we specified:

- Connection properties to IMS via a managed definition
- Operation description with Input and Output

The next element to specify are the characteristics of the IMS Interaction related to this operation.

Accept the proposed values in Figure 8-79 on page 268.

For more explanation on the meaning of the accepted default values, see *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794.


Operations:

custQuery (INDATA : OUTDATA)

InteractionSpec class: com.ibm.connector2.ims.ico.IMSInteractionSpec

InteractionSpec properties for 'custQuery'

Interaction verb: SYNC_SEND_RECEIVE (1)

IMS request type: IMS_REQUEST_TYPE_IMS_TRANSACTION (1)

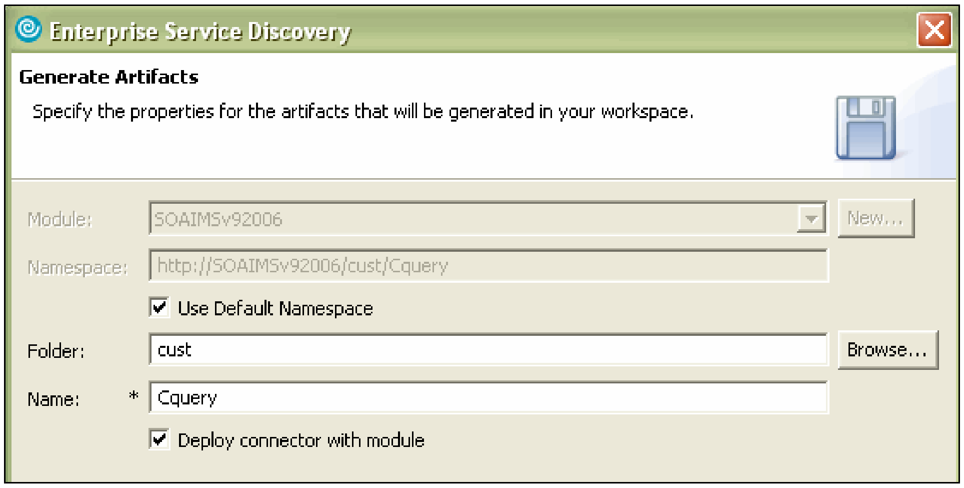
Commit mode: SEND_THEN_COMMIT (1)

Execution timeout: 0

Figure 8-79 Operation Interaction specification

Click **Next.**

A final request for the name of the import component and package appears in the panel of Figure 8-80.



Generate Artifacts

Specify the properties for the artifacts that will be generated in your workspace.

Module: SOAIMSv92006

Namespace: http://SOAIMSv92006/cust/Cquery

☑ Use Default Namespace

Folder: cust

Name: * Cquery

☑ Deploy connector with module

Figure 8-80 Generation of artifacts

Fill in a name as indicated and click **Finish**.

In Figure 8-81 on page 269, find the new look of the SCA module in the Business Integration perspective.

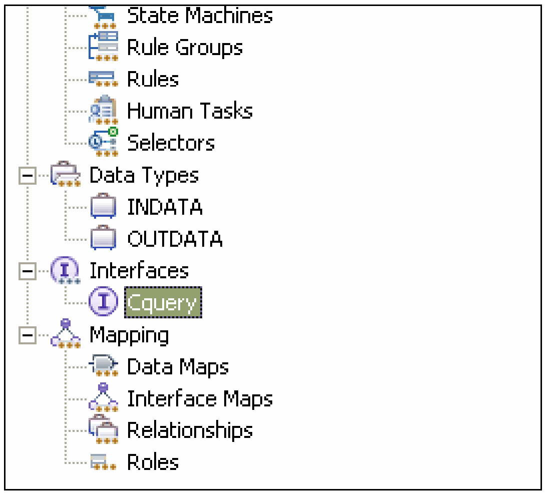


Figure 8-81 Final SOAIMSv92006 Business Module

Notice the following elements in the module:

- component: Cquery
- interface: Cquery
- datatypes: INDATA, OUTDATA

The basic elements for an import are ready, but have to be used in the SCA import component. This happens with the graphical **Assembly** capability of WID.

Assembling the System Component

Click twice on SOAIMSv92006, under the modulename which is also SOAIMSv92006.

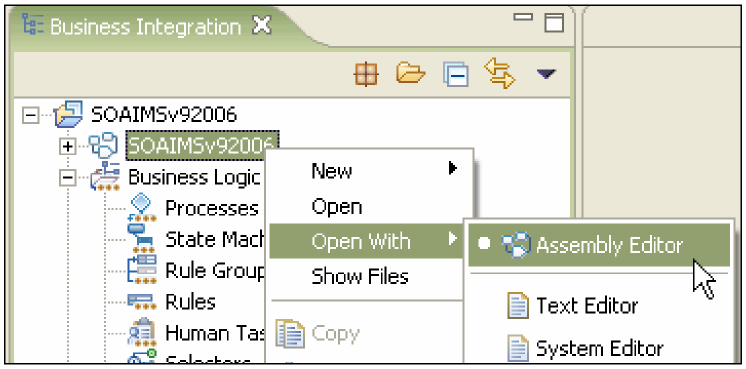
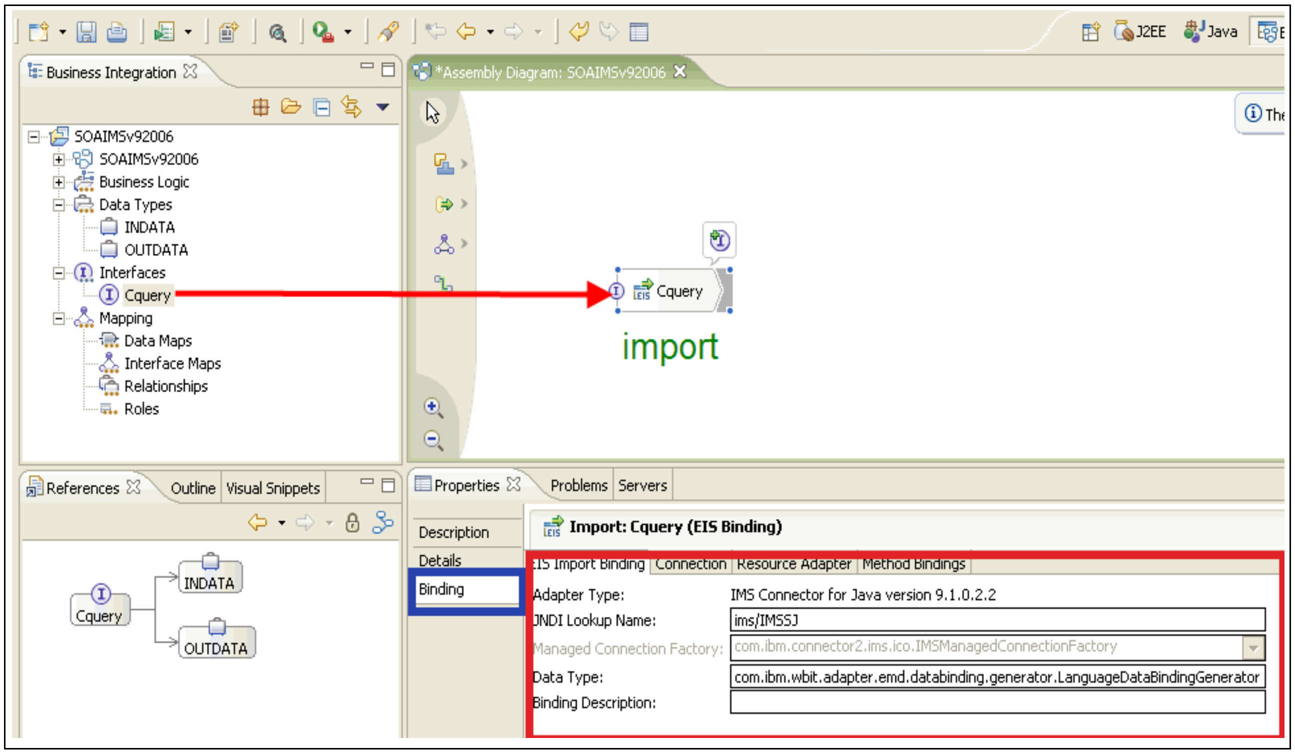


Figure 8-82 Open of Assembly editor

This opens the assembly diagram editor of the module in the right pane. The assembly editor is the primary tool for composing an SCA based application. The module assembly contains a diagram of the integrated business application, consisting of components and the wires that connect them. You use an assembly editor to visually compose the integrated application with elements that you drag/drop from the palette or from the tree in the Business Integration view.

The implementations of components that are used in a module assembly might reside within the module. Components that belong to other modules can be used through imports.

Notice that there is already a component on the board, this is the **import**, that was composed with by the WID wizard. See Figure 8-83 on page 270.



Import: Cquery (EIS Binding)

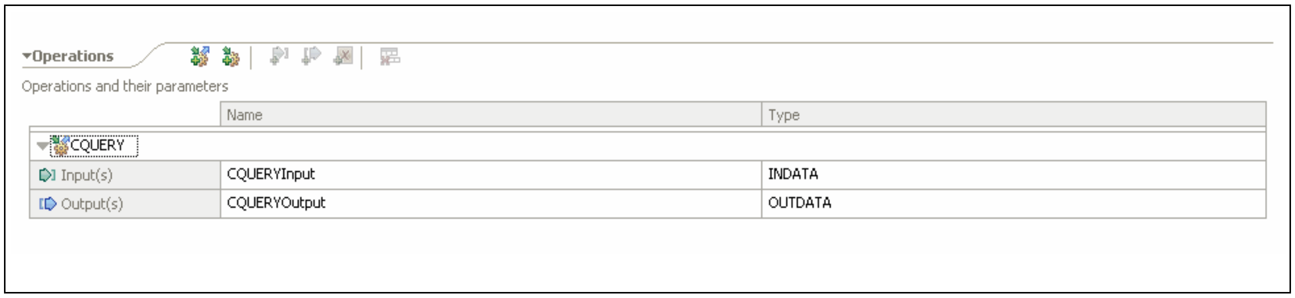
EIS Import Binding | Connection | Resource Adapter | Method Bindings

Adapter Type:	IMS Connector for Java version 9.1.0.2.2
JNDI Lookup Name:	ims/IMSSJ
Managed Connection Factory:	com.ibm.connector2.ims.ico.IMSManagedConnectionFactory
Data Type:	com.ibm.wbit.adapter.emd.databinding.generator.LanguageDataBindingGenerator
Binding Description:	

Figure 8-83 Assembly editor for the Business Module

This **import** has already an interface and also a binding. The binding goes directly to the IMS connectionfactory, combined with a lookup for ims/IMSSJ information, which is the JNDIname for the lookup of the managed definition, available in the resource definitions of the WebSphere Test Environment.

By clicking twice on the interface Cquery you see how this interface is composed. Figure 8-84, shows the interface, operation name and input, output parameters with types.



	Name	Type
CQUERY		
Input(s)	CQUERYInput	INDATA
Output(s)	CQUERYOutput	OUTDATA

Figure 8-84 Interface with 1 operation

This component of type **import** is a real SCA component, which can be used in several ways.

See Figure 8-85 on page 271.

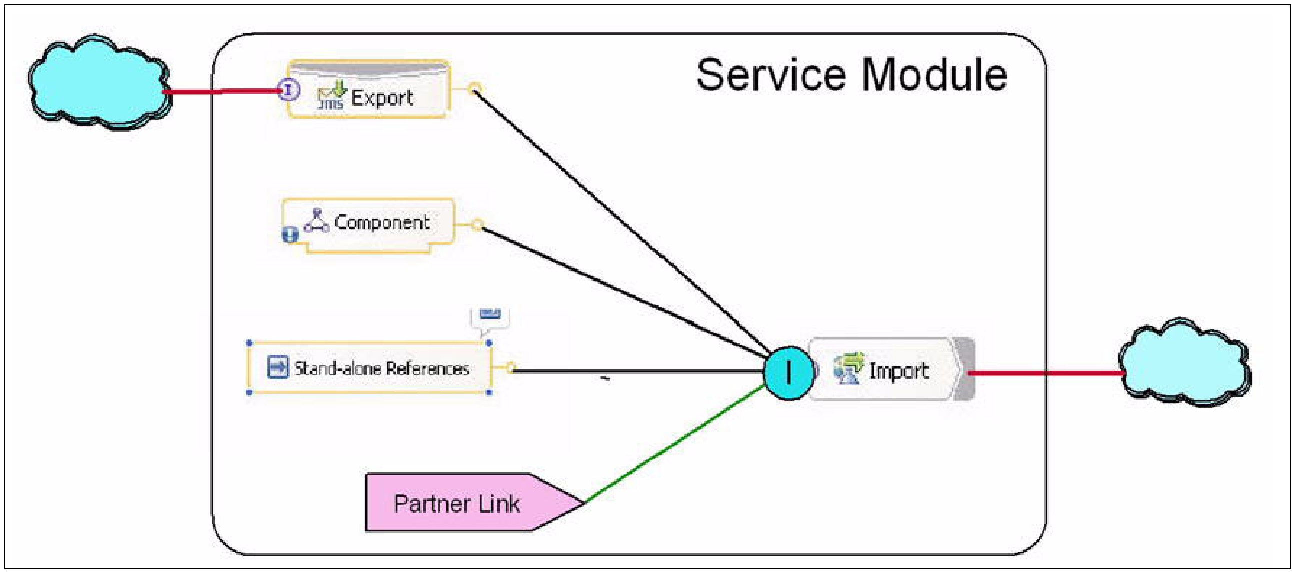


Figure 8-85 Using an import component

- An **export** component, for example, a listener on a JMS queue, could wire to the import, with or without an intermediate interface/data mapping.
- A component of other types can always wire to the import.
- The import could be used as such from an other artifact, for example, a servlet by using a standalone reference.
- It could be used as a partner link in a process composition with the BPEL editor in Process Choreography.

The service module is the deployment unit in the WebSphere Process Server. In reality we deploy real J2EE artifacts, combined in an enterprise archive file (EAR). The EARfile, represented by a project in the workbench has been created and embraces also the required artifacts. The best way to discover what is enclosed in the module is to open the deployment descriptor of the EAR project **SOAIMSv92006App**, which is related to the business module. This name was derived from the business module name, by appending **app**. Other projects, with derived names, have been created by the build of the deployment code, see Figure 8-86 on page 272.

- Ejb project

  SOAIMSv92006EJB
- Ejbclient project

  SOAIMSv92006EJBClient

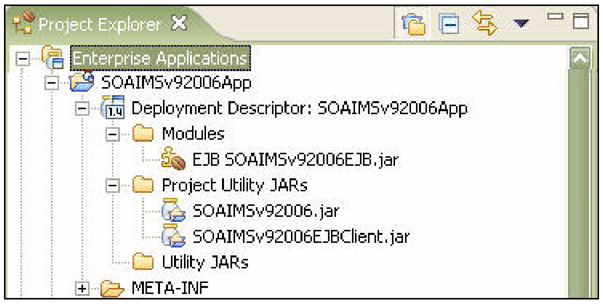


Figure 8-86 SOAIMSv92006App enterprise project

In the Enterprise Applicationproject, see Figure 8-86 we find 1 Ejb project (SOAIMSv92006EJB), and two Project Utility projects (SOAIMSv92006, SOAIMSv92006EJBClient).

The real entry to the System Component is via the EJB module. See Figure 8-87.

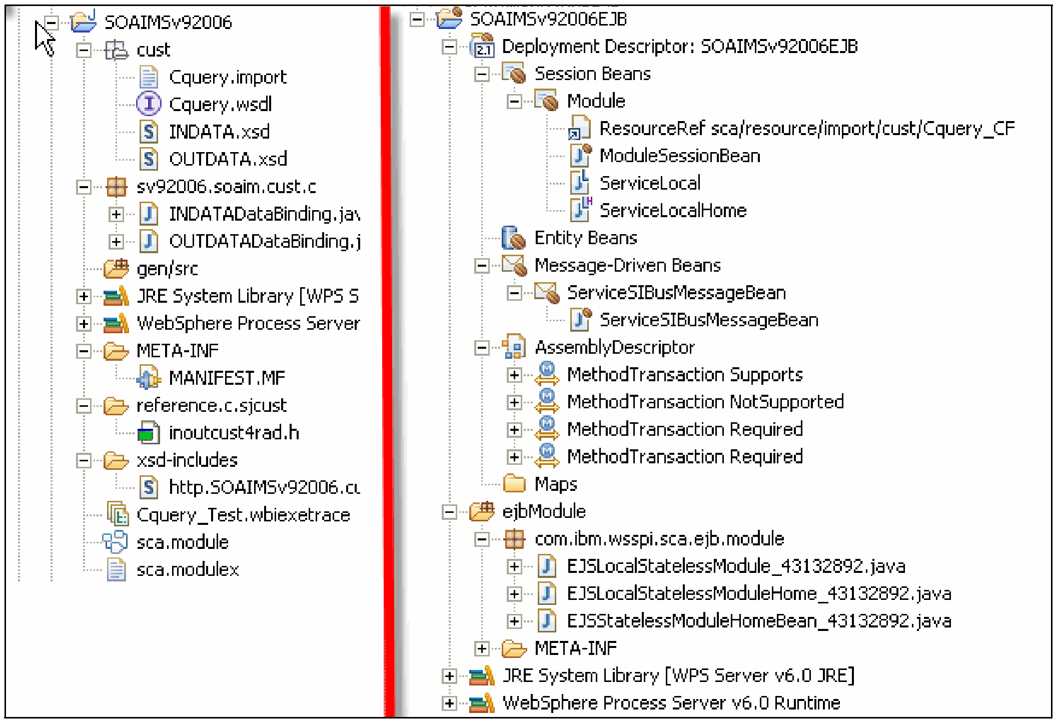


Figure 8-87 SOAIMSv92006, SOAIMSv92006EJB

In the EJB project we find two EJBs. Both represent a access into the component.

- Session EJB: **Module** offers synchronous service entry
- Message Driven EJB: **ServiceSIBusMessageBean** presents an asynchronous service entry via an inyteractionSpec on the ESB.

Using this component in an SCA project, or in a SOA process developed with the BPEL editor is beyond the scope of this chapter. The implementation of a simple standalone reference component is explained in *Approach to Problem Determination in WebSphere Application Server V6*, REDP-4073.

Testing the import component from WID

Testing the service could be one of the responsibilities of the developer. This test can take place from within the Assembly Diagram panel.

Right-click the import component in the assembly diagram → **Test Component.**

See Figure 8-88. During the test you will learn that the WPS test environment comes up; be sure also that the Enterprise Application SOAIMSv92006App got deployed in the server.

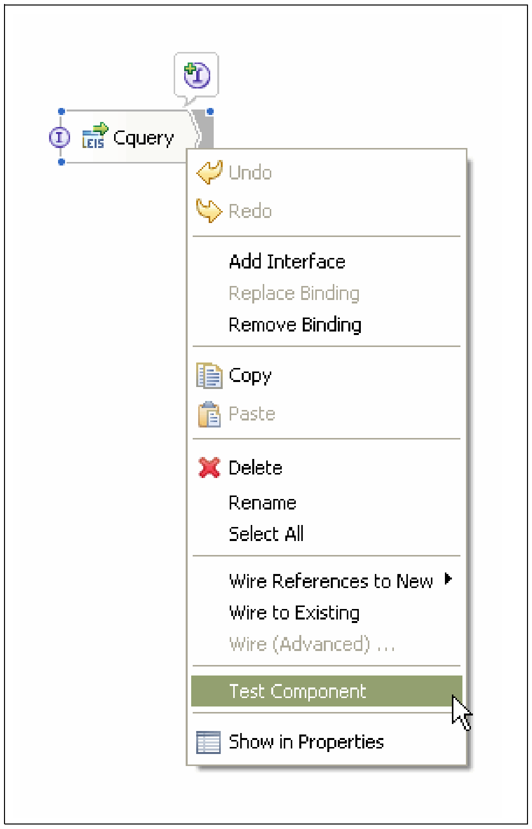
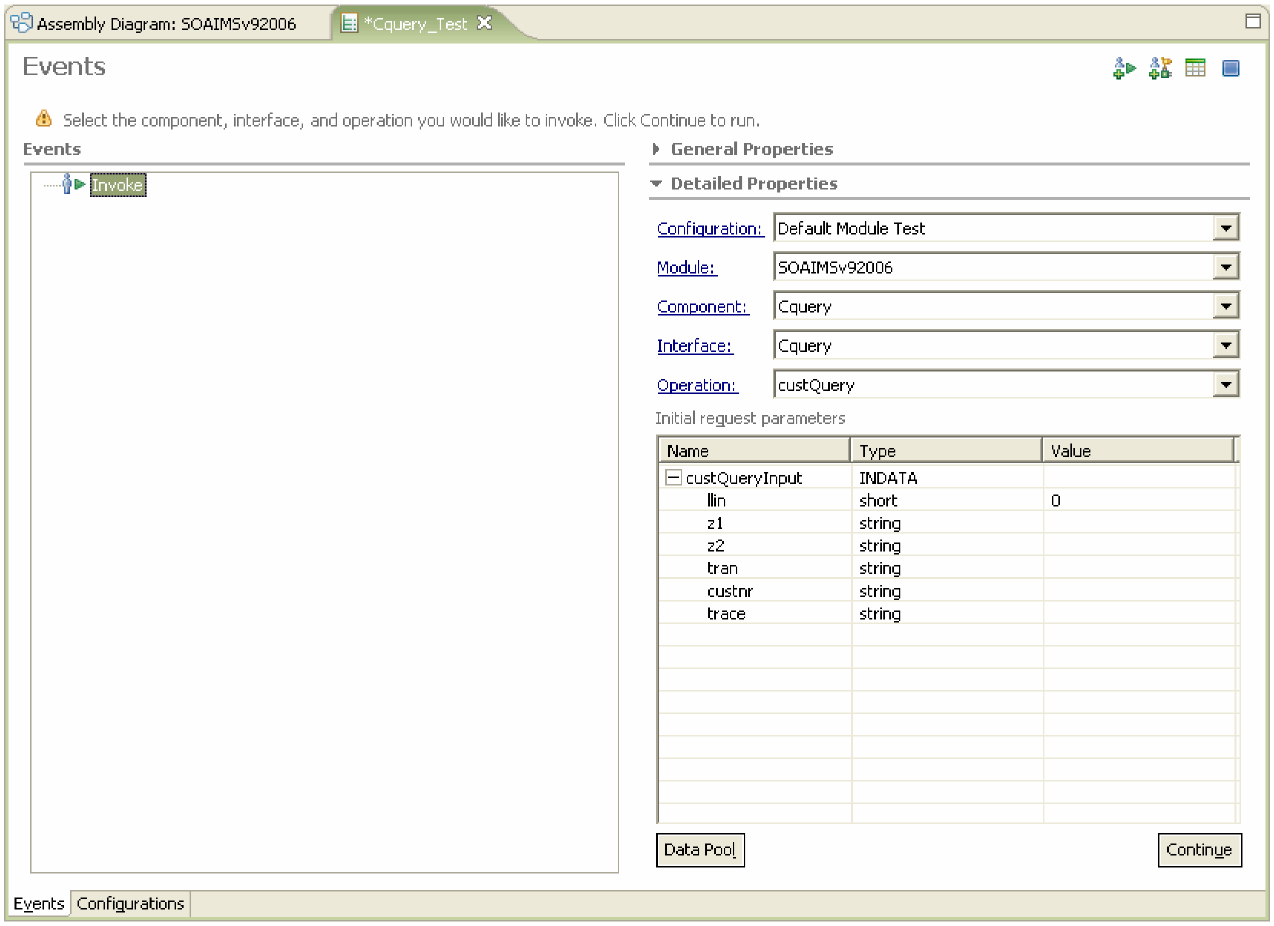


Figure 8-88 Test the CQuery component

The next screen, Figure 8-89 on page 274, allows you to enter parameters, corresponding with the layout of the input message to IMS.



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: SOAIMSv92006
Component: Cquery
Interface: Cquery
Operation: custQuery

Initial request parameters

Name	Type	Value
custQueryInput	INDATA	
llin	short	0
z1	string	
z2	string	
tran	string	
custnr	string	
trace	string	

Figure 8-89 Input for SCA component test

Enter input data as prompted by the layout of the input message and click **Continue** to enter the test.

Conclusion

With a WID wizard we have built a SCA Service Component for an existing IMS transaction. The service endpoints (synchronous, asynchronous) for this service are located in a Ejb project. This service component, which is really SCA compliant, can be used in several ways through clients.

Notice that in this example we used a WID wizard, different from RAD.

This concludes this example.

8.4.3 Importing with WID, a Service built by RAD

In 8.4.1, “HTTP Web Service built with RAD” on page 230, we used RAD to build a Web service for J2C. This is a service, but not really a System Component for SCA.

In 8.4.2, “A Service built with WebSphere Integration Developer (WID)” on page 257, we developed an SCA **import** component, ready to be used by a Process flow via the Process Choreography. This component was completely developed with WID tooling.

In this part we will use a **meet in the middle** approach, using RAD and WID. For most of the artifacts built by RAD we can create imports with the WID toolkit. In Figure 8-90 we show the points at which an **import** could be created.

- In **A**: directly on J2Cbean
- In **B**: on the session EJB, built by the RAD wizard
- In **C**: on the Web services
- In **D:** on the proxy for Web services

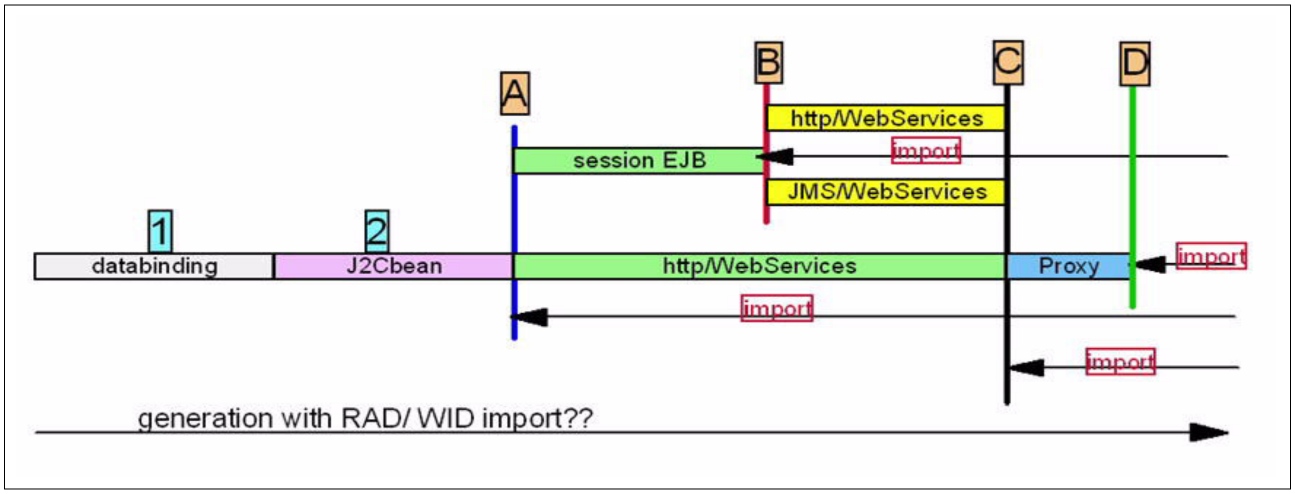


Figure 8-90 Import possibilities by WID

On each of the indicated access points (A, B, C, D) an SCA **import** component could be built, but this requires WID and WPS at runtime.

In all solutions the layers 1 (databinding) and 2 (J2C bean) are common.

We selected to build as a third layer a Session EJB, again with the RAD J2C wizard. The use of an intermediary EJB can have some advantages

- An EJB with remote interface offers remote method invocation on the methods.
- On each method invocation transactional contexts can be considered.

From this layer on, we will build the **import** with WID, using an EJB binding. A Java binding would be possible as well. Look at Figure 8-91 to understand the preparation process.

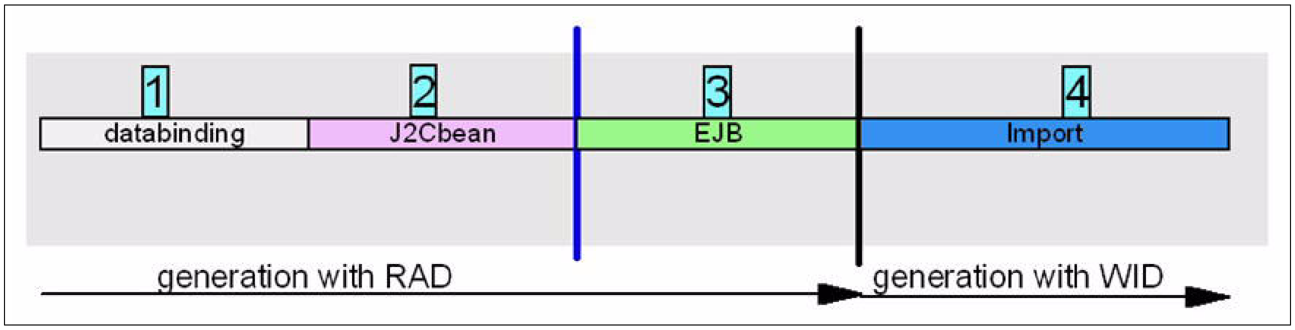


Figure 8-91 RAD, WID solution

The end result is depicted in Figure 8-92 on page 276. In the picture you recognize the layers, and also the RAD/WID tool projects, which are used.

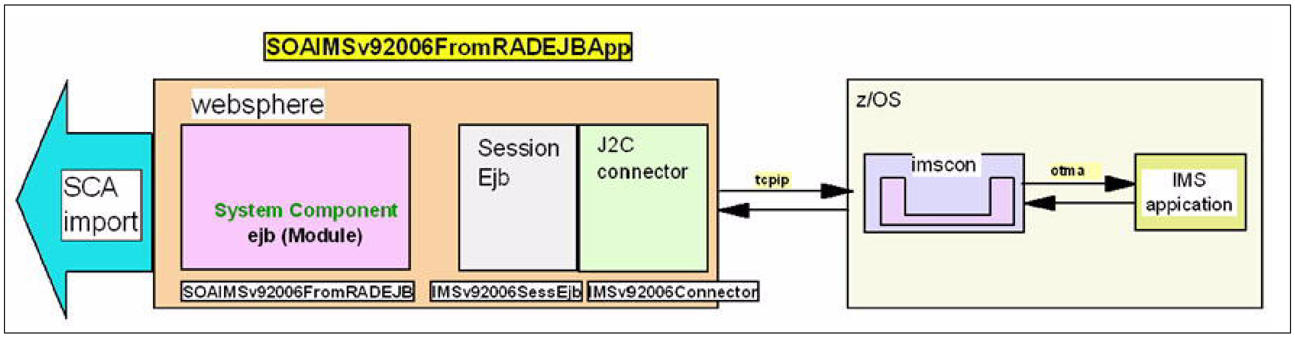


Figure 8-92 SCA import from EJB

The words *import* and *export* in SCA could be very confusing:

- *import*

  From SOA an existing service component is imported (used)

- *export*

  An existing component triggers activity in SOA

For this approach the first thing to do is to build the stateless session EJB on the existing layers 1 and 2.

Session EJBean

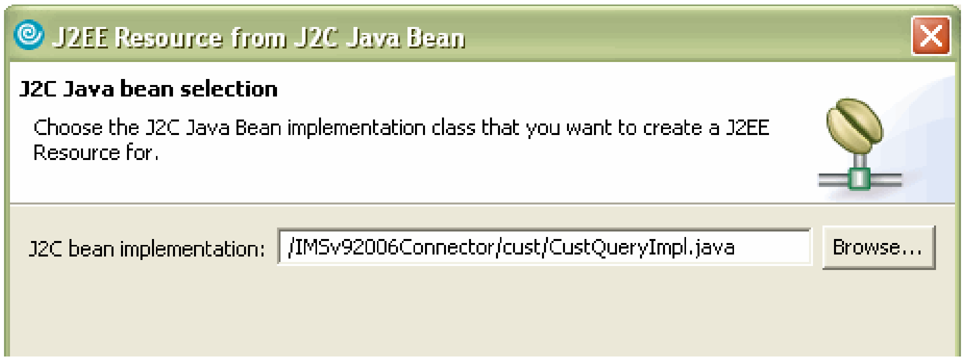
The RAD J2C wizard has an option to generate a Session EJB. It is built upon the J2C bean. A Session EJB is a J2EE artifact that requires a separate EJB project. This will be created by the wizard.

In the previously created connector project **IMSv92006Connector** select the J2CBean **CustQueryImpl.**

Right-click **New** → **Other.**

Select **Web Page, Web Service or EJB from J2C JavaBean.**

A J2C JavaBean selection pop-up is presented, see Figure 8-93. Verify the name of the J2CBean.



J2C Java bean selection

Choose the J2C Java Bean implementation class that you want to create a J2EE Resource for.

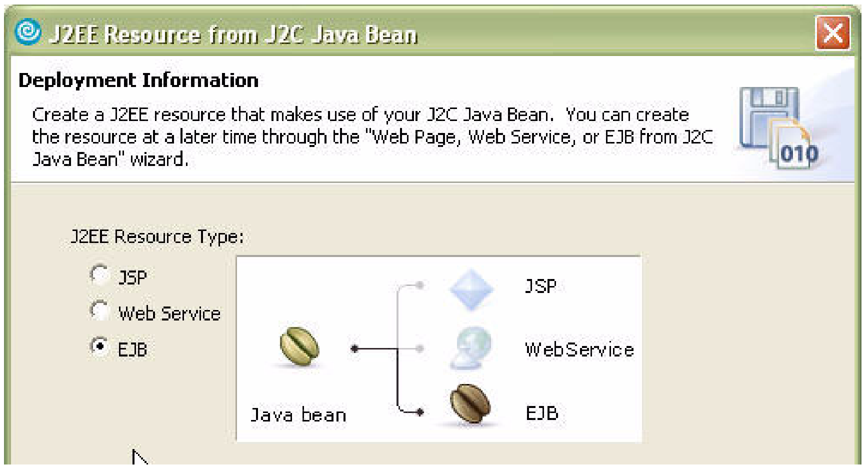
J2C bean implementation: /IMSv92006Connector/cust/CustQueryImpl.java

Figure 8-93 Selected J2C JavaBean

Click **Next.**

A verification takes place that the **managed** connection definition **ims/IMSSJ**, addressed by the JNDIname is present in the server. The server should eventually be started and the information is retrieved by JNDIname.

If alright, the next screen appears. Check **EJB** on the new screen of Figure 8-94.


J2EE Resource from J2C Java Bean

Deployment Information

Create a J2EE resource that makes use of your J2C Java Bean. You can create the resource at a later time through the "Web Page, Web Service, or EJB from J2C Java Bean" wizard.

J2EE Resource Type:

( ) JSP
( ) Web Service
(•) EJB

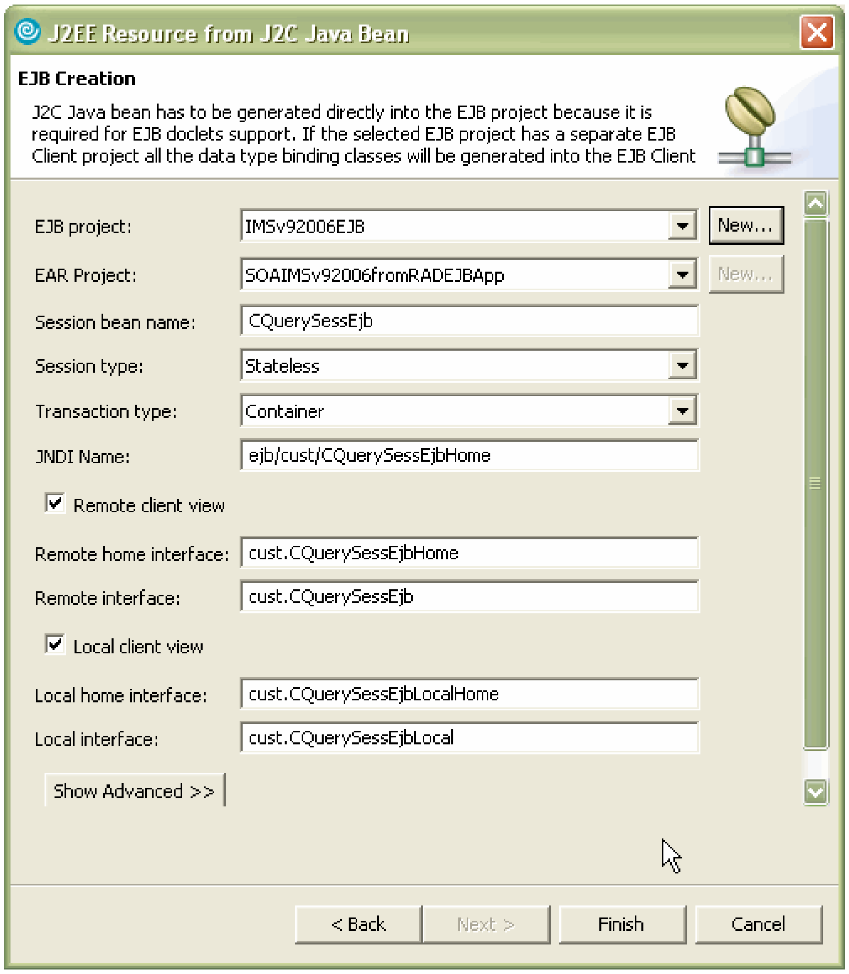
Figure 8-94 Select EJB

Click **Next.**

On the new screen, shown in Figure 8-95 on page 278, the following elements have to be filled in:

- EJB projectname: **IMSv92006EJB**, belongs also to the enterprise project **IMSv92006EAR**
- EJB name: **CQuerySessEjb**

You have the option, besides the remote interface to select also a local interface. All other fields are completed automatically by the wizard.



J2EE Resource from J2C Java Bean

EJB Creation

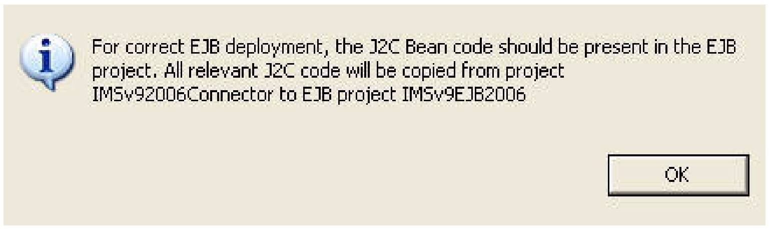
J2C Java bean has to be generated directly into the EJB project because it is required for EJB doclets support. If the selected EJB project has a separate EJB Client project all the data type binding classes will be generated into the EJB Client

EJB project:	IMSv92006EJB
EAR Project:	SOAIMSv92006fromRADEJBApp
Session bean name:	CQuerySessEjb
Session type:	Stateless
Transaction type:	Container
JNDI Name:	ejb/cust/CQuerySessEjbHome
[✓] Remote client view	
Remote home interface:	cust.CQuerySessEjbHome
Remote interface:	cust.CQuerySessEjb
[✓] Local client view	
Local home interface:	cust.CQuerySessEjbLocalHome
Local interface:	cust.CQuerySessEjbLocal

Figure 8-95 Definition of the EJB project

Click **Finish.**

The new project is generated now, an intermediate message asks for copying the J2C code in the new EJB project. See Figure 8-96.

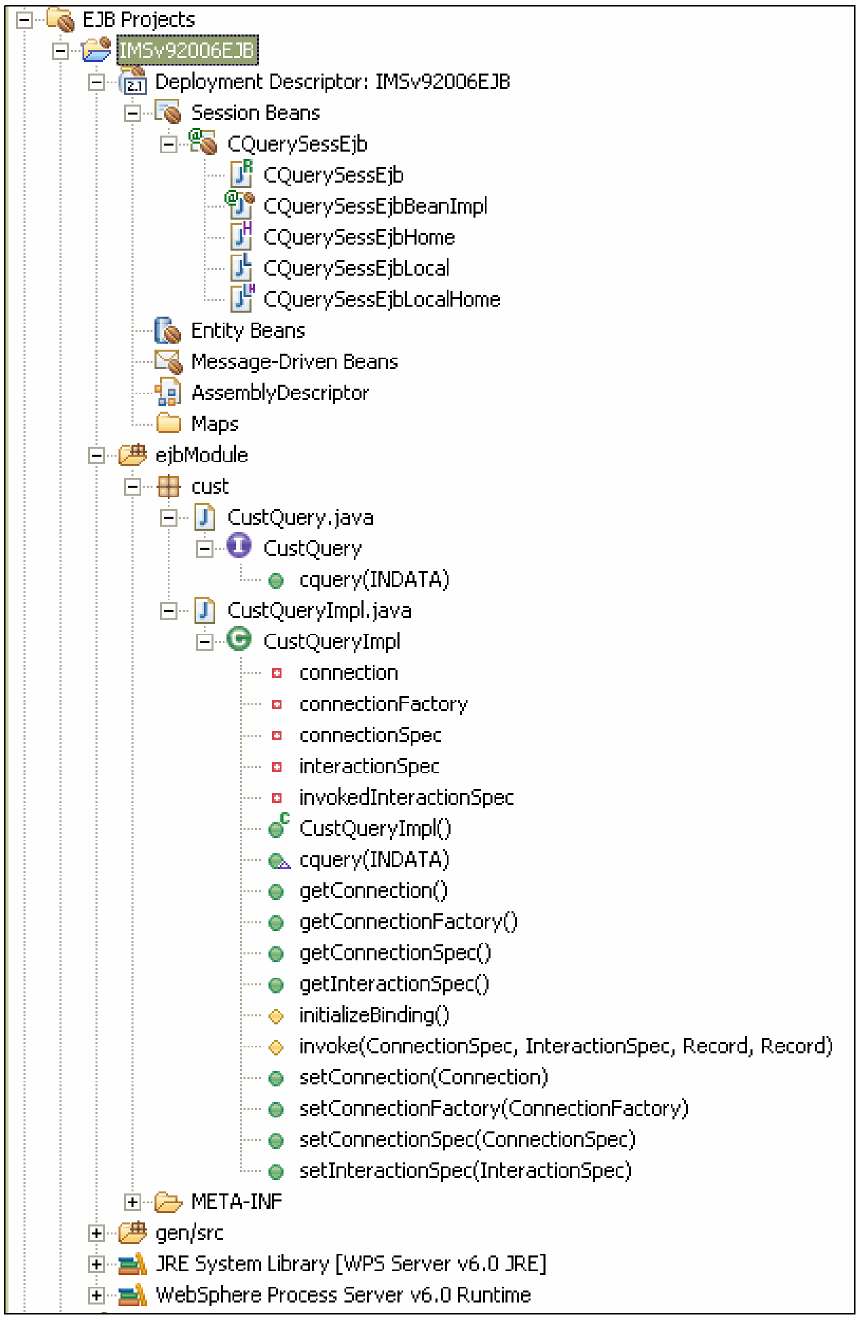


For correct EJB deployment, the J2C Bean code should be present in the EJB project. All relevant J2C code will be copied from project IMSv92006Connector to EJB project IMSv9EJB2006

Figure 8-96 Copy J2C Bean code

Click **OK.**

After this the generation of the Ejb project is completed. The screen capture of Figure 8-97 on page 279 shows you the layout of the project.



```
EJB Projects
  IMSv92006EJB
    Deployment Descriptor: IMSv92006EJB
      Session Beans
        CQuerySessEjb
          CQuerySessEjb
          CQuerySessEjbBeanImpl
          CQuerySessEjbHome
          CQuerySessEjbLocal
          CQuerySessEjbLocalHome
      Entity Beans
      Message-Driven Beans
      AssemblyDescriptor
      Maps
    ejbModule
      cust
        CustQuery.java
          CustQuery
            cquery(INDATA)
        CustQueryImpl.java
          CustQueryImpl
            connection
            connectionFactory
            connectionSpec
            interactionSpec
            invokedInteractionSpec
            CustQueryImpl()
            cquery(INDATA)
            getConnection()
            getConnectionFactory()
            getConnectionSpec()
            getInteractionSpec()
            initializeBinding()
            invoke(ConnectionSpec, InteractionSpec, Record, Record)
            setConnection(Connection)
            setConnectionFactory(ConnectionFactory)
            setConnectionSpec(ConnectionSpec)
            setInteractionSpec(InteractionSpec)
      META-INF
    gen/src
    JRE System Library [WPS Server v6.0 JRE]
    WebSphere Process Server v6.0 Runtime
```

Figure 8-97   Ejb project IMSv92006EJB with imbedded J2C bean and databindings

The session EJB **CQuerySessEjb**, has a reference to **ims/IMSSJ**, which is the managed connection factory. Before generating deployment code the EJB is composed of five artifacts:

- Local Home
- Local Object
- Remote Home
- Remote Object
- Bean
  - Implements the SessionBean interface
  - Extends the J2Cbean

Figure 8-98 gives a graphical detailed look at the EJB **CQuerySessEjb.**

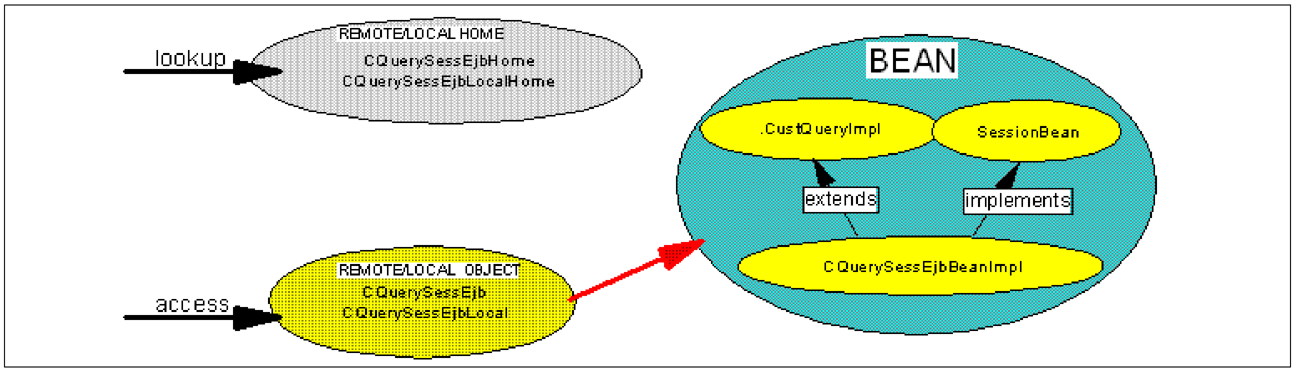


Figure 8-98 Stateless Session EJB on J2C JavaBean

By opening the imbedded **CustQueryImpl** Java code we find the jndiName reference for the lookup by the EJB, indicated by an annotation. The EJB has besides the life cycle methods one business method cquery inherited from the J2C Javabean. Like in all EJB's this is a remote accessible method, which has been promoted to the **CQuerySessEjb**, ejb remote interface and also to the local interface **CQuerySessEjbLocal.**

Example 8-12 Remote interface

```
package cust;

/**
 * Remote interface for CQuerySessEjb bean.
 */
public interface CQuerySessEjb extends javax.ejb.EJBObject {
   /**
    */
   public cust.OUTDATA cquery(cust.INDATA arg)
         throws java.rmi.RemoteException, javax.resource.ResourceException;
}
```

With a RAD Wizard we could now select to build a Web Service SOAP service on it, which can have HTTP or JMS as transport protocol. This possibility is shown in Figure 8-90 on page 275. The outcome will be a SCA import component, but with a Java interface. This has some consequences as explained later

The EJB being ready, in the next step we will create an **import** on it, This happens in a Business Integration module with WID.

Preparation of the Connector Business Integration module

We define a new Business Integration Module, that will contain the import of the J2C session EJB.

Repeat the steps as in "Preparation of the Connector Business Integration module" on page 258, choose as the name of the module **SOAIMSv92006fromRADEJB.**

Build the Import of the EJB

Open the Assembly Diagram of the new module, by clicking twice on **SOAIMSv92006fromRADEJB** under the same module name. A panel opens in the right part of the window, but it is empty.

From the palette on the left in this pane, drop an import component on the board. See Figure 8-99. Change the name of the new component to **CQueryImport_java**.

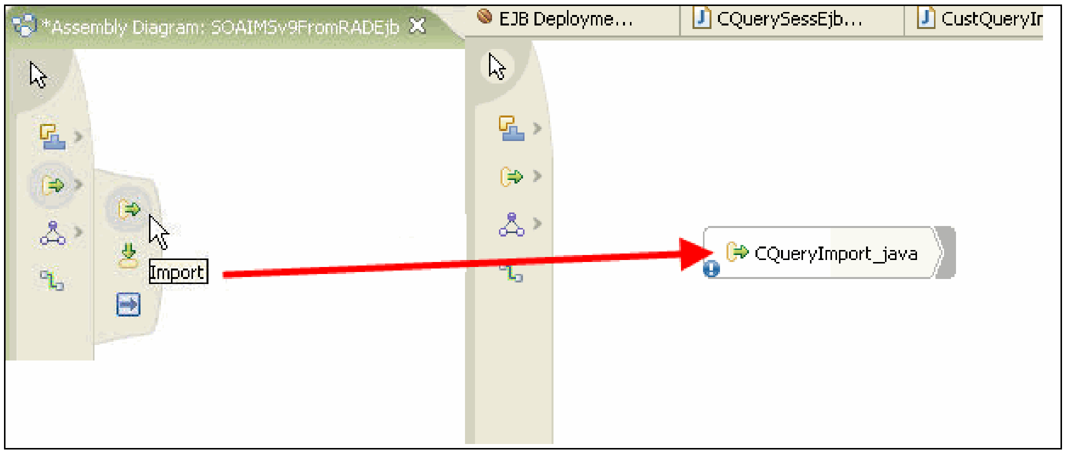


Figure 8-99 Picking an import component from the palette and name change

The **import** component is currently empty without interface and implementation. To be able to use an existing implementation in this component, we have to make this module dependant on foreign projects, which contain the implementation.

Select module **SOAIMSv92006fromRADEJB**.

Right-click **Open Dependency Editor.**

In the window that comes up, expand the J2EE part, **click Add** in that part, to include another J2EE project from which implementations can be included. See Figure 8-100 on page 282.

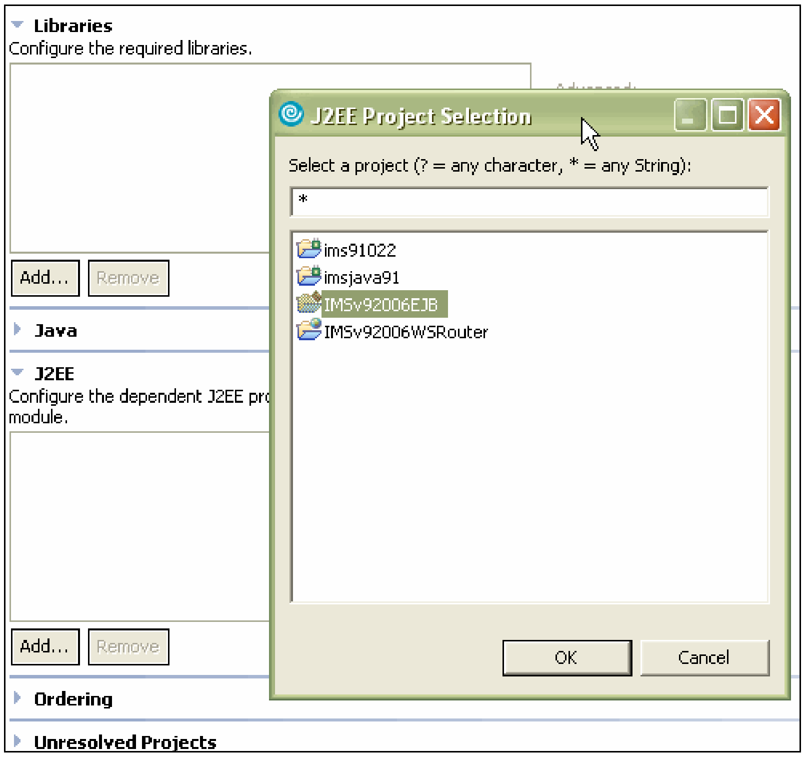


Figure 8-100 Business Dependency

Select project **IMSv92006EJB**, click **OK, close** and **save** the **dependency.** See Figure 8-100.

On the Assembly Diagram board, right-click the **import** component, →**Add Interface.**

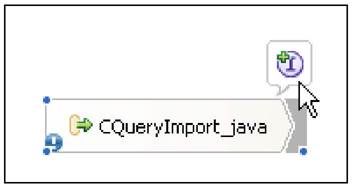


Figure 8-101 Adding an interface

On the pop-up, change the select to **Show Java** and wait till a selection list comes up. This can take a while. Eventually change the selection filter to CQ, so that the list becomes very short. Select **CQuerySessEjb** which is the remote interface object of the session Ejb previously created with the RAD wizard, see Figure 8-102 on page 283.

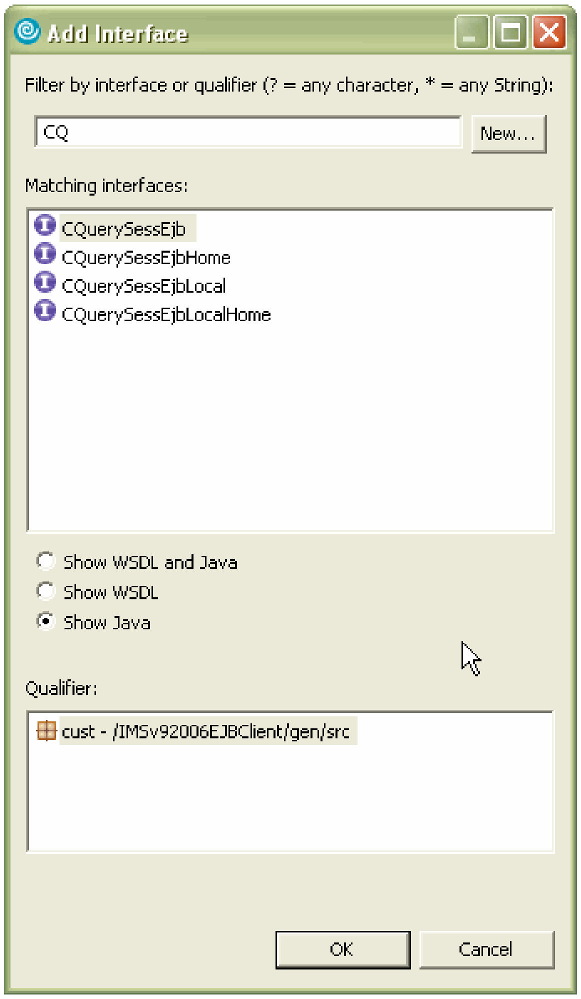


Figure 8-102 Interface selection

Click **OK.**

An interface has now been added to the **import**, the next step is to generate the binding to the EJB implementation. See Figure 8-103 on page 284.

The problem with a **Java** defined interface is that it *cannot* participate in processes built with the Process Choreographer. We have to describe a WSDL interface that can be a **partner link** in a process and this WSDL has to be implemented by a Java component which is a front-end for the EJB invocation. We should also NOT forget that inside a process data are transported in a Service Data Object(SDO) format, which is different of the format used on the J2C interface.

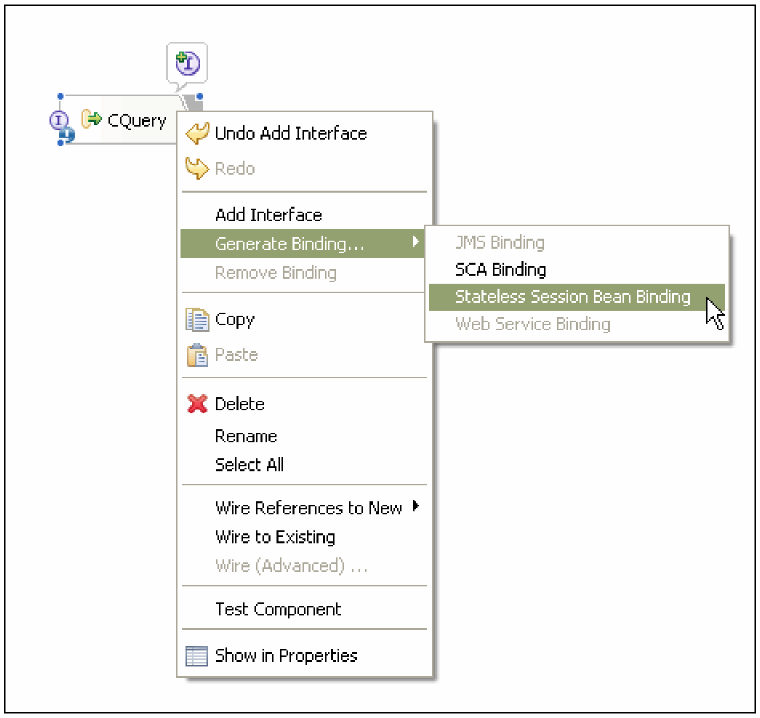


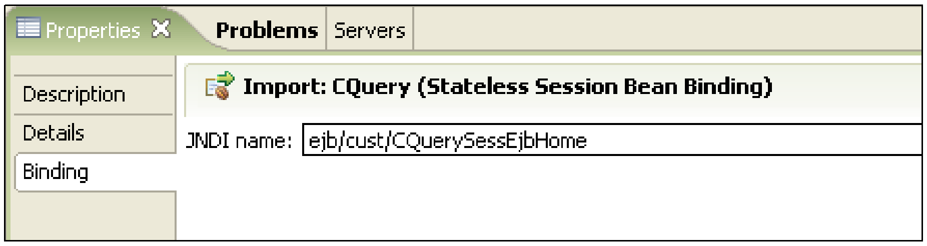
Figure 8-103 Binding generation

Select import component.

Right-click **Generate Binding** → **Stateless Session Bean Binding.**

In the right bottom part of the window, select Properties in the menu, select Binding on the left and verify/complete the jndiName information with the jndiName of the **CQuerySessEjb.**

See Figure 8-104**.**



Import: CQuery (Stateless Session Bean Binding)

JNDI name: ejb/cust/CQuerySessEjbHome

Figure 8-104 Complete binding with JNDIname of Connector EJB

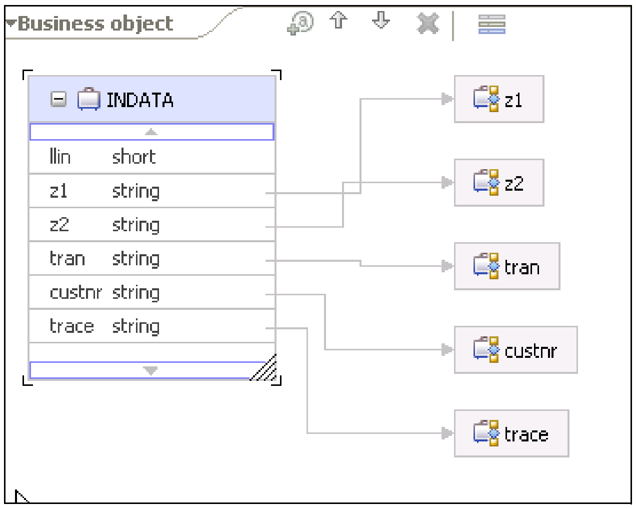
The Business Integration import definition should now be complete, but as we explained it *cannot* be used from the **process choreographer** because of the missing WSDL interface.

We build this WSDL interface manually. Most of it was already prepared by the WID generated system component in project **SOAIMSv92006**. This WSDL is simply an metadata XML representation of:

OUTDATA = CQUERY(INDATA);

We use two datatypes (INDATA,OUTDATA) and a WSDL(CQUERY) representing the cquery entry. Everything can be copied from **SOAIMSv92006** into the current project.

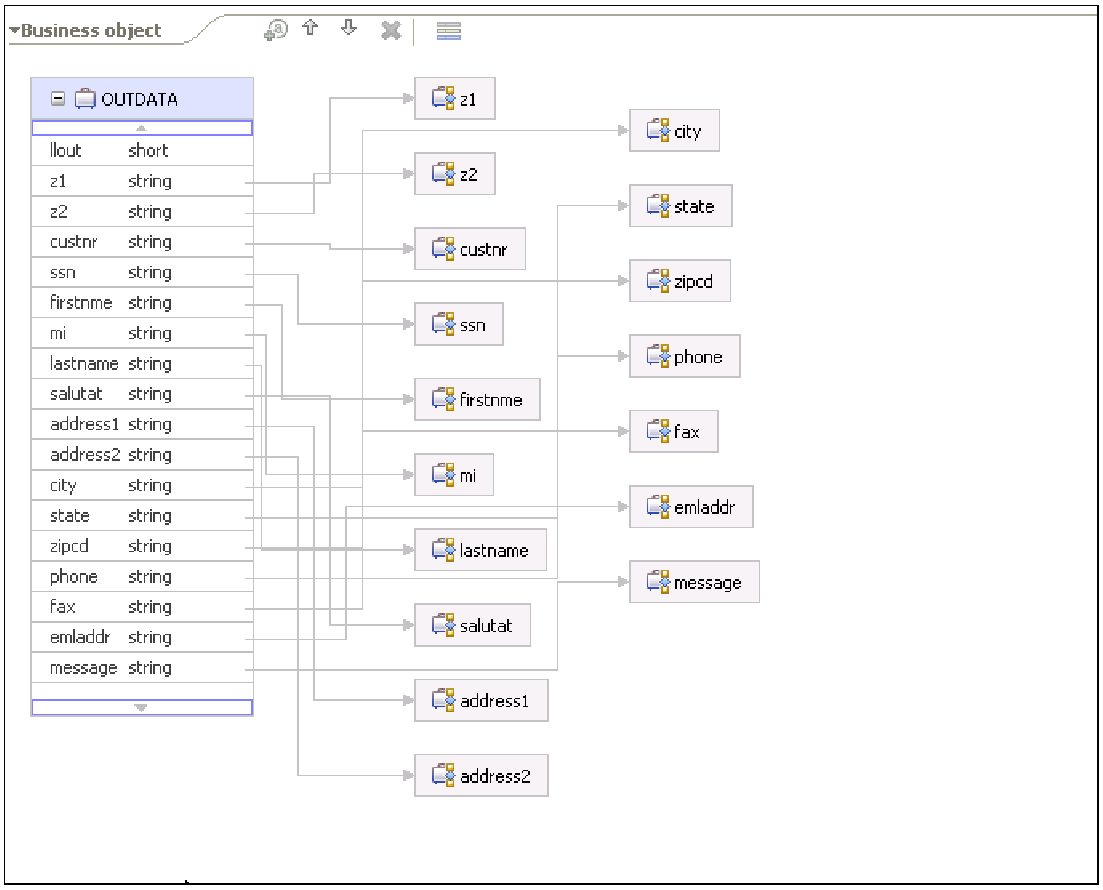
Figure 8-105 shows the generated datatypes.



llin	short
z1	string
z2	string
tran	string
custnr	string
trace	string

Figure 8-105 INDATA

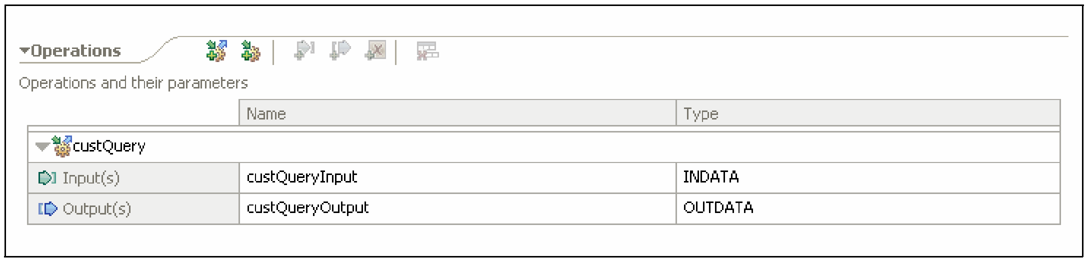
By clicking on the INDATA and OUTDATA we can have a graphical look at the data objects. See Figure 8-106.



llout	short
z1	string
z2	string
custnr	string
ssn	string
firstnme	string
mi	string
lastname	string
salutat	string
address1	string
address2	string
city	string
state	string
zipcd	string
phone	string
fax	string
emladdr	string
message	string

Figure 8-106 OUTDATA

In Figure 8-107 on page 286 you the presentation of the WSDL interface.



	Name	Type
custQuery		
Input(s)	custQueryInput	INDATA
Output(s)	custQueryOutput	OUTDATA

Figure 8-107 WSDL description of the interface

To accommodate the problem of non supported Java interface by the Process builder, we add as a frontal to the import component a Java component which will use the prepared WSDL interface and function as a mapper between the supported **WSDL** based partner and the Java based import.

From the palette on the left take and drop a Java component on the **Assembly** editor. Change the name to **CQueryPassThru4Java.**

This component, for functioning needs the following

- An interface (CQuery.wsdl)
- A reference connection to the import
- An implementation

As before we add an interface **CQuery.wsdl** to the Java component.

We add a **reference connection** by drawing the arrow from the tail of the Java component to the **import** component. After arranging the picture it looks like in the following capture. See Figure 8-108

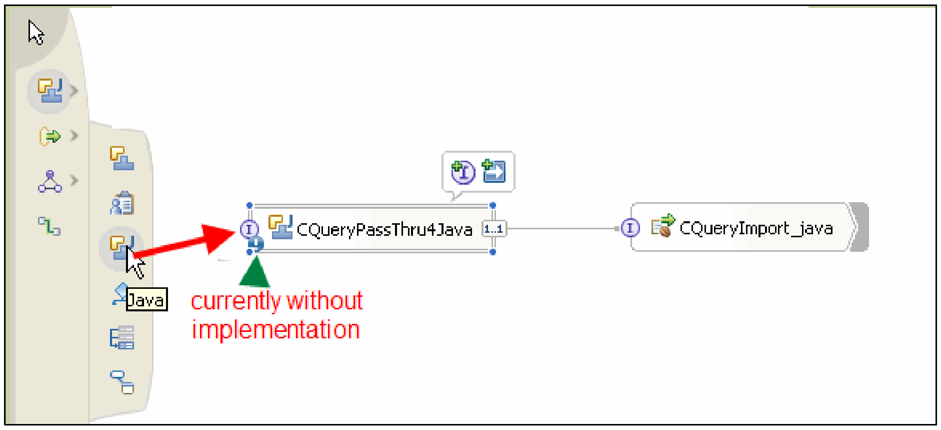


Figure 8-108 Import component with frontal Java component

The next step is to add the implementation on the **Java component**. It is always possible to generate a skeleton that later on has to be completed. Generate the skeleton as indicated.

Select Java component, right-click and select **Generate Implementation.**

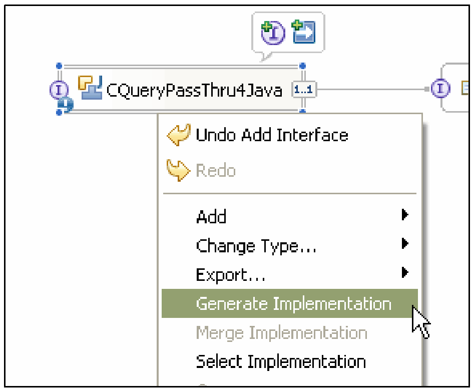


Figure 8-109 Generate skeleton for implementation

Choose cusjava as the directory/folder, click **Finish.** A Java skeleton **`CQueryPassThru4JavaImpl`** is generated now, This has to be completed.

In the generated skeleton code, see Example 8-13, several method entries are important to understand.

`public CQuerySessEjb locateService_CQuerySessEjbPartner()`

Location of the EJB service for the J2C implementation

`public DataObject custQuery(DataObject custQueryInput)`

Mapping between SDO and regular classes (this will be implemented)

`void onCqueryResponse(Ticket __ticket, DataObject returnValue`

It not used here

Example 8-13 Skeleton code generated by wizard

```
package cusjava;

import com.ibm.websphere.sca.Ticket;
import commonj.sdo.DataObject;
import cust.CQuerySessEjb;
import com.ibm.websphere.sca.ServiceManager;

public class CQueryPassThru4JavaImpl {
   /**
    * Default constructor.
    */
   public CQueryPassThru4JavaImpl() {
      super();
   }
   private Object getMyService() {
      return (Object) ServiceManager.INSTANCE.locateService("self");
   }
   /**
    * This method is used to locate the service for the reference
    * named "CQuerySessEjbPartner".  This will return an instance of
    * {@link CQuerySessEjb}.  If you would like to use this service
    * asynchronously then you will need to cast the result
    * to {@link CQuerySessEjbAsync}.
    * @return CQuerySessEjb
    */
   public CQuerySessEjb locateService_CQuerySessEjbPartner() {
      return (CQuerySessEjb) ServiceManager.INSTANCE
```

```
            .locateService("CQuerySessEjbPartner");
   }
   /**
    * Method generated to support implemention of operation "custQuery" defined for WSDL
port type
    * named "interface.Cquery".
    *
    * The presence of commonj.sdo.DataObject as the return type and/or as a parameter
    * type conveys that its a complex type. Please refer to the WSDL Definition for more
information
    * on the type of input, output and fault(s).
    */
   public DataObject custQuery(DataObject custQueryInput) {
      //TODO Needs to be implemented.
      return null;
   }
   /**
    * Method generated to support the async implementation using callback
    * for the operation (@link cust.CQuerySessEjb#cquery(DataObject aINDATA))
    * of java interface (@link cust.CQuerySessEjb)
    * @see cust.CQuerySessEjb#cquery(DataObject aINDATA)
    */
   public void onCqueryResponse(Ticket __ticket, DataObject returnValue,
         Exception exception) {
      //TODO Needs to be implemented.
   }
}
```

The following logic has to be implemented in this code. On input and output we have SDO objects for the data in INDATA and OUTDATA, but the EJB entry expects a regular INDATA J2C databinding class object on input and returns a regular J2C OUTDATA object. In the java code we have to code the mapping between the SDO format and the regular class format.

Example 8-14 follows the changed custQuery method.

Example 8-14 Changed custQuery method

```
   public DataObject custQuery(DataObject indataBO) {
      DataObject outdataDO = null;
      String trace = null;
      boolean debugOn = true;
      System.out.println("++CQuery4JavaImpl_custQuery enter");
      try {
         BOFactory boFactory = (BOFactory) new ServiceManager()
               .locateService("com/ibm/websphere/bo/BOFactory");
         cust.OUTDATA outdata = null;
         cust.INDATA indata = new INDATA();
         trace = indataBO.getString("trace");
         if (trace.equalsIgnoreCase("Y")) {
            debugOn = true;
         }
         if (debugOn) {
            System.out.println("++CQuery4JavaImpl_custQuery tran("
                  + indataBO.getString("tran") + ") custnr("
                  + indataBO.getString("custnr") + ") trace("
                  + indataBO.getString("trace") + ")" );
         }
         indata.setLlin(indataBO.getShort("llin"));
         indata.setZ1(indataBO.getString("z1"));
         indata.setZ2(indataBO.getString("z2"));
         indata.setTran(indataBO.getString("tran"));
         indata.setCustnr(indataBO.getString("custnr"));
         indata.setTrace(indataBO.getString("trace"));
         outdataDO = boFactory.create("http://SOAIMSv92006fromRADEJB/cust",
         "OUTDATA");
         if (debugOn) {
            System.out.println("++CQuery4JavaImpl_custQuery before EJB call");
         }
         outdata = locateService_CQuerySessEjbPartner().cquery(indata);
         if (debugOn) {
            System.out.println("++CQuery4JavaImpl_custQuery after EJB call
outdata(" + outdata + ")");
         }
         outdataDO.setShort("llout", outdata.getLlout());
         outdataDO.setString("z1", outdata.getZ1());
         outdataDO.setString("z2", outdata.getZ2());
         outdataDO.setString("custnr", outdata.getCustnr());
         outdataDO.setString("ssn", outdata.getSsn());
         outdataDO.setString("firstnme", outdata.getFirstnme());
         outdataDO.setString("mi", outdata.getMi());
         outdataDO.setString("lastname", outdata.getLastname());
         outdataDO.setString("salutat", outdata.getSalutat());
         outdataDO.setString("address1", outdata.getAddress1());
         outdataDO.setString("address2", outdata.getAddress2());
         outdataDO.setString("city", outdata.getCity());
         outdataDO.setString("state", outdata.getState());
         outdataDO.setString("zipcd", outdata.getZipcd());
         outdataDO.setString("phone", outdata.getPhone());
         outdataDO.setString("emladdr", outdata.getEmladdr());
         outdataDO.setString("z1", outdata.getZ1());
         outdataDO.setString("message", outdata.getMessage());
      } catch (Exception ex) {
         System.err.println("++CQuery4JavaImpl_custQuery exception "
               + ex.toString());
         outdataDO.setString("message", "+CQuery4JavaImpl_custQuery exception "
               + ex.toString());
      }
      if (debugOn) {
         System.out.println("++CQuery4JavaImpl_custQuery leaving");
      }
      return outdataDO;
   }
```

This SCA component is now completed and can be tested, but the J2EE artifacts have to be generated for deployment in the J2EE container.

Build the module

The Business module is now like in Figure 8-110 on page 290. You recognize the Datatypes, Interfaces and the Javacode.

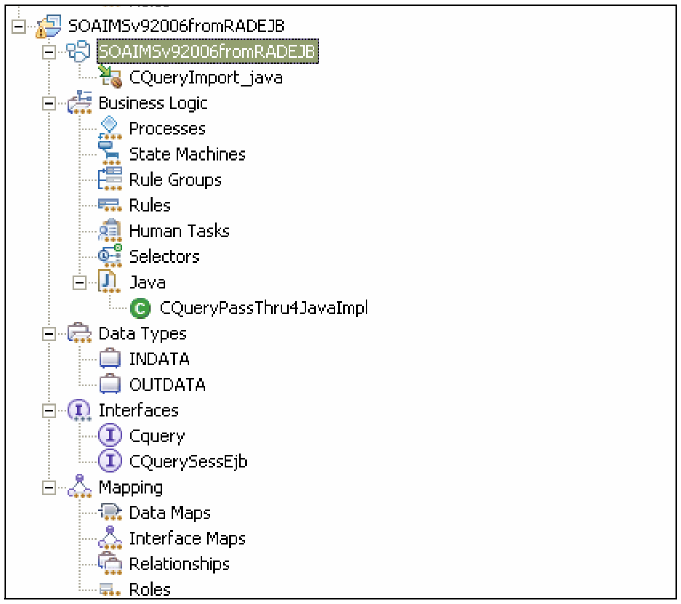


Figure 8-110 Layout of SOAIMSv92006RADEJB business module

A complete build of all artifacts is now required. The best way to perform this build of the whole module is to switch to a J2EE perspective and to select the Business Integration project, and **Clean** on the menu. Clean for this project should be the only choice.

Testing the SCA component

The SCA component can now be tested. Select the Java component, and select Test Component as indicated in Figure 8-111.

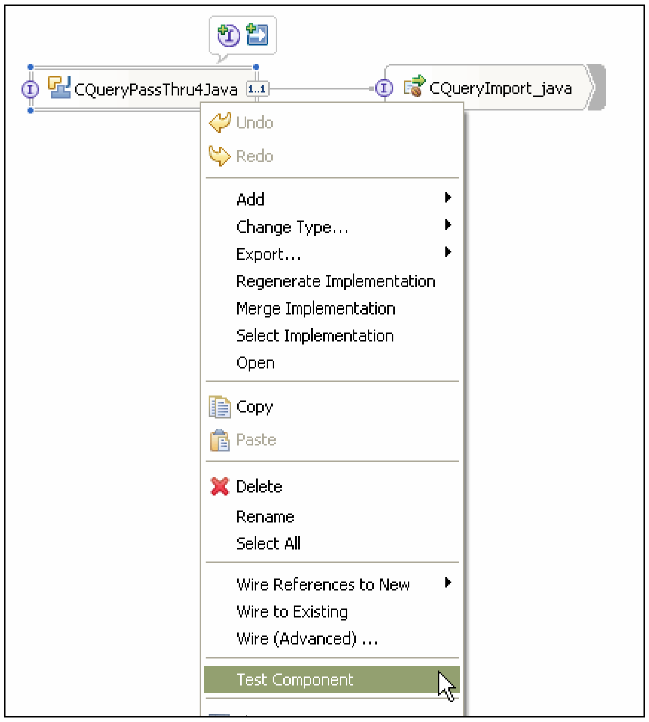
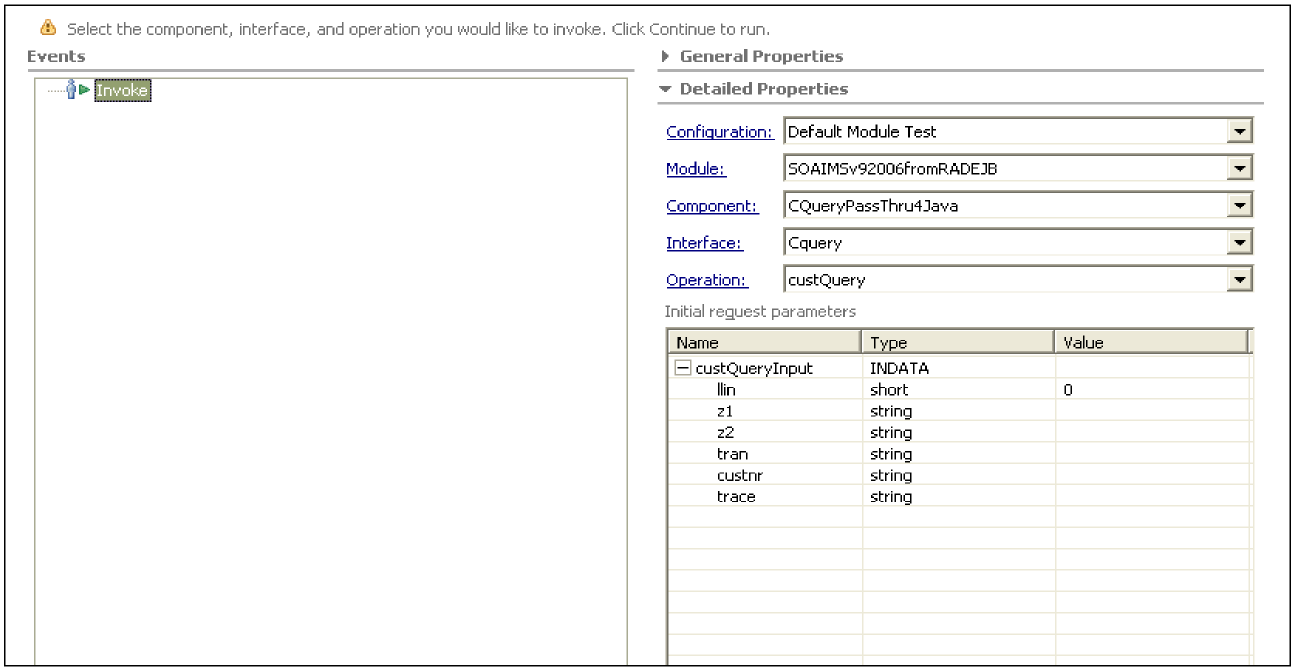


Figure 8-111 Test component from SOA accessible Java component

The WebSphere Process Server is started and testing can be done from Figure 8-112.


Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: SOAIMSv92006fromRADEJB
Component: CQueryPassThru4Java
Interface: Cquery
Operation: custQuery

Initial request parameters

Name	Type	Value
custQueryInput	INDATA	
llin	short	0
z1	string	
z2	string	
tran	string	
custnr	string	
trace	string	

Figure 8-112 Test panel

Conclusion

With a WID wizard we have built a **import** component from an existing RAD solution for an IMS transaction access. Although we did the exercise for an EJB, we could have done for other configurations as shown in Figure 8-91 on page 275.

8.5 Accessing DLI data

IMS Java class libraries (the JDBC driver is included in these libraries) provide direct IMS database access from multiple environments. Direct access means just that - it is direct and there is no need for an IMS transaction (although the driver can be used within an IMS transaction as well to access IMS data). The ODBA interface which was introduced in IMS V7 allows for direct DB access from non-IMS controlled environments. The front-end to the JDBC driver is a JCA resource adapter which allows for deployment in a WebSphere Application Server (much like IMS Connector for Java - which is another JCA resource adapter offered by IMS which provides access to existing IMS transactions).

The WebSphere Application Server can either be WebSphere Application Server z/OS or WebSphere Application Server distributed (in which case you will need to use the RDS solution offered in IMS V9 which allows for WebSphere Application Server distributed to WebSphere Application Server z/OS IIOP communication to offer DB access from a distributed WebSphere Application Server). Your non-z/OS applications can access IMS data over the Internet using a secure protocol with local or global transaction semantics and the standard JDBC API.

Figure 8-113 on page 292 shows the JDBC/DLI solution.

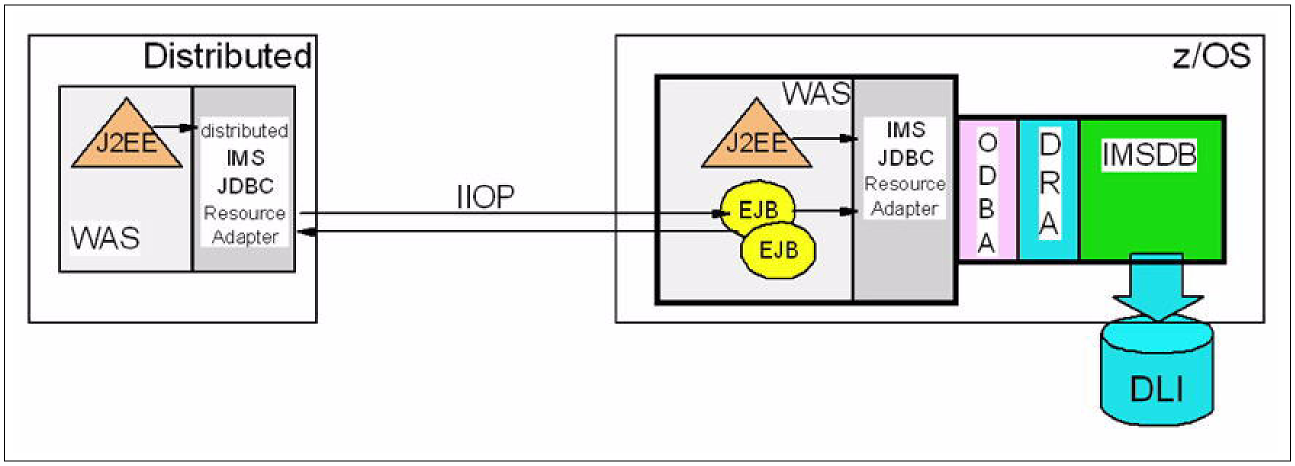


Figure 8-113 Accessing DLI data with JDBC solution

A JDBC resource Adapter is installed in both the z/OS WebSphere Application Server and in the distributed WebSphere Application Server. The distributed WebSphere Application Server could be the WebSphere Test Environment (WTE) of the RAD tooling. In both environments J2EE artifacts (WebApplications, EJB's), can be developed and tested to access directly DLI databases. The access to the DLI data occurs also through a **DataSource** connection factory.

On the z/OS WebSphere Application Server the JDBC resource adapter, uses the Open Data Base Access (ODBA) layer, which is the callable interface to IMSDB. The Database Resource Adapter (DRA) is the bridge between the external subsystem and IMS.

The distributed WebSphere Application Server also has a distributed IMS JDBC Resource Adapter, which contains a type-3 JDBC driver. For this remote access to have functioning, two IMS supplied EJBs have to be installed. One of two IMS Java-supplied EJBs is the host-side component that facilitates communication with and passes transaction information to the IMS JDBC resource adapter. These EJBs act as listeners for remote requests. Depending on whether there is a transaction context on the non-z/OS platform, either a container-managed or bean-managed IMS Java EJB is used. Notice that by remote WebSphere Application Server we also include a RAD WebSphere Test Environment. More information about the subject and downloads can be found at:

http://www.ibm.com/software/data/ims/imsjava/

JDBC access is always on relational databases. DLI databases are hierarchical and lack also descriptive metadata, as it is the case for DB2 with its catalog. Before being able to use JDBC for DLI databases these two issues have to be overcome.

8.5.1 Hierarchical

A database segment definition defines the fields for a set of segment instances similar to the way a relational table defines columns for a set of rows in a table. In this way, segments relate to relational tables, and fields in a segment relate to columns in a relational table. The name of an IMS segment becomes the table name in an SQL query, and the name of a field becomes the column name in the SQL query.

A fundamental difference between segments in a hierarchical database and tables in a relational database is that, in a hierarchical database, segments are implicitly joined with each other. In a relational database, you explicitly join two tables. Looking at the hierarchical data, we can consider that all dependant segments have a foreign key, which is a concatenations of the keys its parent path. A segment instance in a hierarchical database is already joined with its parent segment and its child segments, which are all along the same hierarchical path. In a relational database, this relationship between tables is captured by foreign and primary keys.

To be able to be used with JBDC this hierarchical has to be transformed logically in a relational view. This happens by concatenating all segments in its hierarchical path. The key of a segment/row in the relational view is a concatenation of all keys in its path. This is shown in Figure 8-114.

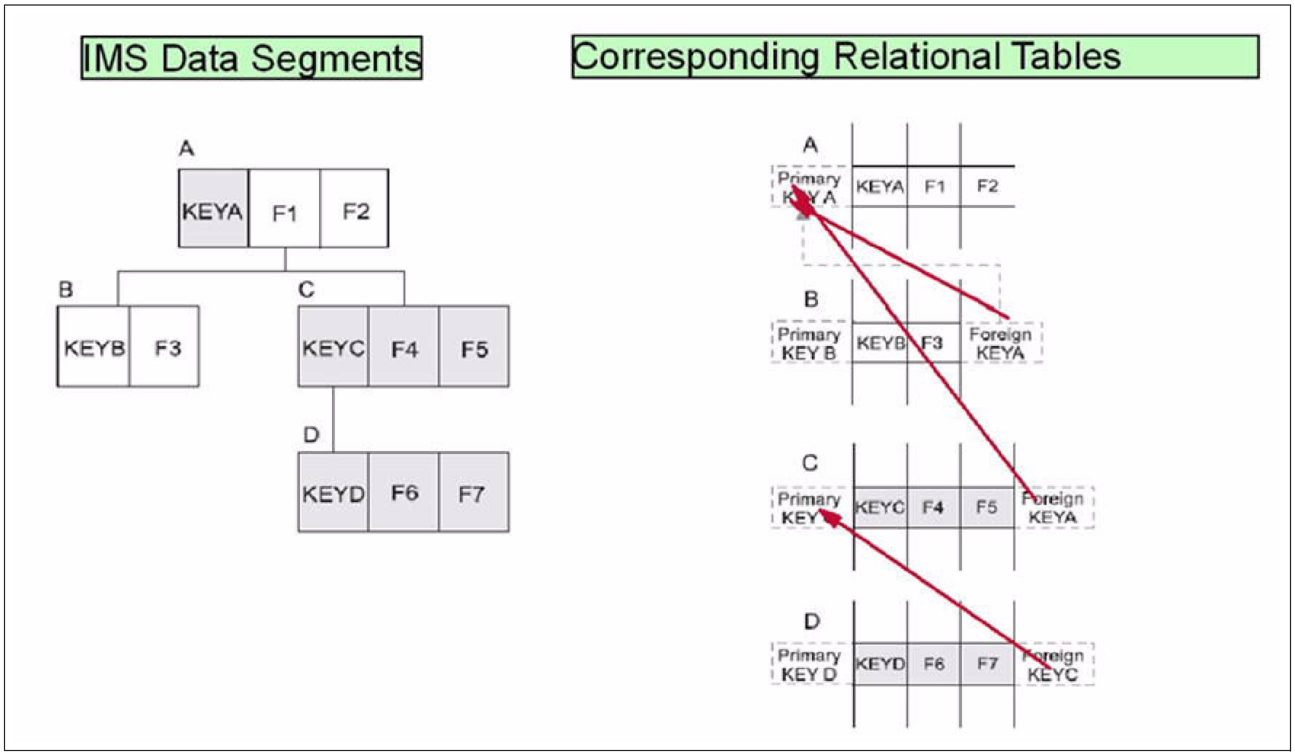


Figure 8-114 Hierarchical to relational transformation

8.5.2 Metadata

Metadata is a requirement for the JDBC functioning. In order for a Java application to access an IMS database, it needs information about the database. This information is contained in the PSB (program specification block) and DBDs (database descriptions), but you must first convert this information into a form that you can use in the Java application: a subclass of the com.ibm.ims.db.DLIDatabaseView class called the IMS Java metadata class. Metadata is not available for DLI databases and it has to be built with the *DLIModel* utility function. The DLIModel utility generates this metadata from the IMS PSBs, DBDs, COBOL copybooks, and other input specified by utility control statements.

This can be achieved in two ways:

1. Using Java Batch, see Figure 8-115 on page 294.

   The function produces on request three outputs:

   – DatabaseView class, these are the metadata
   – A Java Report
   – Execution trace of the DLIModel function

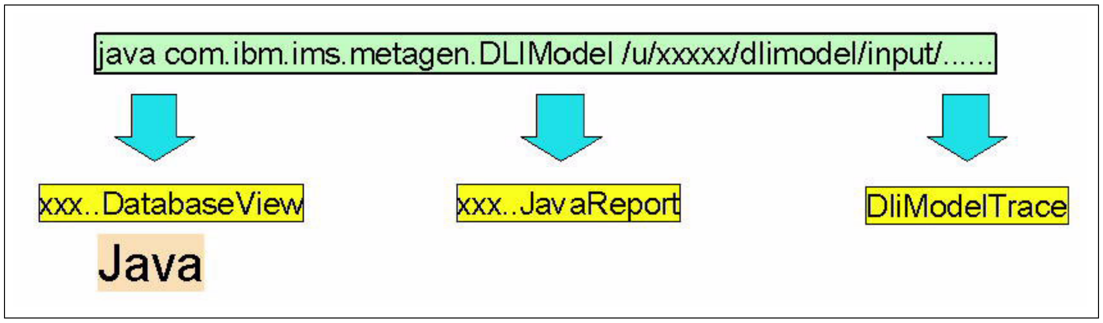

java com.ibm.ims.metagen.DLIModel /u/xxxxx/dlimodel/input/......
xxx..DatabaseView
Java
xxx..JavaReport
DliModelTrace

Figure 8-115 DLIModel

2. An Eclipse **DLIModel** plug-in exists also for WSAD and RAD. At the time of writing this plug-in was *not* available for Eclipse 3 (RAD).

The outcome of either of these operations is a **DatabaseView** Java class, which describes the views to the data through one particular PSB. The same data can be viewed under several angles, depending on the presence of secondary indices, logical relationships. The view to the data can also be partial.

This view can then be considered as the equivalent of a data source, described in the WebSphere Application Server environment and reached via a lookup in the same way as a regular DB2 relational database.

Examples of the DLIModel can be found in reference *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794 and *IMS Version 9: IMS Java Guide and Reference*, SC18-7821.

It is quite easy to create an Enterprise Java Bean that uses the JCA/DLI resource adapter to directly access IMS data. With the RAD development toolkit the EJB can be turned into a Web Service. This is shown in Figure 8-116.

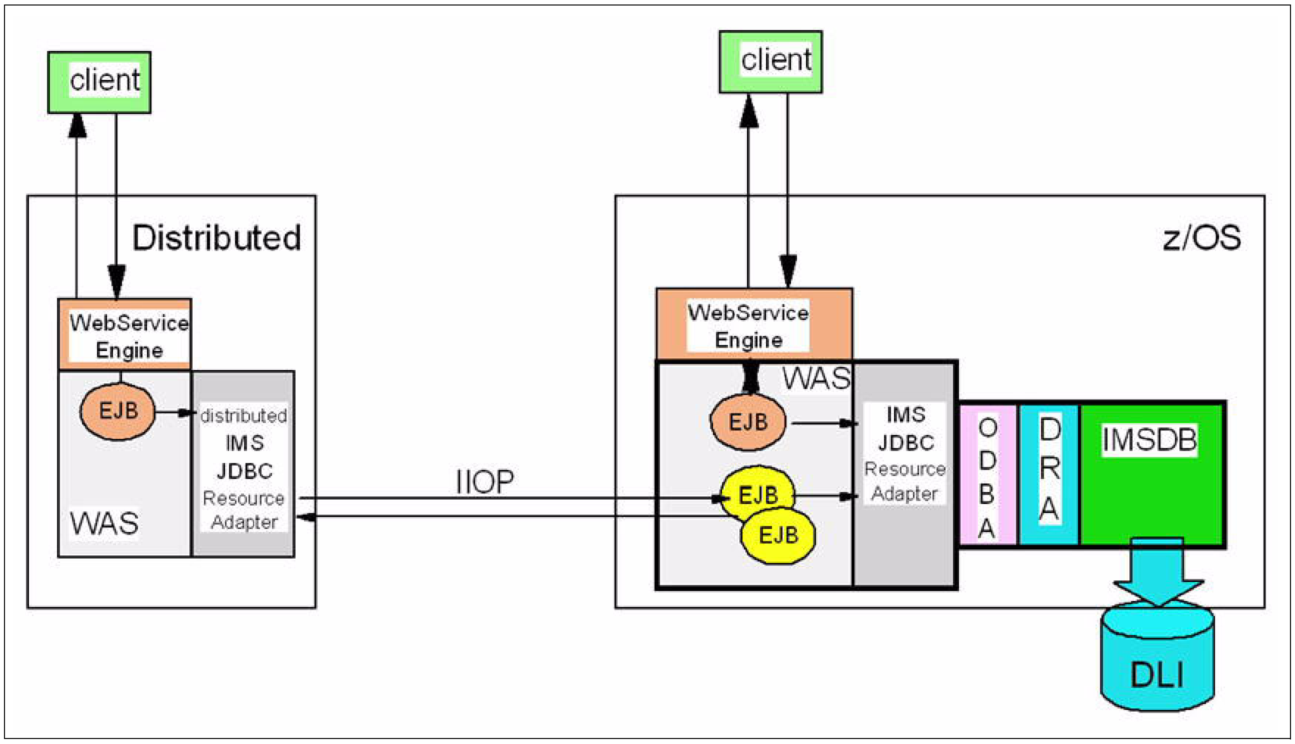


Figure 8-116 IMS and Web services deployment

8.5.3 Conclusion

With a DLI/JDBC adapter it is possible to include access to DLI hierarchical databases in J2EE design. This access can be local or remote, so that even during design, from within the RAD toolkit we use real data on the host. DLI databases are existing for a long time, are stable and well performing, and this additional solution eliminates reasons to migrate to other types of databases.

9

Informix IDS and SOA

Informix IDS is a strategic IBM data service with a special focus on high performance OLTP, easy application integration, low administration and very low total cost of ownership. IDS has a very modern process-architecture and extensibility-architecture which is key for an easy integration of IDS into an SOA framework.

In this chapter we introduce the IDS data service and also cover the following SOA related topics:

- Providing Web services on top of IDS through the WORF framework and exposing IDS database operations through EGL (Enterprise Generation Language) based Web services.
- Consuming Web services by using the Apache Axis framework in combination with IDS J/Foundation.
- IDS foundation technologies like the WebSphere MQ DataBlade and XML related DataBlades which can help with an easy SOA integration.
- Modernizing existing Informix 4GL applications and integrating them into an SOA architecture.

This chapter contains these sections:

- IBM Informix Dynamic Server: An overview
- IDS as a Web services provider
- IDS as a Web services consumer
- XML related DataBlades
- Using WebSphere MQ with Informix applications
- Integrating I4GL applications with SOA through EGL

© Copyright IBM Corp. 2006. All rights reserved.

9.1 IBM Informix Dynamic Server: An overview

IBM Informix Dynamic Server (IDS) is IBM's flagship data service for industrial-strength, embedded computing. IDS offers the lowest TCO (total cost of ownership), providing high performance with low overhead cost, while delivering a robust database platform for on demand businesses.

IDS is the first database to support rolling application upgrades when built using IBM's industry-leading Enterprise Replication technology. IDS provides a full-featured relational database management system (ORDBMS) platform with capabilities to extend its functionality to meet unique business requirements with the speed of native database functions.

IDS reduces downtime, improves reliability, and supports a wide-array of platforms and application development environments including Java TM and Eclipse, and Microsoft .NET IDEs.

9.1.1 The IDS architecture

The IBM IDS engine architecture is based on advanced technology that efficiently uses virtually all of today's hardware and software resources.

Called the Dynamic Scalable Architecture (DSA), it fully exploits the processing power available in SMP environments by performing similar types of database activities (such as I/O, complex queries, index builds, log recovery, inserts and backups/restores) in parallel groups rather than as discrete operations.

The DSA design architecture includes built-in multi-threading and parallel processing capabilities, dynamic and self-tuning shared memory components, and intelligent logical data storage capabilities, supporting the most efficient use of all available system resources.

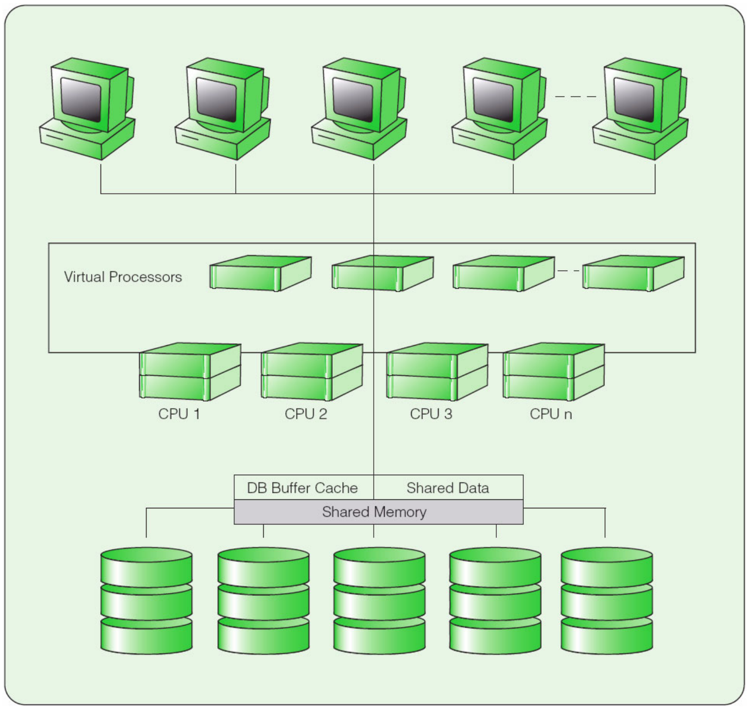


Figure 9-1 IBM IDS architectural overview

Processing

IBM IDS provides the unique ability to scale the database system by employing a dynamically configurable pool of database server processes called virtual processors. Database operations such as a sorted data query are broken into task-oriented subtasks (for example, data read, join, group, sort) for rapid processing by virtual processors that specialize in that type of subtask. Virtual processors mimic the functionality of the hardware CPUs in that virtual processors schedule and manage user requests using multiple, concurrent threads.

A thread represents a discrete task within a database server process and many threads may execute simultaneously, and in parallel, across the pool of virtual processors. Unlike a CPU process-based (or single-threaded) engine, which leaves tasks on the system CPU for its given unit of time (even if no work can be done thus wasting processing time), virtual processors are multi-threaded. Consequently, when a thread is either waiting for a resource or has completed its task, a thread switch will occur and the virtual processor will immediately work on another thread. As a result, precious CPU time is not only saved, but it is used to satisfy as many user requests as possible in the given amount of time. This is referred to as fan-in parallelism.

Not only can one virtual processor respond to multiple user requests in any given unit of time, but one user request can also be distributed across multiple virtual processors. For example, with a processing-intensive request such as a multi-table join, the database server divides the task into multiple subtasks and then spreads these subtasks across all available virtual processors. With the ability to distribute tasks, the request is completed quicker. This is referred to as fan-out parallelism. Together with fan-in parallelism, the net effect is more work being accomplished quicker than with single-threaded architectures; in other words, the engine is faster.

Dynamic load balancing occurs within IBM IDS because threads are not statically assigned to virtual processors. Outstanding requests are serviced by the first available virtual processor, balancing the workload across all available resources. For efficient execution and versatile tuning, virtual processors can be grouped into classes, each optimized for a particular function, such as CPU operations, disk I/O, communications and administrative tasks. An administrator can configure the system with the appropriate number of virtual processors in each class to handle the workload. Adjustments can be made while the engine is online without interrupting database operations in order to handle occasional periods of heavy activity or different load mixes.

In UNIX and Linux systems, the use of multi-threaded virtual processors significantly reduces the number of UNIX/Linux processes and, consequently, less context switching is required. In Microsoft Windows systems, virtual processors are implemented as threads to take advantage of the operating system's inherent multi-threading capability. Because IBM IDS includes its own threading capability for servicing client requests, the actual number of Windows threads is decreased, reducing the system thread scheduling overhead and providing better throughput.

In fully utilizing the hardware processing cycles, IBM IDS engines do not need as much hardware power to achieve comparable to better performance than other database engines.

Memory

All memory used by IBM IDS is shared among the pool of virtual processors. Beyond a small initial allocation of memory for engine-level management, usually a single shared memory portion is created and used by the virtual processors for all data operations. This portion contains the buffers of queried and modified data, sort, join and group tables, lock pointers, and so on. Should database operations require more (or less) shared memory, additional segments will be dynamically added and dropped from this portion without interrupting user activities.

An administrator can also make similar modifications manually while the server is running. When a user session terminates, the thread-specific memory for that session is freed within the portion and reused by another session.

The buffer pool is used to hold data from the database disk supply during processing. When users request data, the engine first attempts to locate the data in the buffer pool to avoid unnecessary disk I/Os. Depending on the characteristics of the engine workload, increasing the size of the buffer pool can result in a significant reduction in the number of disk accesses, which can help significantly improve performance, particularly for online transaction processing (OLTP) applications.

Disks

The parallelism and scalability of the DSA processor and memory components are supported by the ability to perform asynchronous I/O across database tables and indexes that have been logically partitioned. To speed up what is typically the slowest component of database processing, IBM IDS uses its own asynchronous I/O (AIO) feature, or the operating system's kernel AIO, when available. Because I/O requests are serviced asynchronously, virtual processors do not have to wait for one I/O operation to complete before starting work on another request. To ensure that requests are prioritized appropriately, four specific classes of virtual processors are available to service I/O requests: logical log I/O, physical log I/O, asynchronous I/O and kernel asynchronous I/O. With this separation, an administrator can create additional virtual processors to service specific types of I/O in order to alleviate any bottlenecks that might occur.

Data partitioning

Table and index data can be logically divided into partitions, or fragments, using one or more "partitioning schemes" to improve the ability to access several data elements within the table or index in parallel as well as increase and manage data availability and currency. For example, if a sequential read of a partitioned table were required, it would complete quicker because the partitions would be scanned simultaneously rather than each disk section being read serially from the top to the bottom. With a partitioned table, database administrators can move, associate or disassociate partitions to easily migrate old or new data into the table without tying up table access with mass inserts or deletes.

IBM IDS has two major partitioning schemes that define how data is spread across the fragments. Regardless of the partitioning scheme chosen, or even if none is used at all, the effects are transparent to end users and their applications. Table partitions can be set and altered without bringing down the database server and, in some cases, without interrupting user activity within the table. When partitioning a table, an administrator can specify either:

- Round robin - Data is evenly distributed across each partition with each new row going to the next partition sequentially.
- Expression-based - Data is distributed into the partitions based on one or more sets of logical rules applied to values within the data.

Leveraging the strengths of DSA

With an architecture as robust and efficient as IBM IDS, the engine provides a number of performance features that other engines cannot match.

The speed with which IBM IDS responds to a data operation can vary depending on the amount of data being manipulated and the database's design. While many simple OLTP operations such as single row inserts/updates/deletes can be executed without straining the system, a properly designed database can leverage IBM IDS features such as parallel data query, parallel scan, sort, join, group and data aggregation for larger, more complex operations.

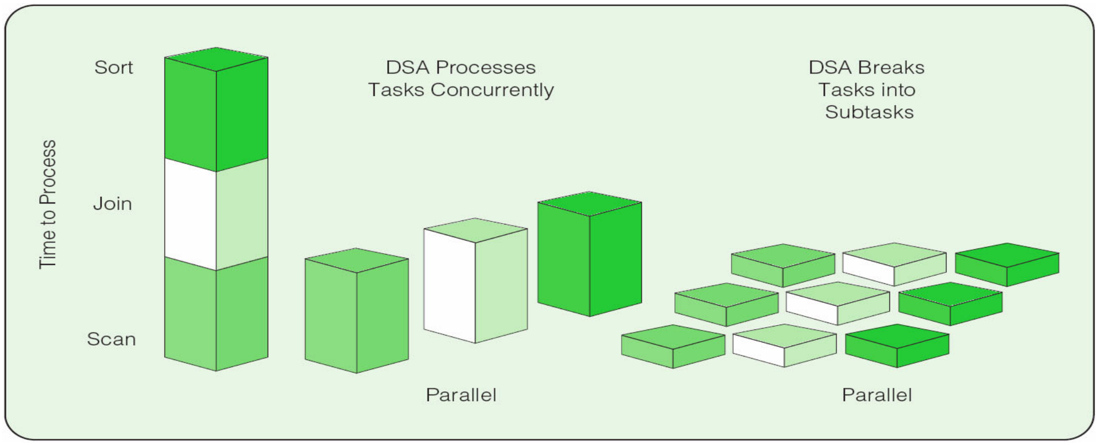

Time to Process
Sort
Join
Scan
DSA Processes Tasks Concurrently
DSA Breaks Tasks into Subtasks
Parallel
Parallel

Figure 9-2 Benefits of IBM IDS parallel data query feature

The parallel data query (PDQ) feature takes advantage of the CPU power provided by SMP systems and the IBM IDS virtual processors to execute fan-out parallelism. PDQ is of greatest benefit to more complex SQL operations that are more analytical, or OLAP oriented, than operational, or OLTP oriented. With PDQ enabled, not only is a complex SQL operation divided into a number of sub-tasks but the sub-tasks are given higher or lower priority for execution within the engine's resources based on the overall "PDQ-priority" level requested by the operation.

9.1.2 Extensibility in IDS: Key for SOA integration

IBM IDS provides a complete set of features to extend the database server, including support for new data types, routines, aggregates and access methods. With this technology, in addition to recognizing and storing standard character and numeric-based information, the engine can, with the appropriate access and manipulation routines, manage non-traditional data structures that are either modeled more like the business environment or contain new types of information never before available for business application processing.

Though the data may be considered "nonstandard," and some types can be table-like in and of themselves, it is stored in a relational manner using tables, columns and rows. In addition, all data, data structures created through Data Definition Language (DDL) commands, and access routines recognize objected-oriented behaviors such as overloading, inheritance and polymorphism.

This object-relational extensibility supports transactional consistency and data integrity while simplifying database optimization and administration.

Other database management systems (DBMS) rely on middleware to link multiple servers, each managing different data types, to make it look as though there is a single processing environment. This approach compromises not only performance, but also transactional consistency and integrity because problems with the network can corrupt the data. This is not the case with IBM IDS. Its object-relational technology is built into the DSA core and can be used, or not, at will within the context of a single database environment.

Data types

IBM IDS uses a wide range of data types to store and retrieve data. The breadth and depth of the data types available to the database administrator and application developer is significant allowing them to truly define data structures and rules that accurately mirror the business environment rather than trying to approximate it through normalized database design and access constraints.

Some types, referred to as built-in types, include standard data representations such as character(n), decimal, integer, serial, varchar(n), date, and datetime, alias types such as money, and simple large objects (LOBs). IBM has also added additional built-in types to recent releases of IBM IDS, including boolean, int8, serial8 and an even longer variable length character string, the lvarchar.

Extended data types themselves are of two classes, including:

- Super-sets of built-in data types with enhanced functionality
- Types that were not originally built into the IBM Informix database engine but that, once defined, can be used to intelligently model data objects to meet business needs.

Extended data types can be used in queries or function calls, passed as arguments to database functions, indexed and optimized in the same way as the core built-in data types.

Since any data that can be represented in C or Java can be natively stored and processed by the engine, IBM IDS can encapsulate applications that have already implemented data types as C or Java structures.

Because the definition and use of extended data types is built into the DSA architecture, specialized access routines support high performance. The access routines are fully and automatically recoverable, and they benefit from the proven manageability and integrity of the IBM Informix database architecture.

User-defined routines, aggregates and access methods

In earlier versions of the IDS engine, developers and administrators who wanted to capture application logic that manipulated data and have it execute within the engine only had stored procedures to work with. Although stored procedures have an adequate amount of functionality, they may not optimize performance.

IDS provides the ability to create significantly more robust and higher performing application or data manipulation logic in the engine where it can benefit from the processing power of the physical server and the DSA.

A “user-defined routine” (UDR) is a collection of program statements that - when invoked from an SQL statement, a trigger, or from another UDR - perform new domain-specific operations, such as searching geographic data or collecting data from Web site visitors. UDRs are most commonly used to execute logic in the engine, either generally useful algorithms or business specific rules, reducing the time it takes to develop applications and increasing the applications’ speed.

UDRs can be either functions that return values or procedures that do not. They can be written in IBM Informix Stored Procedure Language (SPL), C or Java. SPL routines contain SQL statements that are parsed, optimized and stored in the system catalog tables in executable format - making SPL ideal for SQL-intensive tasks. Since C and Java are powerful, full-function development languages, routines written in these languages can carry out much more complicated tasks than SPL routines. C routines are stored outside the engine with the path name to the shared library file registered as the UDR. Java routines are first collected into “jar” files, which are stored inside the engine as “smart large objects”

(SLOs). Regardless of their storage location, C and Java routines execute as though they were a built-in component of the engine.

A “user-defined aggregate” (UDA) is a UDR that can either extend the functionality of an existing built-in aggregate (for example, SUM or AVG) or provide new functionality that was not previously available. Generally speaking, aggregates return summarized results from one or more queries. For example, the built-in SUM aggregate adds values of certain built-in data types from a query result set and returns their total. An extension of the SUM aggregate can be created to include user-defined data types, enabling the reuse of existing client application code without requiring new SQL syntax to handle the functionality of new data types within the application. To do so, using the example of the SUM aggregate, would require creating (and registering) a user-defined function that would overload the “plus” function and take the user-defined data types, which needed to be added together, as input parameters.

DataBlades

IBM Informix DataBlade modules bring additional business functionality to the engine through specialized user-defined data types, routines and access methods. Developers can use these new data types and routines to more easily create and deploy richer applications that better address a company’s business needs. IBM IDS provides the same level of support to DataBlade functionality that is accorded to built-in or other user-defined types/routines. With IBM Informix DataBlade modules, almost any kind of information can be easily managed as a data type within the engine. There is a portfolio of third-party DataBlade modules, or developers can use the IBM Informix DataBlade Developer’s Kit (DBDK) to create specialized blades for a particular business need.

Within the context of this redbook the following IBM Informix DataBlades and Bladelets will be of primary interest:

- WebSphere MQ DataBlade
- Web DataBlade
- XSLT DataBlade
- XML generating UDRs DataBlade

The DBDK is a single development kit for Java-, C- and SPL-based DataBlades and the DataBlade application programming interface.

The DataBlade API is a server-side “C” API for adding functionality to the database server, as well as for managing database connections, server events, errors, memory and processing query results.

9.2 IDS as a Web services provider

There are multiple options to utilize IDS as a Web services provider. Those options heavily depend on your development environment, programming language preferences and deployment platforms.

In the following sections we will cover the most common and likely also most interesting Web services approaches for IDS developers and users.

9.2.1 IDS Web services based on Enterprise Java Beans (EJBs)

This is a very straightforward process. In order to access an IDS based entity bean, you create a stateless session bean first and then use the Web services wizard in the Rational SDP to generate the necessary code for accessing the session bean.

Since these kind of Web services are more or less database independent due to the intermediate abstraction layer (session and entity beans) we did not include any example in this redbook, but instead refer you to another redbook which covers this topic in great detail: *Self-Study Guide: WebSphere Studio Application Developer and Web Services*, SG24-64077.

9.2.2 IDS and simple Java Beans Web services

Using Java beans for IDS Web services is a very flexible and simple approach. The Java bean could contain either Informix JDBC calls to the database, IBM's data access bean code (a different abstraction layer to a pure JDBC application), calls to the SQLToXML and XMLToSQL class libraries, or even Java bean code which has been generated by any other 3rd party Java development environment.

9.2.3 IDS and EGL Web services

Those developers who have the need to develop robust Web services on top of IDS, but who do not want to use pure Java technology to achieve that goal, should look at IBM Enterprise Generation Language and its very powerful Web services support.

For an introduction in the EGL language and examples on how to use the EGL Web services capabilities, refer to 5.5, "Enterprise Generation Language (EGL) and SOA" on page 126.

Also refer to 9.6, "Integrating I4GL applications with SOA through EGL" on page 344 to learn how to modernize an existing Informix 4GL application to utilize Web services through the conversion into EGL.

9.2.4 IDS and WORF (DADX Web services)

The document access definition extension (DADX) Web services had been originally developed with IBM DB2 and its XML Extender in mind. It allows you to easily wrap IBM DB2 XML Extender or regular SQL statements inside a Web service.

Fortunately, the non XML Extender related operations also work without any problems when using IBM Informix IDS[1]. Let us look at the supported DADX functions for IDS:

- Query
- Insert
- Update
- Delete
- Call Stored Procedures (limited support for IDS 7[2])

The runtime component of DADX Web services is called Web Services Object Runtime Framework (WORF). WORF uses the SOAP protocol and the DADX files and provides the following features:

- Resource based deployment and invocation
- Automatic service redeployment, at development time, when defining resource changes
- HTTP GET and POST bindings, in addition to SOAP
- Automatic WSDL and XSD generation, including support for UDDI Best Practices

[1] WORF has been already certified against IDS 10.x

[2] In IDS 7 you can only call stored procedures which do not return any results!

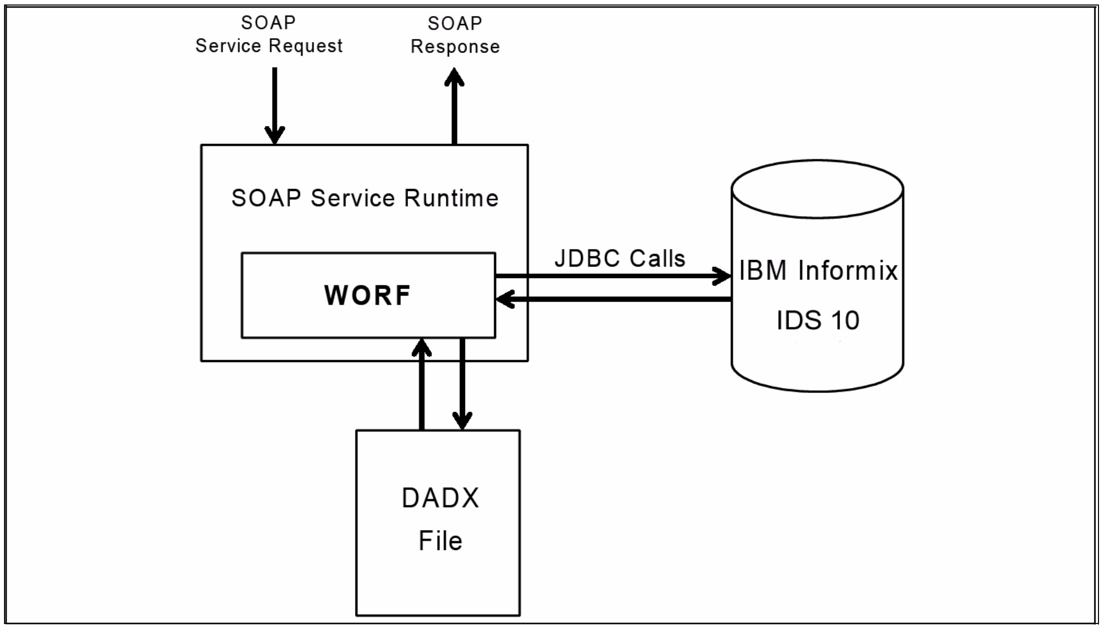


Figure 9-3 How WORF and IDS work together

So how does WORF handle a Web service request in combination with IDS?

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

   The URL of the request specifies a DADX or DTD file, and the requested action, which can be a DADX operation or a command, such as TEST, WSDL, or XSD. A DADX operation can also contain input parameters. WORF performs the following steps in response to a Web service request:

2. Loads the DADX file specified in the request.
3. Generates a response, based on the request.

   For operations:

   – Replaces parameters with requested values
   – Connects to IDS and runs any SQL statements, including UDR calls
   – Formats the result into XML, converting types as necessary

   For commands:

   – Generates necessary files, test pages, or other responses required.

4. Returns the response to the service requestor.

Since it is so easy to implement an IDS based Web service with DADX Web services, in the next section we look at how to develop such a service.

How to build a DADX Web service with IDS

In order to build a DADX Web service, we need to have some SQL statements on which the service should be based. In this section we are also assuming that the reader knows how to use the Rational development tools. For the following examples we are using the Rational Application Developer 6.0.1.1 (RAD).

Before we can actually build the DADX Web service we need to create a new Dynamic Web Project and define some SQL Statements. To keep the example simple, we will only define two statements, one SELECT statement and one INSERT statement.

Create a Web project and a connection to the database

You should first start RAD, then define a new workspace (or choose an existing one), and create a new Web Project, by selecting **File** → **New** → Dynamic Web Project.

Give the new Web project a name and optionally choose an associated page template while navigating through the project wizard. In our example we'll name the project **InformixSOADemo**. In the ProjectExplorer window you should see two project folders, one for the actual Web project (located in the Dynamic Web Projects folder) and a related Enterprise Archive (EAR) project with a similar name in the Enterprise Applications folder.

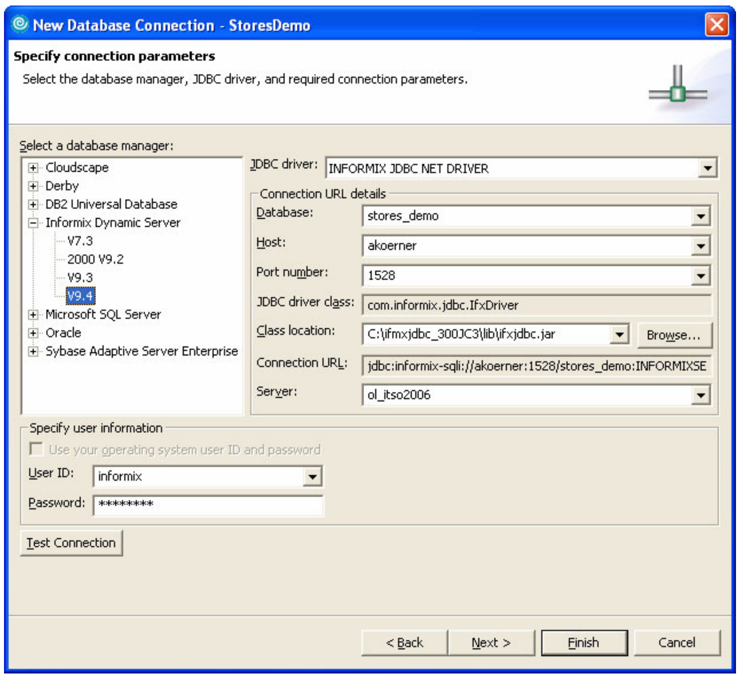
In our next step we define a connection to the standard IDS stores demo database and create a simple SELECT statement to select all customer data from the demo database.

Switch to the RAD Data view by selecting **Window** → **Open Perspective** → **Data**. Right-click into the Database Explorer window and select **New Connection**.

On the first screen on the New Database Connection wizard, select Choose a database manager and JDBC driver and provide a connection name, for example, StoresDemo.

In the New Database Connection fill in the correct connection properties to connect to the stores_demo database. Choose the appropriate Informix Dynamic Server version in the **Select a database manager** field. For IDS 10.x you should use **Informix Dynamic Server, V9.4** and the JDBC driver Class location should point to a JDBC driver jar file of Version 3.30.JC1 or higher.

The database connection window is shown in Figure 9-4.



New Database Connection - StoresDemo
Specify connection parameters
Select the database manager, JDBC driver, and required connection parameters.
Select a database manager:
Cloudscape
Derby
DB2 Universal Database
Informix Dynamic Server
V7.3
2000 V9.2
V9.3
V9.4
Microsoft SQL Server
Oracle
Sybase Adaptive Server Enterprise
Connection URL details
Browse...
Specify user information
Use your operating system user ID and password
Test Connection
< Back
Next >
Finish
Cancel

JDBC driver: INFORMIX JDBC NET DRIVER

Database: stores_demo

Host: akoerner

Port number: 1528

JDBC driver class: com.informix.jdbc.IfxDriver

Class location: C:\ifmxjdbc_300JC3\lib\ifxjdbc.jar

Connection URL: jdbc:informix-sqli://akoerner:1528/stores_demo:INFORMIXSE

Server: ol_itso2006

User ID: informix

Password: *********

Figure 9-4 The New Database Connection wizard window with IDS settings

Before you proceed its recommended to use the **Test Connection** button to verify that all connection details are correct. On the following wizard screens your have some options include and exclude some tables from underlying database schema.

Tip: If your database and its tables are owned by user informix, you might want to uncheck the “Schema NOT LIKE informix” option on the second wizard screen or otherwise you will not be able to import the schema information of all the tables which belong to user informix.

Towards the end of the schema import from the IDS database, you will be asked if you want to copy the schema information into an existing project. After answering that question with yes, select the newly created Dynamic Web project from above (InformixSOADemo).

Define a SELECT statement by utilizing the Query Builder

In the Data Definition window, navigate to the **InformixSOADemo** → **Web Content** → **WEB-INF** → **InformixSOADemo** → **stores_demo** → **Statements** folder. Right-click the **Statements** folder and choose **New** → Select Statement. Name the Statement, for example, selectOneCustomer and click **OK**.

Now you should see the interactive query builder. In the tables window, you need to select (by right-clicking) the tables you want to include in the query. In our demo we select only the table **informix.customer**, because we want to show the complete customer information. Since you want to include all attributes from the customer table, select all customer attributes in the table attribute check boxes.

We want to select only one customer, therefore we need to define a WHERE condition. To do this, select the **Conditions tab** in the lower window of the query builder. Select informix.customer.customer_num as the column, and choose = as the operator. We also need to provide a host variable, which acts as a placeholder for different customer_num values later in the process. Let us name the host variable **:customernum** (the colon is important!)

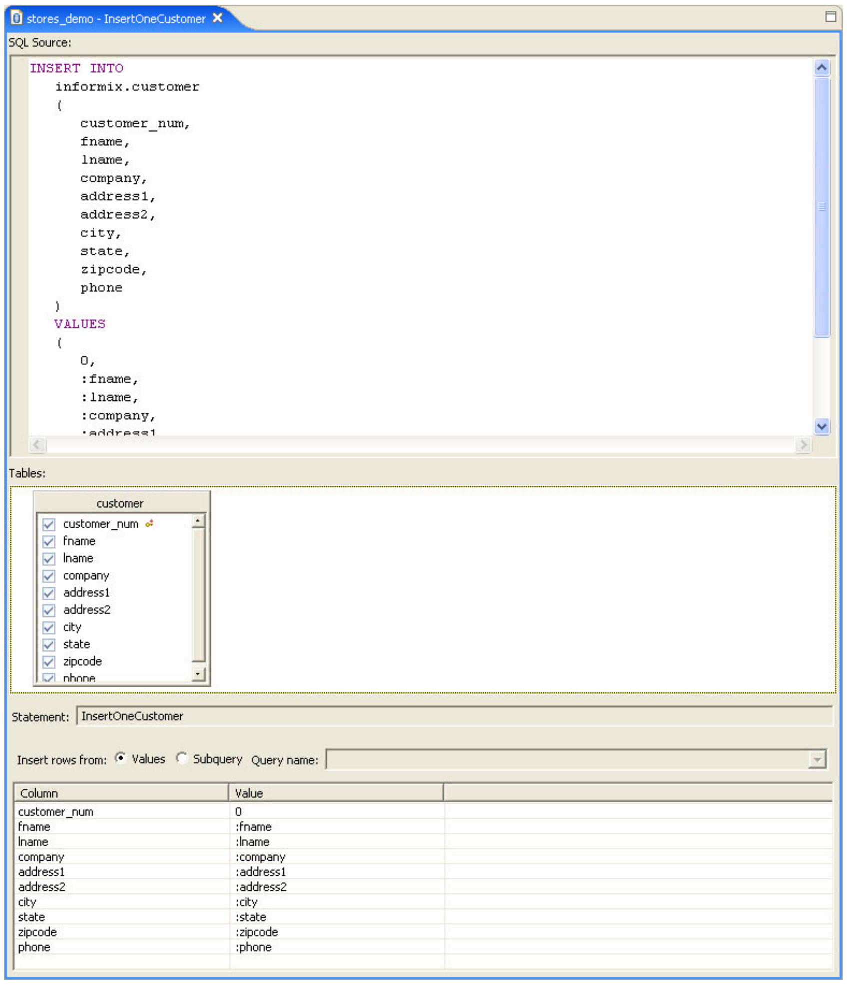
Now save your statement into the Web project by selecting **File** → **Save stores_demo - selectOneCustomer**.

Define an INSERT Statement for the demo DADX Web service

Switch to the Data perspective by selecting **Window** → **Open** Perspective → **Data**. In the **Data Definition** window open the **InformixSOADemo/Web Content/InformixSOADemo/stores_demo** folder. Right-click the **Statements** folder and select **New** → Insert Statement. Name the new statement **InsertOneCustomer** and click **OK**.

In the interactive SQL builder window, right-click into the **Tables** window. Select **Add Table** and then from the tables selection menu, select the **informix.**customer table. Within the informix.customer table, select all attributes for the INSERT statement. In the window below the **Tables** window, we now have to define the host variables as placeholders for the later inserts which should be executed against the customer table.

To make it simple we’ll name all host variables by using the column name with a colon (:) in front. To do this, click the **Value** column for each table attribute and enter the host variable name, for example, :fname for the fname attribute. Important: since the customer_num attribute is defined as a SERIAL data type in the database, we set the insert value to zero to automatically generate a new customer_num value during each insert! So eventually, the SQL builder window should look like in Figure 9-5 on page 308. As soon as you have defined the INSERT statement, save it into the demo Web project.



```
INSERT INTO
   informix.customer
   (
      customer_num,
      fname,
      lname,
      company,
      address1,
      address2,
      city,
      state,
      zipcode,
      phone
   )
   VALUES
   (
      0,
      :fname,
      :lname,
      :company,
      :address1
```

Statement: InsertOneCustomer

Insert rows from: [✓ Values] Subquery Query name:

Column	Value
customer_num	0
fname	:fname
lname	:lname
company	:company
address1	:address1
address2	:address2
city	:city
state	:state
zipcode	:zipcode
phone	:phone

Figure 9-5 The InsertOneCustomer Statement is being constructed in the SQL Builder

Create a DADX group and define its properties

In preparation for the to be generated DADX file, we need first to create a DADX group, which combines one or more SQL statement into one logical DADX Web service. So it could make sense, for example, to group all operations on the customer table into one group, while operations on the account table will be grouped into another DADX group.

Each DADX group also maintains its own database connection properties, so one could also use different DADX groups to connect to different databases or even different database servers (vendors).

To create a new DADX group:

1. Open the **Web Perspective** by selecting **Window** → **Open** Perspective → **Web**. Then select the Project Explorer window.
2. Select **File** → **New** → **Other** → Web Services → **Web Service DADX Group Configuration**. Click **Next**.
3. In the next window, select the **InformixSOADemo** folder and then click **Add group**.
4. For the group name, enter **ITSOCustomerService**. Click **OK**.
5. While still being in the same window, now select the **InformixSOADemo/ITSOCustomerService** folder and then click **Group properties**.

6. In the DADX Group Properties pop-up window, fill in the following information:

   DB driver: `com.informix.jdbc.IfxDriver`

   DB URL:
   `jdbc:informix-sqli://akoerner:1528/stores_demo:INFORMIXSERVER=ol_itso2006;user=informix;password=informix`
7. Leave the other fields as-is. Click **OK**.
8. In the **DADX Group Configuration** window, click Finish.

Generate the DADX file

Now we can generate the DADX file for the two SQL Statements. To do this:

1. Select **File** → **New** → **Other** → Web Services → **DADX File**. Click Next.
2. In the Create DADX window select InformixSOADemo as the project and the ITSOCustomerService DADX Group. As a filename enter ITSOCustomerService.dadx. Also select the Generate a DADX file from a list of SQL queries or Stored Procedures. Click **Next**.
3. In the **Select SQL Statements and Stored Procedures** window, open the **InformixSOADemo/Web Content/InformixSOADemo/stores_demo/Statements** folder
4. Since we want to select both SQL statements (insertOneCustomer, selectOneCustomer) we need to do the following: Click the **insertOneCustomer** statement first and then control-click the **selectOneCustomer** too. Now both statements should be selected (highlighted). Click **Next**.
5. Just click **Next** in the **Select DAD Files** window since DAD files are not yet supported with IBM Informix IDS.
6. In the DADX operations window click **Finish**.

The generated ITSOCustomerService.dadx file should look the one in Example 9-1. Notice the XML compliant format and the specific DADX keywords.

Example 9-1 The ITSOCustomerService.dadx file

```
<?xml version="1.0" encoding="UTF-8"?>
<dadx:DADX xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx dadx.xsd">
    <dadx:operation name="selectOneCustomer">
        <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
            <![CDATA[
            ]]>
        </dadx:documentation>
        <dadx:query>
            <dadx:SQL_query>
                <![CDATA[
                SELECT * FROM informix.customer WHERE
informix.customer.customer_num = :customernum
                ]]>
            </dadx:SQL_query>
            <dadx:parameter name="customernum" type="xsd:int"/>
        </dadx:query>
    </dadx:operation>
    <dadx:operation name="InsertOneCustomer">
        <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
```

```
            <![CDATA[

            ]]>
        </dadx:documentation>
        <dadx:update>
            <dadx:SQL_update>
                <![CDATA[
                INSERT INTO informix.customer ( customer_num, fname, lname,
company, address1, address2, city, state, zipcode, phone ) VALUES ( 0, :fname,
:lname, :company, :address1, :address2, :city, :state, :zipcode, :phone )
                ]]>
            </dadx:SQL_update>
            <dadx:parameter name="fname" type="xsd:string"/>
            <dadx:parameter name="lname" type="xsd:string"/>
            <dadx:parameter name="company" type="xsd:string"/>
            <dadx:parameter name="address1" type="xsd:string"/>
            <dadx:parameter name="address2" type="xsd:string"/>
            <dadx:parameter name="city" type="xsd:string"/>
            <dadx:parameter name="state" type="xsd:string"/>
            <dadx:parameter name="zipcode" type="xsd:string"/>
            <dadx:parameter name="phone" type="xsd:string"/>
        </dadx:update>
    </dadx:operation>
</dadx:DADX>
```

Since the interactive query builder in RAD only supports SELECT, INSERT, UPDATE and DELETE statements you might have to edit the generated DADX file manually if you want to add support for IDS user defined routines (UDR).

Create a DADX Web service based on the generated DADX file

Now let us generate the necessary files for a DADX Web service based on the DADX file we generated in the previous section.

First we need to prepare the InformixSOADemo Web project for Informix IDS database access in combination with the DADX Web service:

1. In the Project Explorer right-click the InformixSOADemo project folder and then select **Properties**.
2. In the **Properties** window select **Java Build Path** and then the **Libraries** tab.
3. Now add the Informix JDBC driver to the Class Path entries by clicking **Add External JARs**. In the file browser select the correct **ifxjdbc.**jar file and the **ifxjdbcx.**jar file and click **Open**.
4. Close the Properties window by clicking **OK**.

Now we build the Web service itself:

1. Open the Web perspective and in the InformixSOADemo Web project, click the file **Java Resources/JavaSource/groups.ITSOCustomerService/ITSOCustomerService.dadx**
2. Select **File** → **New** → **Other** → Web Services → **Web Service**. Click Next.
3. In the Web Services window select as the **Web service type**: DADX Web Services. In addition, check the **Start Web service in Web project** option and the options, **Overwrite files without warning** and **Create folders when necessary**. Click Next.
4. In the Object Selection Page verify that have selected the correct DADX file. In our example it is:

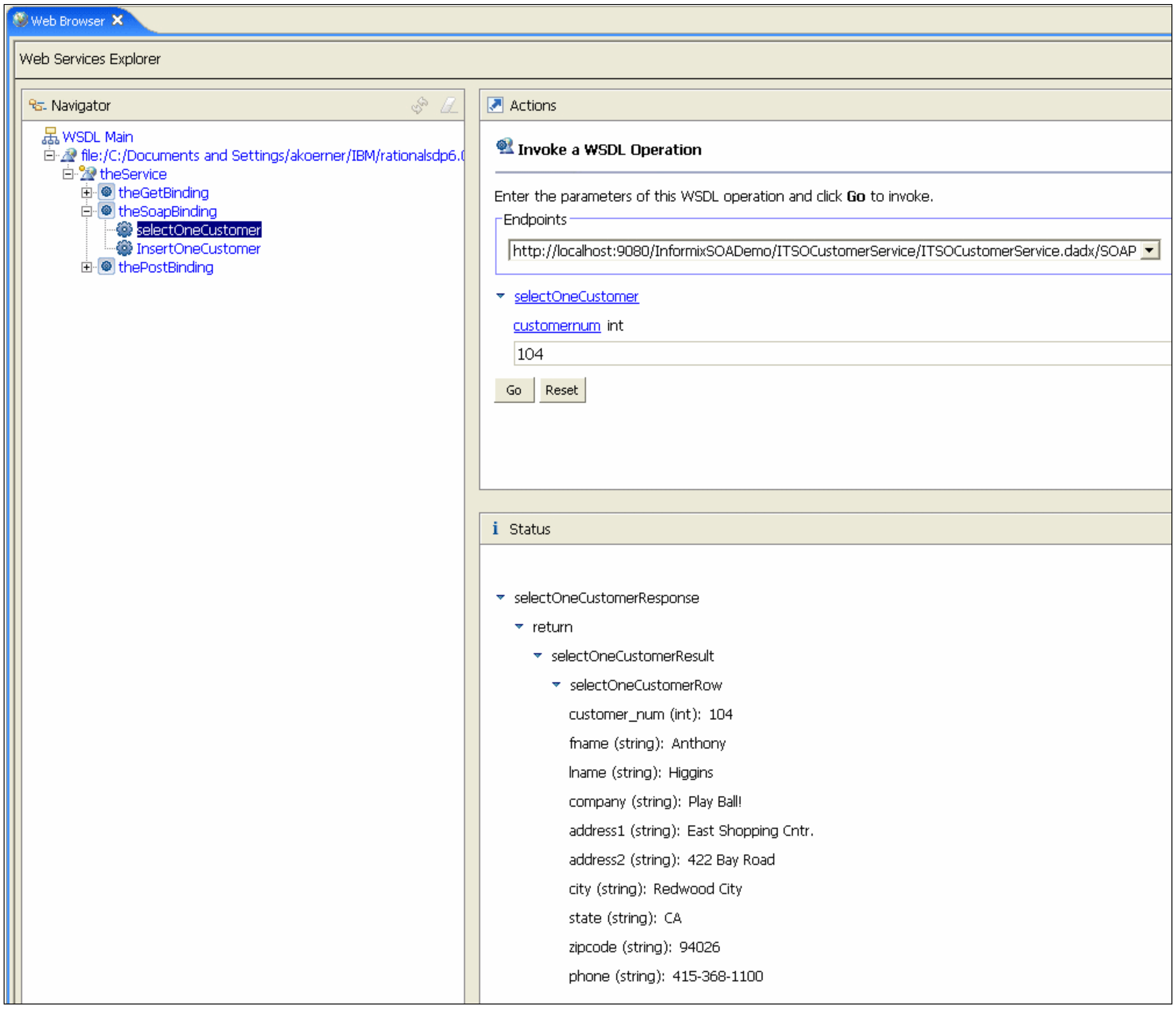
/InformixSOADemo/JavaSource/groups/ITSOCustomerService/ITSOCustomerService.dadx.

Click **Next.**

5. In the **Service Deployment Configuration** leave the default values and as the **Service project** choose **InformixSOADemo** and as the EAR project choose **InformixSOADemoEAR**. Click **Next**.
6. In the Web Services DADX Group properties verify the database connection information. Click **Next.**
7. In the Web Service Publication window **click Finish.**

Let's test the newly created Web service with the built-in RAD Web services Test client:

1. In the Project Explorer window, in the InformixSOADemo Web project folder, locate the **WebContent/wsdl/ITSOCustomerService/ITSOCustomerService.wsdl** file. Right-click this file and the select **Web Services → Test with Web Services Explorer.**
2. While in the Web Services Explorer select theService → theSOAPBinding → selectOneCustomer.
3. In the Actions window enter a valid value (like 104) for the customer_num value and click **GO**. You see the result in Figure 9-6 on page 312.



Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://localhost:9080/InformixSOADemo/ITSOCustomerService/ITSOCustomerService.dadx/SOAP

selectOneCustomer

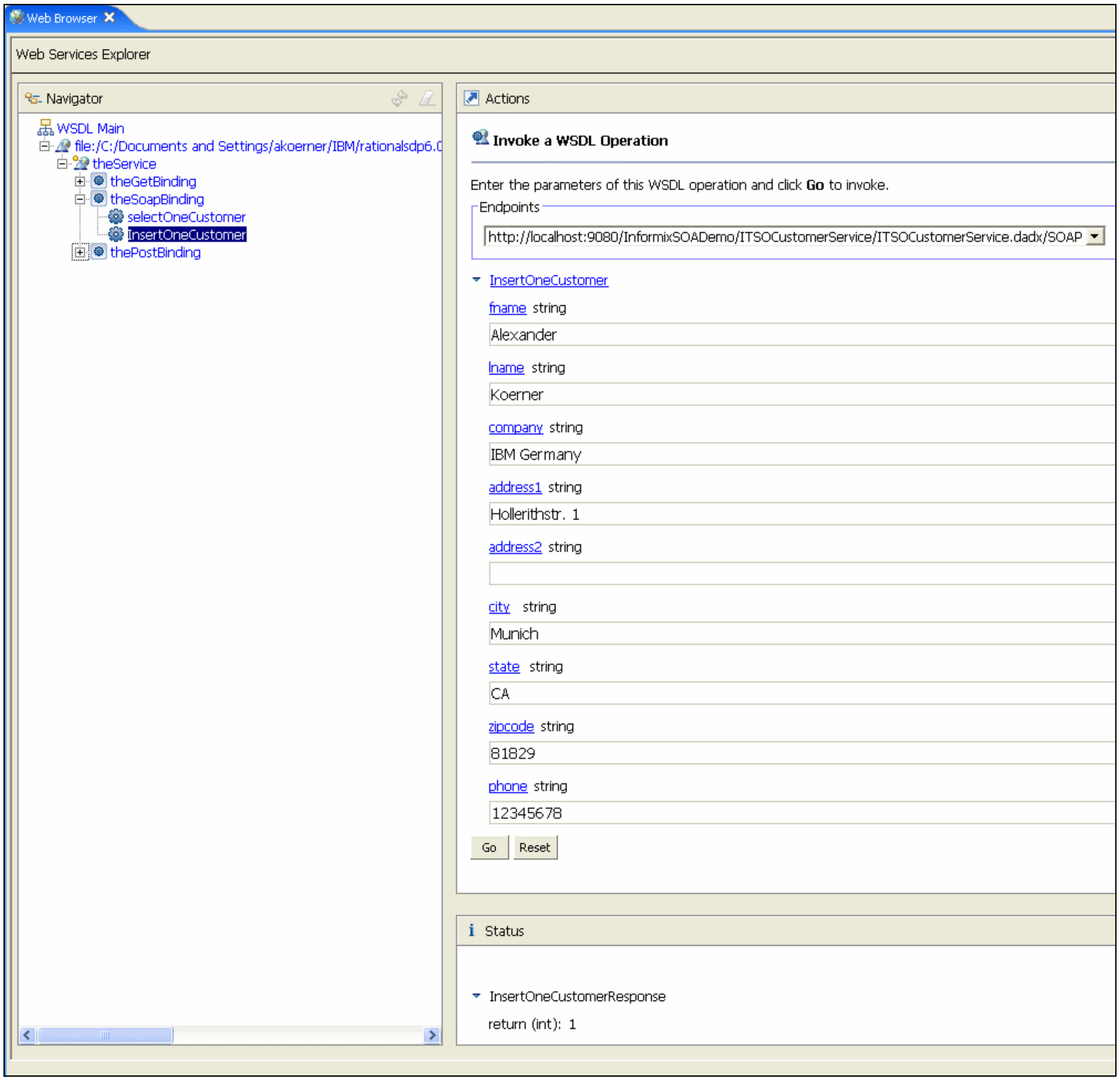
customernum int

104

```
selectOneCustomerResponse
  return
    selectOneCustomerResult
      selectOneCustomerRow
        customer_num (int): 104
        fname (string): Anthony
        lname (string): Higgins
        company (string): Play Ball!
        address1 (string): East Shopping Cntr.
        address2 (string): 422 Bay Road
        city (string): Redwood City
        state (string): CA
        zipcode (string): 94026
        phone (string): 415-368-1100
```

Figure 9-6 Testing the selectOneCustomer DADX service with RAD Web Services Explorer

4. Now your can also try the InsertOneCustomer Web service. In this case you need to provide some values for the customer fields (fname, lname, and so on.). The Web services explorer is shown in Figure 9-7 on page 313.



Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints
http://localhost:9080/InformixSOADemo/ITSOCustomerService/ITSOCustomerService.dadx/SOAP

InsertOneCustomer

fname string	Alexander
lname string	Koerner
company string	IBM Germany
address1 string	Hollerithstr. 1
address2 string	
city string	Munich
state string	CA
zipcode string	81829
phone string	12345678

InsertOneCustomerResponse
return (int): 1

Figure 9-7 The InsertOneCustomer DADX Web service

DADX support for user defined routines / stored procedures

In addition to standard SQL statements like SELECT, INSERT, DELETE and UPDATE, the WORF framework also supports the execution of user defined routines or stored procedures in IDS. In order to do this, the framework utilizes (internally) the JDBC CallableStatement class which is a very portable way of calling stored procedures and functions in database servers.

Since this feature is unfortunately not supported through the interactive SQL builder in RAD, we need to either create a new DADX file or modify an existing one.

Before we go ahead with an example, let us look at the DADX file syntax for stored procedures/functions based on the XML schema for DADX files (see Example 9-2 on page 314).

Example 9-2 DADX call operation (XML schema definition)

```
<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>
```

As mentioned earlier, the WORF framework utilizes the JDBC java.sql.CallableStatement interface for the execution of IDS user defined routines. Therefore the syntax for calling routines in IDS this way should follow the JDBC guidelines. For a simple example in DADX syntax how to call a user defined routine which does not return any results, see Example 9-3.

Example 9-3 Simple UDR call in DADX syntax

```
<dadx:operation name="createOneCustomerSimple">
   <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
   </dadx:documentation>
   <dadx:call>
      <dadx:SQL_call>
<![CDATA[
            { call create_customer_simple (:fname, :lname, :company, :address1, :address2,
:city, :zipcode, :state, :phone)}
]]>
      </dadx:SQL_call>
      <dadx:parameter name="fname" type="xsd:string" kind="in"/>
      <dadx:parameter name="lname" type="xsd:string" kind="in"/>
      <dadx:parameter name="company" type="xsd:string" kind="in"/>
      <dadx:parameter name="address1" type="xsd:string" kind="in"/>
      <dadx:parameter name="address2" type="xsd:string" kind="in"/>
      <dadx:parameter name="city" type="xsd:string" kind="in"/>
      <dadx:parameter name="zipcode" type="xsd:string" kind="in"/>
      <dadx:parameter name="state" type="xsd:string" kind="in"/>
      <dadx:parameter name="phone" type="xsd:string" kind="in"/>
   </dadx:call>
</dadx:operation>
```

If you need to return results back to the DADX Web service consumer you might have different options:

- Starting with IDS 9 you can utilize multiple out parameters in the UDR parameter list. To use these out parameters in combination with DADX you need to declare them as in/out parameters and the Web service caller might have to supply dummy values (for example, zero for integer types) to make it work. This behavior seems to be IDS specific and does not apply to other databases. The UDR create_customer_out in Example 9-4 shows a simple SPL UDR which uses one out parameter (newcustomernum).

Example 9-4 IDS UDR with an out parameter (in SPL)

```
create procedure create_customer_out (fname lvarchar, lname lvarchar,
                company lvarchar, address1 lvarchar, address2 lvarchar,
                city lvarchar, zipcode lvarchar, state lvarchar,
                phone lvarchar, OUT customernum int)

   define new_customernum int;
   insert into customer values (0, fname, lname, company, address1,
                                       address2, city, state, zipcode, phone);
   let new_customernum = dbinfo('sqlca.sqlerrd1');
   let customernum = new_customernum;

end procedure;
```

Example 9-5 shows the correct DADX syntax for calling such a UDR. Notice the in/out option for the newcustomernum parameter.

Example 9-5 DADX syntax fragment for the IDS UDR from Example 9-4

```
<dadx:operation name="createOneCustomer">
   <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
   </dadx:documentation>
   <dadx:call>
      <dadx:SQL_call>
<![CDATA[
         { call create_customer_out (:fname, :lname, :company, :address1, :address2,
:city, :zipcode, :state, :phone, :newcustomernum)}
]]>
      </dadx:SQL_call>
      <dadx:parameter name="fname" type="xsd:string" kind="in"/>
      <dadx:parameter name="lname" type="xsd:string" kind="in"/>
      <dadx:parameter name="company" type="xsd:string" kind="in"/>
      <dadx:parameter name="address1" type="xsd:string" kind="in"/>
      <dadx:parameter name="address2" type="xsd:string" kind="in"/>
      <dadx:parameter name="city" type="xsd:string" kind="in"/>
      <dadx:parameter name="zipcode" type="xsd:string" kind="in"/>
      <dadx:parameter name="state" type="xsd:string" kind="in"/>
      <dadx:parameter name="phone" type="xsd:string" kind="in"/>
      <dadx:parameter name="newcustomernum" type="xsd:int" kind="in/out"/>
   </dadx:call>
</dadx:operation>
```

Tip: This restriction seems to be specific to the DADX/IDS combination, since a similar restriction had been already removed since the IBM Informix JDBC 2.21.JC4 driver and is no longer valid. Callers need to use registerOUTparameter() only and they do not need to use setXXX() method on OUT parameters. A future version of DADX will very likely address this change in the Informix JDBC driver.

- You could simply return a result for a UDR or even complete results sets. See the following important tip regarding the support in IDS for that feature.

Tip: IDS 10 supports a feature which allows that the columns of a result set for a user defined routine can have display labels. The WORF framework requires the usage of those labels in IDS or one could not use UDRs with result sets.

For information about how the DADX syntax for an UDR and what a result set should look like, look at the SPL UDR in Example 9-6 and the associated DADX syntax in Example 9-7. Notice the display label syntax in the stored procedure (`returning ... as ...`) and also the result_set definition and usage in the DADX file fragment.

Example 9-6 IDS stored procedure with display labels for the result set

```
create procedure read_address  (lastname char(15))
        returning char(15) as pfname, char(15) as plname,
                char(20) as paddress1, char(15) as pcity,
                char(2)  as pstate, char(5) as pzipcode;
   define p_fname, p_city char(15);
   define p_add char(20);
   define p_state char(2);
   define p_zip char(5);
   select fname,  address1, city, state, zipcode
      into p_fname,  p_add, p_city, p_state, p_zip
      from customer
      where lname = lastname;
   return p_fname, lastname, p_add, p_city, p_state, p_zip;
end procedure;
```

Example 9-7 shows the DADX syntax associated to Example 9-6.

Example 9-7 DADX syntax fragment for the UDR in

```
<dadx:result_set_metadata name="customerAddress" rowName="customer1">
   <dadx:column name="pfname" type="VARCHAR" nullable="true" />
   <dadx:column name="plname" type="VARCHAR" nullable="true" />
   <dadx:column name="paddress1" type="VARCHAR" nullable="true" />
   <dadx:column name="pcity" type="VARCHAR" nullable="true" />
   <dadx:column name="pstate" type="VARCHAR" nullable="true" />
   <dadx:column name="pzipcode" type="VARCHAR" nullable="true" />
</dadx:result_set_metadata>

<dadx:operation name="readOneCustomer">
   <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
   </dadx:documentation>
   <dadx:call>
      <dadx:SQL_call>
<![CDATA[
             { call read_address (:lname) }
]]>
      </dadx:SQL_call>
      <dadx:parameter name="lname" type="xsd:string" kind="in"/>
      <dadx:result_set name="customer1" metadata="customerAddress" />
   </dadx:call>
</dadx:operation>
```

Important: The result set metadata definitions (<dadx:result_set_metadata> </dadx:result_set_metadata> tag) are global to the DADX and must precede all of the operation definition elements in a DADX file

9.2.5 IDS and other Web services environments (.NET, PHP)

Developers have options on choosing the development environment to use for their Web services development. Some might prefer the Java environment, while others prefer .NET or other powerful frameworks like PHP. The IBM Informix database development tools support all major development environments including .NET and PHP database access. For an introduction into PHP Web services programming, refer to Chapter 16, "PHP client design" on page 507.

IBM Informix .NET Provider

The IBM Informix .NET Provider is a .NET assembly that lets .NET applications access and manipulate data in IBM Informix databases. It does this by implementing several interfaces in the Microsoft .NET Framework that are used to access data from a database. See Figure 9-8.

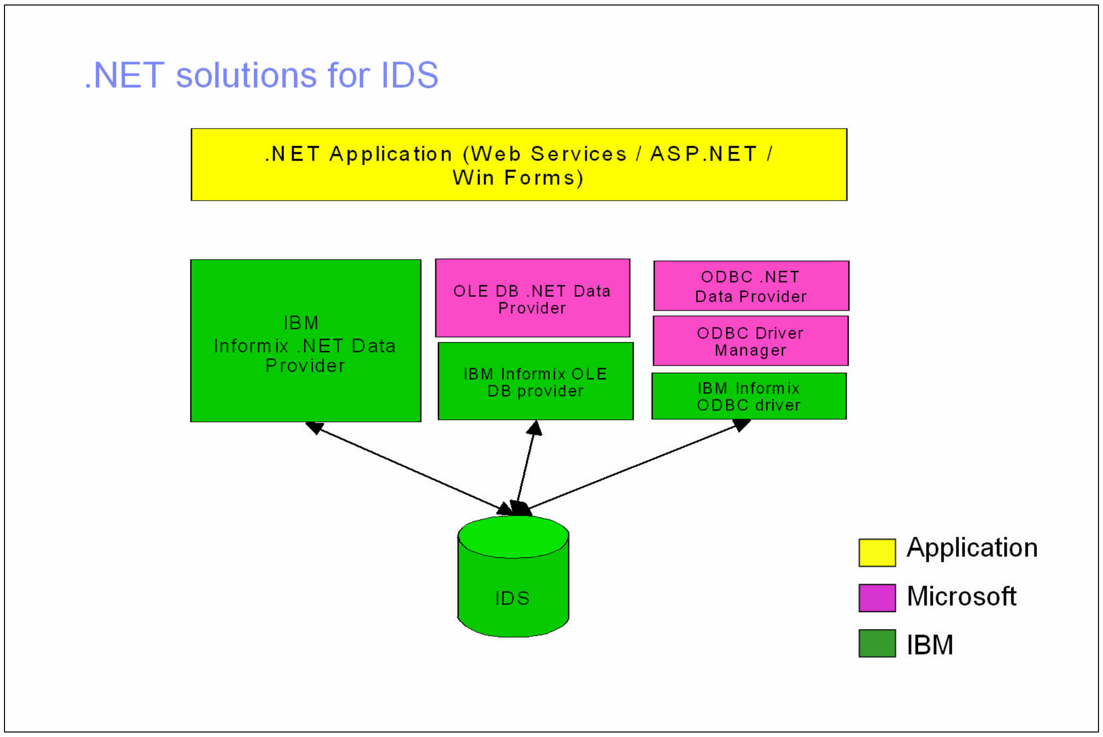

.NET solutions for IDS
.NET Application (Web Services / ASP.NET / Win Forms)
IBM Informix .NET Data Provider
OLE DB .NET Data Provider
IBM Informix OLE DB provider
ODBC .NET Data Provider
ODBC Driver Manager
IBM Informix ODBC driver
IDS
Application
Microsoft
IBM

Figure 9-8 Available .NET integration options for IDS developers

Using the IBM Informix .NET Provider is more efficient than accessing the an IBM Informix database through either of these two methods:

- Using the Microsoft .NET Framework Data Provider for ODBC along with the IBM Informix ODBC Driver

- Using the Microsoft .NET Framework Data Provider for OLE DB along with the IBM Informix OLE DB Provider

Example 9-8 shows IDS .NET provider based sample code.

Example 9-8 IDS .NET provider based sample code

```
using IBM.Data.Informix;

IfxConnection MyConn = new IfxConnection();

MyConn.ConnectionString = "Server=ids10; DataBase=db;";

MyConn.Open();

IfxCommand MyCmd = MyConn.CreateCommand();

MyCmd.CommandText = “INSERT INTO customer(custID) VALUES (101)”;

MyCmd.ExecuteNonQuery();
```

IBM IDS native PDO Driver for PHP 5

An application programing interface for rapid, cost-effective Web development using PHP Data Objects. IBM's IDS PDO driver supports PHP 5 so developers can take advantage of IDS's object-relational capabilities for developing high-performance applications.

Features of the native PDO driver for IDS:

- Higher performance; stress tested
- PECL extension built on top of IDS ODBC driver

  ```
  bash$ ./configure --with-pdo-informix=/path/to/SDK[,shared]
  ```
- Connecting to IDS Databases using PDO_INFORMIX DSN
- PHP 5 only

Example 9-9 shows a simple PDO_INFORMIX application.

Example 9-9 A simple PDO_INFORMIX application

```
$db = new PDO($dsn,$user,$pass);

$sql = "SELECT name, breed FROM ANIMALS WHERE weight < ?";

$stmt = $db->prepare( $sql );

$res = $stmt->execute( array(10) );

if( $res )
   {
      while( $row = $stmt->fetch( PDO::FETCH_BOTH ) )
      {
         print "{$row['NAME']} is a {$row['BREED']}.\n";
      }
   }
```

The most recent version of the IDS PDO_INFORMIX driver can be downloaded from here:

`http://pecl.php.net/package/PDO_INFORMIX`

9.3 IDS as a Web services consumer

In the previous sections we described in detail how to use the different tools to enable IBM Informix IDS as a Web services provider.

Now we want to focus on IDS as a Web services consumer.

This section is intended as a how-to guide to use IDS 10 as a Web service consumer. It requires a basic knowledge of the Java language, for example you should know how to edit and compile a Java program. You should also have a basic understanding of the IDS 10 extensibility features.

Why have IDS as a Web services consumer?

In addition to provide Web services, it can be very interesting for an application developer to integrate existing Web services. Those Web services could be either special B2B scenarios or public accessible services like currency conversion, stock ticker information, news, weather forecasts, search engines, and many more. Wouldn't it be great to have dynamic access to an official currency conversion service on a database level if the application needs to deal with this information? Or if an application wants to relate actual business data stored in an IDS database against news from news agencies?

Sources for public accessible Web services are, for example:

`http://www.webservicelist.com`
`http://www.xmethods.net`

Web services rely on very simple open standards like XML and SOAP and be accessed through any kind of client application. Typically those applications are written in Java, C++, or C#. For somebody who already has an existing application which is based on an SQL database and also already utilizes business logic in the database server through user defined routines, developers might want to integrate access to Web services on the SQL level.

Some of the advantages of having Web services accessible from SQL would include easy access through the SQL language and standardized APIs (for example, ODBC, JDBC), moving the Web service results closer to the data processing in the database server which could speed up applications, and providing Web service access to the non Java or C++ developers.

What are the basic Web services consumer requirements for IDS?

In order to be able to call a Web service from within IDS, you need to be able to:

- Construct a SOAP message based on a given Web service description and
- Send this SOAP message to the Web services provider via the required protocol (typically HTTP)
- Finally, be able to receive the Web service response, parse it, and handle the results on an SQL level.

All of this needs to be executed from the IDS SQL layer to achieve the required portability.

Why IDS 10 and not IDS 7?

Although IDS 7 supports stored procedures with an already very powerful stored procedure language (SPL), it is somewhat limited if there is a need, for example, to access external networks or include external libraries.

IDS10 through its very powerful DataBlade technology allows the easy integration of external routines written in C or Java into so called user defined routines (UDRs). Those UDRs can also be written in SPL. So one can say that UDRs are the generalized description of SPL, C, and Java stored procedures. In addition to the very flexible options of writing UDRs, IDS 9 also supports new data types and user defined types (UDTs).

Having these extensibility technologies available in IDS 10 in combination with the underlying, proven, high-end OLTP architecture of IDS 7 makes it a perfect choice to develop some database extensions which will provide access to Web services across standard network protocols.

Since you have the choice as an IDS 10 developer to either use C or Java for the development of Web service consumer routines, you could either include, for example, a C based SOAP framework or a Java based SOAP framework in your final solution.

To be as much platform independent as possible, and also to give you a kick-start on this topic, we chose the Apache AXIS Java framework for the development of IDS 10 Web service consumer routines.

9.3.1 Utilizing J/Foundation and Apache's Axis for Web services consumption

IDS 9 with J/Foundation enables database developer's to write server-side business logic using the Java language.

IDS 10 and J/Foundation

Java User Defined Routines (UDRs) have complete access to the leading extensible database features of the IDS 10 database. Making IDS 10 the ideal platform for Java database development.

In addition to Java UDRs, IDS conforms to the SQLJ standard for Java-stored procedures, enabling the use of the standard Java packages that are included in the Java Development Kit (JDK). Writing UDRs in Java delivers far more flexible applications that can be developed faster than C, and more powerful and manageable than stored procedure languages.

IDS with J/Foundation provides these advantages over other Java based solutions:

- Better performance and scalability
- Fully certified and optimized standard JVMs for each supported platform
- Simpler application integration
- Easy migration of existing Java applications
- Transaction control through stored data

J/Foundation is provided with IDS on many of the supported IDS 10 platforms.

Technology

IDS 10 provides the infrastructure to support Java UDRs. The database server binds SQL UDR signatures to Java executables and provides mapping between SQL data values and Java objects so that the database server can pass parameters and retrieve returned results. IDS 10 also provides support for data type extensibility and sophisticated error handling.

Java UDRs execute on specialized virtual processors called Java Virtual Processors (JVPs). IDS 10 embeds a Java Virtual Machine (JVM) in the code of each JVP. The JVPs are responsible for executing all server-based Java UDRs and applications.

Although the JVPs are mainly used for Java-related computation, they have the same capabilities as a CPU VP, and they can process all types of SQL queries. This eliminates the need to ship Java-related queries back and forth between CPU VPs and JVPs.

For more technical details of J/Foundation, refer to the IBM Informix J/Foundation Developer's Guide.

The Apache AXIS framework

So what is the Apache AXIS framework?

The Axis framework is a Java-based, open source implementation of the latest SOAP specification, SOAP 1.2, and SOAP with Attachments specification from the Apache Group. The following are the key features of this AXIS framework:

- **Flexible messaging framework**: Axis provides a flexible messaging framework that includes handlers, chain, serializers, and deserializers. A handler is an object processing request, response, and fault flow. A handler can be grouped together into chains and the order of these handlers can be configured using a flexible deployment descriptor.
- **Flexible transport framework**: Axis provides a transport framework that helps you create your own pluggable transport senders and transport listeners.
- **Data encoding support**: Axis provides automatic serialization of a wide variety of data types as per the XML Schema specifications and provides a facility to use your own customized Serializer and Deserializer.
- **Additional features**: Axis provides full support for WSDL as well as Logging, Error, and Fault Handling mechanisms.

Axis also provides a simple tool set to easily generate Java classes based on given Web service description files (WSDL) and has tools to monitor Web services.

The latest Axis distribution and more detailed information about Axis can be obtained at:

`http://ws.apache.org/axis`

9.3.2 Installation and configuration of IDS 10 and AXIS 1.3 for the examples

Tip: All of the configuration and installation information in this section is based on Windows XP, but can be easily also applied to other platforms like Linux or UNIX.

AXIS 1.3 installation and preparation

First, download the AXIS release from:

`http://ws.apache.org/axis/java/releases.html`

The release we have been using for the examples below is based on AXIS 1, Version 1.3 Final After downloading the release, extract the AXIS distribution into a directory of your choice (for example, directly into the C:\ directory). Make sure that you also extract the folder structure.

If you are finished, you should have an <install_dir>\axis-1_3 directory.

In addition to AXIS we also need a JAXP 1.1 XML compliant parser. The recommended one is the Apache Xerces: Just download the latest stable version from:

http://xml.apache.org/dist/xerces-j

(for example, Xerces-J-bin.2.5.0.zip) and extract it into a local directory (for example, C:\). Eventually you should have an <install_dir>\xerces-2_5_0 directory.

For more advanced Axis SOAP handling (for example, SOAP attachments), you also might want to download the following Java packages: jaf-1_0_2-upd2 (JavaBeans Activation Framework, http://java.sun.com/products/javabeans/glasgow/jaf.html) and javamail-1_3_3_01 (Java Mail, http://java.sun.com/products/javamail/). All the classpath settings in our examples below include the necessary activation.jar and mail.jar files out of the optional Java packages for completeness.

IDS 10 with J/Foundation configuration for AXIS

Since the AXIS Framework is Java based, we need to configure IDS 10 for Java UDRs. Before we go ahead, make sure that you're using an IDS 10 with J/Foundation. You can verify this by checking the $INFORMIXDIR/extend directory for the existence of a *krakatoa* subdirectory. If this directory is missing you do not have the correct version of IDS 10.

First, we need to enable J/Foundation for your IDS 10 instance:

1. Create an sbspace to hold the Java JAR files. The database server stores Java JAR files as smart large objects in the system default sbspace. If you do not already have a default sbspace, you must create one. After you create the sbspace, set the SBSPACENAME configuration parameter in the ONCONFIG file to the name that you gave to the sbspace.
2. Add (or modify) the Java configuration parameters in the ONCONFIG configuration file. The ONCONFIG configuration file ($INFORMIXDIR/etc/$ONCONFIG) includes the following configuration parameters that affect Java code:
   - JDKVERSION
   - JVPPROPFILE
   - JVMTHREAD
   - JVPCLASSPATH
   - JVPHOME
   - JVPJAVALIB
   - JVPJAVAVM
   - JVPLOGFILE
   - JVPARGS
   - VPCLASS

   Make sure that these parameters exist or are not un-commented. For an example ONCONFIG file fragment, see Example 9-10.

Example 9-10 J/Foundation settings for the AXIS framework in the IDS ONCONFIG file

```
VPCLASS        jvp,num=1   # Number of JVPs to start with

JVPJAVAHOMEC:\informix\extend\krakatoa\jre# JDK installation root directory
JVPHOMEC:\informix\extend\krakatoa# Krakatoa installation directory

JVPLOGFILEC:\informix\extend\krakatoa\ol_itso2006_jvp.log# VP log file
JVPPROPFILEC:\informix\extend\krakatoa\.jvpprops_ol_itso2006# JVP property file

JDKVERSION      1.4        # JDK version supported by this server

# The path to the JRE libraries relative to JVPJAVAHOME
```

```
JVPJAVALIB      \bin\

JVPJAVAVM       jsig;dbgmalloc;hpi;jvm;java;net;zip;jpeg

# Classpath to use upon Java VM start-up (use _g version for debugging)
#JVPCLASSPATH
C:\informix\extend\krakatoa\krakatoa.jar;C:\informix\extend\krakatoa\jdbc.jar
JVPCLASSPATHfile:C:\informix\extend\krakatoa\jvp_classpath

#JVPARGS -Djava.security.policy=C:\informix\extend\krakatoa\informix.policy
```

In the foregoing example, we also define the JVPCLASSPATH to point to a file in the krakatoa directory. Having an external file to contain the JVP classpath information gives us more flexibility regarding the maximal length of the JVPCLASSPATH since the length in the ONCONFIG file is otherwise limited to 256 characters. See Example 9-11 for an AXIS compliant classpath file.

Tip: In our examples we're copying the AXIS class libraries directly into the $INFORMIXDIR\extend\krakatoa directory to avoid any changes to the informix.policy file. It is probably a cleaner approach to keep the AXIS files in their original directories and adjust the informix.policy file to allow access for the J/Foundation class loader.

Example 9-11 jvp_classpath file for the AXIS integration

```
C:\informix\extend\krakatoa\krakatoa.jar;C:\informix\extend\krakatoa\jdbc.jar;C:\i
nformix\extend\krakatoa\axis.jar;C:\informix\extend\krakatoa\jaxrpc.jar;C:\informi
x\extend\krakatoa\saaj.jar;C:\informix\extend\krakatoa\commons-logging-1.0.4.jar;C
:\informix\extend\krakatoa\commons-discovery-0.2.jar;C:\informix\extend\krakatoa\w
sdl4j-1.5.1.jar;C:\informix\extend\krakatoa\xercesImpl.jar;C:\informix\extend\krak
atoa\xmlParserAPIs.jar;C:\informix\extend\krakatoa\axis-ant.jar;C:\informix\extend
\krakatoa\log4j-1.2.8.jar;
```

In addition, we also need to modify the default security settings for the Java VM.

The default security settings for J/Foundation can be defined in the JVPHOME/informix.policy file. The necessary entries to support the AXIS framework with J/Foundation are listed in Example 9-12.

Example 9-12 The informix.policy file with AXIS support

```
grant codeBase "file:/C:/informix/extend/krakatoa/-" {
   permission java.security.AllPermission;
};

grant {
   permission java.io.SerializablePermission "enableSubstitution";
   permission java.lang.RuntimePermission "shutdownHooks";
   permission java.lang.RuntimePermission "setContextClassLoader";
   permission java.lang.RuntimePermission "reflectionFactoryAccess";
   permission java.lang.RuntimePermission "unsafeAccess";
   permission java.net.NetPermission "specifyStreamHandler";
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
   permission java.util.PropertyPermission "user.language","write";
   permission java.util.PropertyPermission "user.dir","write";
   permission java.security.SecurityPermission "getPolicy";
   permission java.util.PropertyPermission "java.naming.factory.initial","write";
   permission java.util.PropertyPermission "java.naming.provider.url","write";
};

grant {
   permission java.util.PropertyPermission "java.protocol.handler.pkgs","write";
};
```

3. Create the JVP properties file (optional). It is optional to define the JVP properties, but they are often used for debugging Java UDRs. You will find a template file in the $INFORMIXDIR\extend\krakatoa directory.
4. Set environment variables.You do not need any extra environment variables to execute UDRs written in Java code. However, since we are developing Java UDRs, you must include JVPHOME/krakatoa.jar in your CLASSPATH environment variable so that JDK can compile the Java source files that use Informix Java packages. For a complete description of the CLASSPATH settings for AXIS UDR development, refer to "Java classpath settings for AXIS UDR development" on page 324.
5. Now copy all Java class libraries from the AXIS distribution (for example, c:\axis-1_3\lib) into the $INFORMIXDIR\extend\krakatoa directory.
6. Finally, copy the **xercesImpl.jar** and the **xmlParserAPIs.jar** class library from the Xerces distribution (for example, C:\xerces-2_5_0) also into the $INFORMIXDIR\extend\krakatoa directory.

Java classpath settings for AXIS UDR development

The Java classpath for developing the AXIS based UDRs is shown in Example 9-13.

Example 9-13 Classpath settings for AXIS UDR development

```
C:\axis-1_3\lib\axis.jar;C:\axis-1_3\lib\jaxrpc.jar;C:\axis-1_3\lib\saaj.jar;c:\ax
is-1_3\lib\commons-logging-1.0.4.jar;C:\axis-1_3\lib\commons-discovery-0.2.jar;C:\
axis-1_3\lib\wsdl4j-1.5.1.jar;C:\xerces-2_5_0\xercesImpl.jar;C:\xerces-2_5_0\xmlPa
rserAPIs.jar;C:\informix\extend\krakatoa\krakatoa.jar;C:\jaf-1.0.2\activation.jar;
C:\javamail-1.3.3_01\mail.jar;.
```

9.3.3 The basic IDS Web service consumer development steps

Before we start to access some Web services from IDS 10, let us consider the required steps:

1. Obtain access to the WSDL file for the desired Web service, either by downloading it to the local server or have access to it via the http protocol.
2. Use the AXIS WSDl2Java tool to generate the Web service Java class files.
3. Compile the class files from step 2 (no coding needed!)
4. Write a small Java UDR wrapper to access the generated AXIS classes. You can take the Java UDR wrappers from the examples below as templates for your own projects.
5. Create a Java jar file which should contain the generated AXIS class files and your Java UDR wrapper class.
6. Write a simple SQL script to register your Java UDR in the IDS database of your choice.
7. Register your Java UDR in the database of your choice with the SQL script from step 6.
8. Run and test your Java UDRs to access the Web services.

9.3.4 The AXIS WSDL2Java tool

The WSDL2Java tool which part of the org.apache.axis.wsdl.WSDL2Java class is the starting point to generate Java classes from a given WSDL file.

Its normally being executed by the following command line:

```
java org.apache.axis.wsdl.WSDL2Java <WSDL-file-URL>
```

Tip: To make the execution of this tool easier for you throughout the examples in the following sections, we suggest to create a small batch/script file like the one shown in Example 9-14. Call this file (in a Windows environment) wsdl2java.bat.

Example 9-14 The wsdl2java.bat file (for Windows platforms)

```
@echo off
SET TMPCLASSPATH=%CLASSPATH%
SET CLASSPATH=.
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\axis.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\jaxrpc.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\saaj.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\commons-logging-1.0.4.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\commons-discovery-0.2.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\wsdl4j-1.5.1.jar
SET CLASSPATH=%CLASSPATH%;C:\xerces-2_5_0\xercesImpl.jar
SET CLASSPATH=%CLASSPATH%;C:\xerces-2_5_0\xmlParserAPIs.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\axis-ant.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\log4j-1.2.8.jar
SET CLASSPATH=%CLASSPATH%;C:\jaf-1.0.2\activation.jar
SET CLASSPATH=%CLASSPATH%;C:\javamail-1.3.3_01\mail.jar
echo --------------------------------------------
echo --= Classpath has been set for AXIS needs =--
echo --------------------------------------------
java org.apache.axis.wsdl.WSDL2Java -p %2 -v %1
SET CLASSPATH=%TMPCLASSPATH%
```

The wsdl2java.bat script file has **two parameters**: **the WSDL file** URL and a **package name**. The package name becomes also a local subdirectory to the directory in which you're executing the wsdl2java.bat file.

The WSDL file URL can be either a local filename or an URL on the Internet (for example, http://www.someserver.com/webserviceinfo/myservice.wsdl).

9.3.5 A simple IDS Web service consumer example

So let us start with our example project, the currency exchange Web service from:

http://www.xmethods.net

This Web service allows the currency conversion between different foreign currencies. You only have to provide the source currency country name and then the target currency country name.

Now follow the development steps we have outlined in 9.3.3, "The basic IDS Web service consumer development steps" on page 324:

1. Obtain a copy of the Web service WSDL file:

The WSDL file for this Web service can be obtained from:

`http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl`

You can either download the WSDL file to your local disk or use the above URL directly as input to the WSDL2Java tool. For your convenience we have also included the WSDL file in Example 9-15.

Example 9-15 The CurrencyExchange WSDL file

```
<?xml version="1.0"?>
<definitions name="CurrencyExchangeService"
targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
   <message name="getRateRequest">
      <part name="country1" type="xsd:string"/>
      <part name="country2" type="xsd:string"/>
   </message>
   <message name="getRateResponse">
      <part name="Result" type="xsd:float"/>
   </message>
   <portType name="CurrencyExchangePortType">
      <operation name="getRate">
         <input message="tns:getRateRequest" name="getRate"/>
         <output message="tns:getRateResponse" name="getRateResponse"/>
      </operation>
   </portType>
   <binding name="CurrencyExchangeBinding" type="tns:CurrencyExchangePortType">
      <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getRate">
         <soap:operation soapAction=""/>
         <input name="getRate">
            <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </input>
         <output name="getRateResponse">
            <soap:body use="encoded" namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         </output>
      </operation>
   </binding>
   <service name="CurrencyExchangeService">
      <documentation>Returns the exchange rate between the two
currencies</documentation>
      <port name="CurrencyExchangePort" binding="tns:CurrencyExchangeBinding">
         <soap:address location="http://services.xmethods.net:80/soap"/>
      </port>
   </service>
</definitions>
```

While looking at the WSDL file, you might have already noticed that the two input parameter (country1 and country2) are of type String and the result is of type float.

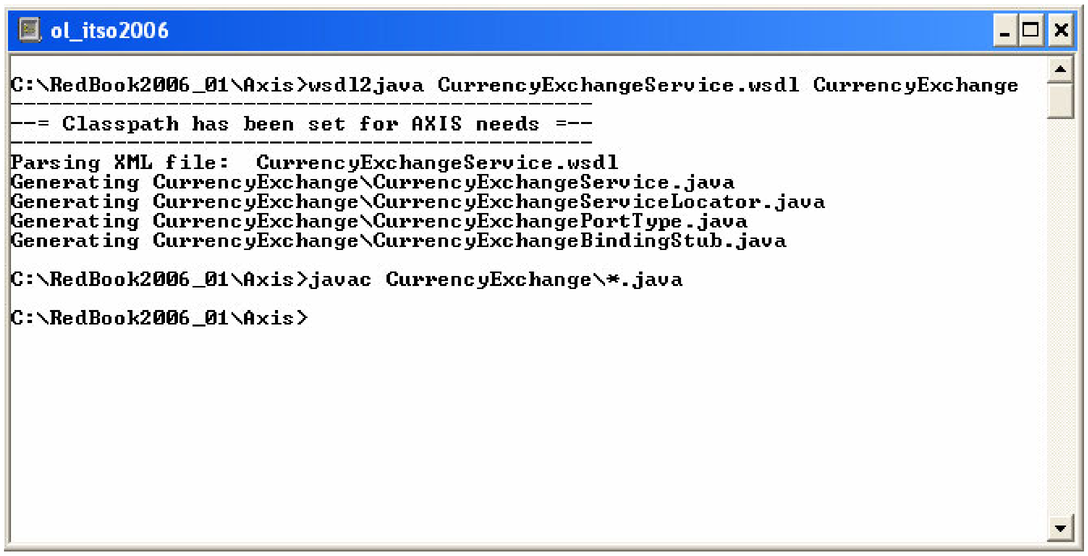
2. Now we need to generate the AXIS Java classes for our Web service.

To do this, create a directory of your choice (for example, C:\Redbook2006_01\Axis) and copy the WSDL file into this directory.

From a command line window, run the prepared wsdl2java.bat scrip file with the following parameters:

```
wsdl2java CurrencyExchangeService.wsdl CurrencyExchange
```

This will generate a subdirectory called CurrencyExchange, and this subdirectory should contain the following files: CurrencyExchangeBindingStub.java, CurrencyExchangePortType.java, CurrencyExchangeService.java, CurrencyExchangeServiceLocator.java (Figure 9-9).



```
C:\RedBook2006_01\Axis>wsdl2java CurrencyExchangeService.wsdl CurrencyExchange
-----------------------------------------------
--= Classpath has been set for AXIS needs =--
-----------------------------------------------
Parsing XML file:  CurrencyExchangeService.wsdl
Generating CurrencyExchange\CurrencyExchangeService.java
Generating CurrencyExchange\CurrencyExchangeServiceLocator.java
Generating CurrencyExchange\CurrencyExchangePortType.java
Generating CurrencyExchange\CurrencyExchangeBindingStub.java

C:\RedBook2006_01\Axis>javac CurrencyExchange\*.java

C:\RedBook2006_01\Axis>
```

Figure 9-9 Generating and compiling the CurrencyExchange AXIS classes

3. Now you need to compile the generated Java classes from step 2 by simply executing:

   ```
   javac CurrencyExchange\*.java
   ```

   Before you execute the Java compiler, make sure that you have set the CLASSPATH environment variable correctly (see Example 9-13 on page 324) and also that you have the Java compiler in your PATH environment variable (for example, C:\j2sdk1.4.2_06\bin).

4. In order to utilize the generated AXIS class files for the CurrencyExchange Web service we need to write a simple Java wrapper UDR to call the required methods.

   So first look at the final code in Example 9-16.

Example 9-16 CurrencyExchangeUDRs.java

```
import CurrencyExchange.*;

public class CurrencyExchangeUDRs
{
   public static double currencyExchange( String country1, String country2)
                                                                  throws Exception
   {
      double  RetVal;

      CurrencyExchange.CurrencyExchangeService service =
                        new CurrencyExchange.CurrencyExchangeServiceLocator();
```

```
      CurrencyExchange.CurrencyExchangePortType port =
                                            service.getCurrencyExchangePort();

      RetVal = port.getRate(country1, country2);

      return RetVal;
   }
};
```

The currencyExchange method implements the Web service API by accepting the two country descriptions as Java strings and returns a Java double type.

First, we need to create a service instance of type CurrencyExchangeService which can be achieved by creating a new CurrencyExchangeServiceLocator object.

Then we need to obtain the port object of type CurrencyExchangePortType from the service object.

And finally we need to call the getRate(String, String) method to generate the SOAP message which is then being sent to the Web service provider.

The getRate() method is defined in the CurrencyExchangeBindingStub.java file.

Save the Java code from Example 9-16 on page 327 into your example directory (for example, C:\RedBook2006_01\Axis) as CurrencyExchangeUDRs.java.

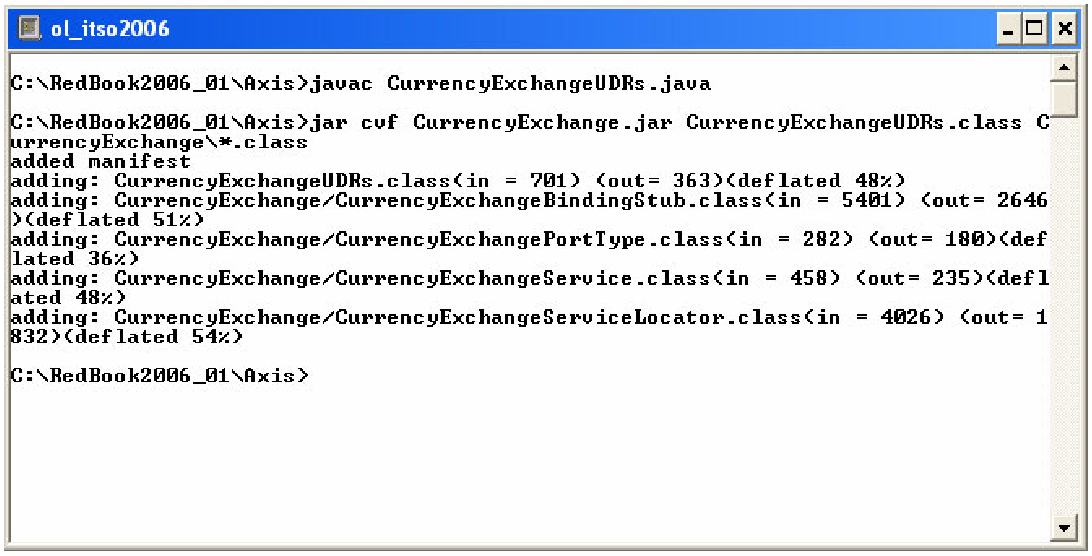
Now compile the CurrencyExchangeUDRs.java file:

```
javac CurrencyExchangeUDRs.java
```

5. In preparation for the registration in your IDS 9 database we need to pack all of our classes (generated AXIS classes plus the UDR wrapper) into a Java jar file. To do this, execute this command:

```
jar cvf CurrencyExchange.jar CurrencyExchangeUDRs.class
CurrencyExchange\*.class
```

(Also see Figure 9-10.)



```
C:\RedBook2006_01\Axis>javac CurrencyExchangeUDRs.java

C:\RedBook2006_01\Axis>jar cvf CurrencyExchange.jar CurrencyExchangeUDRs.class C
urrencyExchange\*.class
added manifest
adding: CurrencyExchangeUDRs.class(in = 701) (out= 363)(deflated 48%)
adding: CurrencyExchange/CurrencyExchangeBindingStub.class(in = 5401) (out= 2646
)(deflated 51%)
adding: CurrencyExchange/CurrencyExchangePortType.class(in = 282) (out= 180)(def
lated 36%)
adding: CurrencyExchange/CurrencyExchangeService.class(in = 458) (out= 235)(defl
ated 48%)
adding: CurrencyExchange/CurrencyExchangeServiceLocator.class(in = 4026) (out= 1
832)(deflated 54%)

C:\RedBook2006_01\Axis>
```

Figure 9-10 Compile the UDR wrapper and create the jar file

6. Now we need to create a simple SQL script to first store our CurrencyExchange.jar file which contains the UDR wrapper plus the generated AXIS classes into the database and

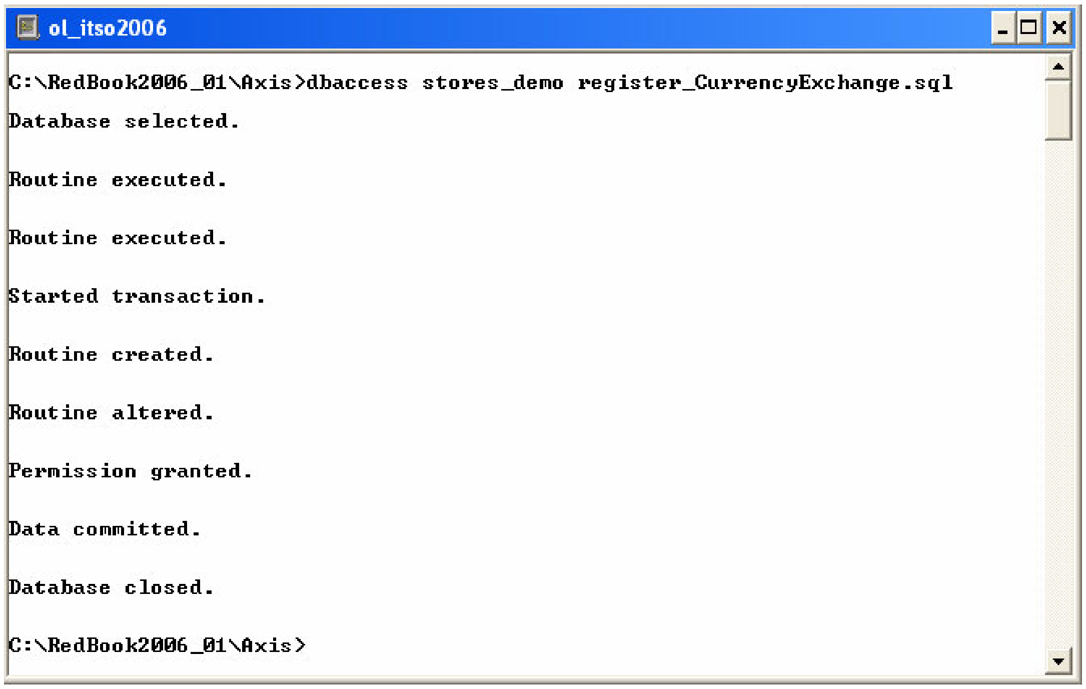
then connect the Java classes with the SQL layer by defining a Java UDR with the CREATE FUNCTION SQL statement.

You can use the SQL script from Example 9-17 on page 329 as a template for similar Java UDRs in the future. So on the SQL level we're naming our user defined routine simply CurrencyExchange. This routine takes two LVARCHARs as parameters and returns a SQL FLOAT data type which matches the Java double type.

Example 9-17 The register_CurrencyExchange.sql script

```
execute procedure
install_jar('file:C:/RedBook2006_01/Axis/CurrencyExchange.jar','CurrencyExchange')
;

execute procedure ifx_allow_newline('t');

begin work;

create function CurrencyExchange (lvarchar, lvarchar)
returns float as exchange_rate
external name
'CurrencyExchange:CurrencyExchangeUDRs.currencyExchange(java.lang.String,
java.lang.String)'
   language java;

alter function CurrencyExchange (lvarchar, lvarchar)
   with (add parallelizable);

grant execute on function CurrencyExchange (lvarchar, lvarchar) to public;

commit work;
```

The install_jar procedure stores the CurrencyExchange.jar into a smart blob in the default smart blob space in the IDS 10 instance and gives it the symbolic name CurrencyExchange. which can be used in the create function statement to reference the jar file. See Figure 9-11.



```
C:\RedBook2006_01\Axis>dbaccess stores_demo register_CurrencyExchange.sql
Database selected.

Routine executed.

Routine executed.

Started transaction.

Routine created.

Routine altered.

Permission granted.

Data committed.

Database closed.

C:\RedBook2006_01\Axis>
```

Figure 9-11 Register the Java “Wrapper” UDR with the stores_demo database

The create function finally registers the Java UDR with the database and makes it available to any SQL compliant application.

7. In order to register your CurrencyExchange UDR you should have a database with logging enabled. Assuming you might want to register your UDR with the IDS stores_demo database you only have to run the SQL script by executing:

   ```
   dbaccess stores_demo register_CurrencyExchange.sql
   ```

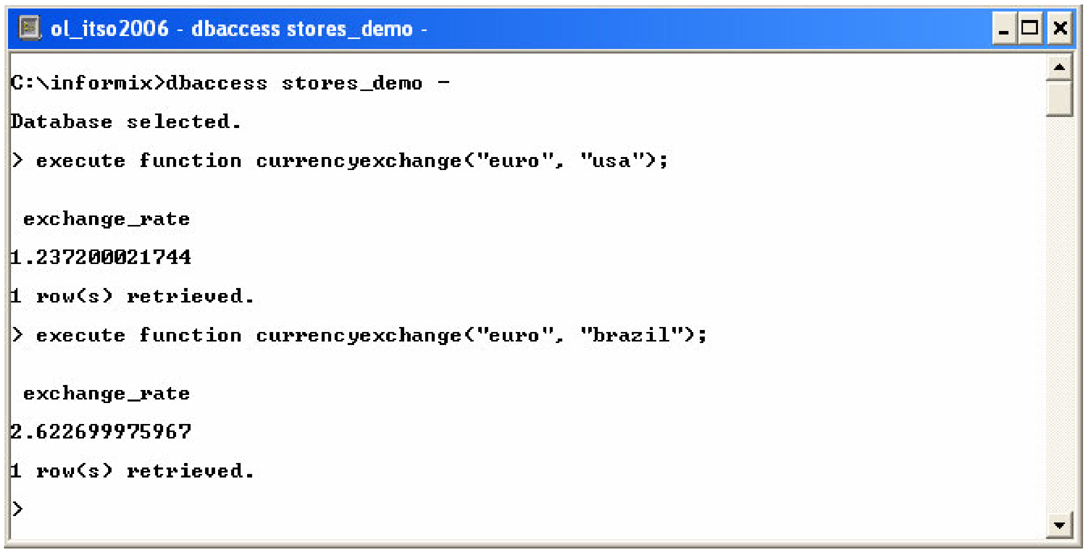
   See also Figure 9-11 on page 330.

8. Now we’re ready to test the Java UDR to call the Web service. Before you can test the UDR make sure that you’re connected to the Internet. Then, for example, start dbaccess to connect to the stores database and execute the CurrencyExchange function. Since we’re using SQL and SQL does not differentiate between lowercase and uppercase letters we just simply type:

   ```
   execute function currencyexchange(“<country1>”, “<country2”>).
   ```

   For valid values for the country parameters, consult the CurrencyExchange Web service description on the Web site:

   http://www.xmethods.net



```
C:\informix>dbaccess stores_demo -

Database selected.

> execute function currencyexchange("euro", "usa");

 exchange_rate

1.237200021744

1 row(s) retrieved.

> execute function currencyexchange("euro", "brazil");

 exchange_rate

2.622699975967

1 row(s) retrieved.

>
```

Figure 9-12 Test of the CurrencyExchange() UDR from within dbaccess

Tip: If you are behind a firewall, then you might have to set additional properties for the J/Foundation Java VM via the **JVPARGS** variable.

So if you typically use a SOCKS compliant proxy, replace the **JVPARGS** value in the ONCONFIG file with the following line:

```
-Djava.security.policy=C:\informix\extend\krakatoa\informix.policy;-DsocksProxy
Host=<SocksProxyhostname>;-DsocksProxyPort=<socksProxyPortvalue>
```

If you're using a standard HTTP proxy, you might have to use the following value for **JVPARGS** instead:

```
-Djava.security.policy=C:\informix\extend\krakatoa\informix.policy;-Dhttp.proxy
Host=<httpProxyhostname>;-Dhttp.proxyPort=<httpProxyPortvalue>
```

For more details about proxy support in the Java VM, consult this Web site:

http://java.sun.com/j2se/1.4.2/docs/guide/net/properties.html

9.4 XML related DataBlades

In this section we describe XML related DataBlades: UDRs, XSLT and Web.

9.4.1 XML generating UDRs

The extensibility features in IDS 10, through extended data types and user defined routines (UDR) in C Java and Stored Procedure Language (SPL), allow for easy enhancement of the basic functionality of the underlying database server. Having such flexibility gives you the capability to implement new features in the server without having to wait for the database vendor to provide it.

Following this approach one could very easily implement user defined routines which can return dynamically generated XML, for example, a given table and attributes of this table.

A very good example for such a user defined routine (or stored procedure) can be found on the IBM Informix Developer Zone[3]. In this article, for example, the author, Jacques Roy, describes a function genxml, which can be used to generate an XML fragment or, in combination with some helper routines, a complete XML document.

Example 9-18 shows how the genxml function can be called and how the output might look.

Example 9-18 The genxml function (stored procedure)

```
SQL Statement:

SELECT genxml("customer", customer)
FROM customer;


Output of the Statement above:

<customer>
   <customer_num>101<//customer_num>
   <fname>Ludwig       </fname>
   <lname>Pauli        </lname>
   <company>All Sports Supplies </company>
   <address1>213 Erstwild Court  </address1>
   <city>Sunnyvale     </city>
   <state>CA</state>
   <zipcode>94086</zipcode>
   <phone>408-789-8075      </phone>
</customer>
```

9.4.2 XSLT DataBlade

In addition to the XML generating function above, there is also an *XSLT (XSL Transformations)* DataBlade available on the IBM Alphaworks Web site[4]. This DataBlade allows the transformation of XML documents by the means of the XSL/XSLT mechanism into a new target format. The XSLT DataBlade stores the XSL documents in LVARCHAR or CLOB data types which allow very complex XSL transformations. It even supports the HTML data type of the Web DataBlade, discussed in the next section.

9.4.3 Web DataBlade

The Web DataBlade is an optional extension of IDS 10 and supports the dynamic generation of any kind of markup language in combination with dynamic data from the IDS 9 database. The initial purpose for the Web DataBlade had been dynamic HTML publishing based on IDS, but it has been extended, for example, to produce XML or SGML documents.

The Web DataBlade utilizes user defined tags within the used markup language, in this case XML. The XML template pages are stored in the database which allows the utilization of IDS 10 server features like replication (ER, HDR), transaction security, and centralized backup.

In combination with the XSLT DataBlade it can be a very powerful way of producing dynamic XML documents, especially for very flexible publishing applications. There is a very detailed article available on the Informix DeveloperZone which covers the combination of the Web DataBlade and the XSLT DataBlade[5].

[3] http://www7b.software.ibm.com/dmdd/zones/informix/library/techarticle/0302roy/0302roy2.html.
[4] http://www.alphaworks.ibm.com/tech/xsltblade

Although the Web DataBlade is perfect for producing dynamic XML pages, it does not solve the issue how to consume XML documents, for example, and map them to database tables.

9.5 Using WebSphere MQ with Informix applications

WebSphere Message Queue (WMQ) software suite provides reliable messaging for distributed, heterogeneous applications to exchange information, delegate jobs, coordinate events, and create enterprise service BUS. When Informix applications use WMQ, you write custom code, manage multiple connections, and route data through your application. Informix Dynamic server, IDS, version10.00.UC3, introduces built-in support for Informix applications to interact with WMQ via SQL callable functions with 2-phase commit support. This eliminates development overhead and encapsulates integration complexity.

9.5.1 Brief description of WebSphere MQ

In its simplest form, WMQ is a method to exchange messages between two end points. It acts as an intermediary between to systems and provides value added functionalities like reliability, transactional semantics, and so on.

Whether you buy a book on amazon.com or enroll in e-business with ibm.com®, the order event triggers a work of the information through multiple modules: user account management, billing, packaging and shipping, procurement, customer service, partner services. The execution in triggered modules will generate subsequent work flow. To meet reliability and scaling requirements, it is typical to have application modules on multiple machines.

If you are using the same software on all systems, for instance an SAP stack, the software itself usually comes with workflow management features. If the modules are running in homogeneous environment – like LINUX machines running WebSphere and Informix – it is easier to change information via distributed queries or enterprise replication. On the other hand, if the application is running on heterogeneous systems – combination of WebSphere, DB2, Oracle and Informix – programming and setup of distributed queries or replication becomes complex and in many cases will not meet the application requirements. See Figure 9-13 on page 334.

WMQ is designed to address integration issues like this. It prefers no platform and enforces no paradigms: WMQ supports more than 80 platforms, and APIs in C, C++, Java, JMS and visual Basic. WMQ is also the mainstay for designing enterprise service bus (ESB) for SOA.

[5] http://www7b.software.ibm.com/dmdd/zones/informix/library/techarticle/0303cline/0303cline.html

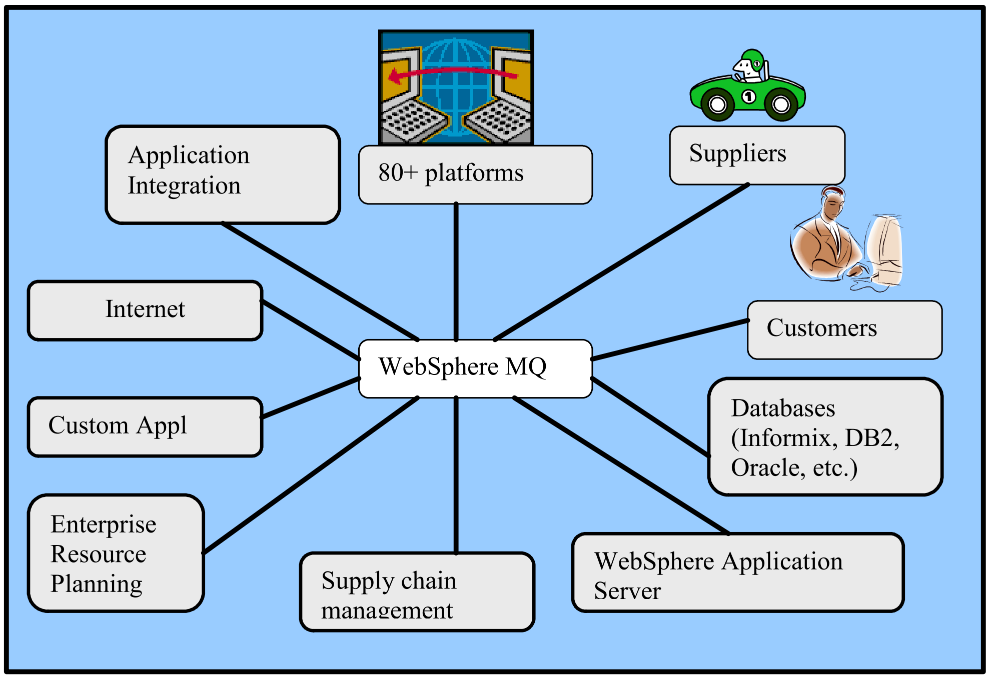


Figure 9-13 WebSphere MQ for enterprise business application integration

WMQ provides a reliable store-and-forward mechanism so each module can send and receive messages to and from it. WMQ achieves this by persistent queues and APIs for programming. In addition, WMQ Message Broker – another software in WebSphere WMQ product suite – provides message routing and translation services. Simplicity of infrastructure means the applications have to establish message formats, queue attributes, and so on. WMQ also supports publish and subscribe semantics for queues making it easy to send a single message to multiple receivers and subscribing message from queue by need – similar to mailing list.

The applications predetermine queue names, messages, message formats, just like the two network applications agree on socket numbers. The application to application message exchange is asynchronous – one application will not wait for other application to receive the message. WMQ assures the message will be stored reliably and have the message available for target application. But, it is the responsibility of the target application to receive the message from WMQ.

9.5.2 How do Informix and other database applications use WMQ?

The applications have many input sources: user entry, B2B transactions, workflow messages, data in the database, and so on. The Order Entry Application above needs to store data in Informix database and send and receive messages to WMQ. The application establishes connections with Informix and WMQ. In addition, the application will use a transaction manager to ensure reliability of data exchange. For instance, the order saved in the database has to be sent to a queue and marked as processed in the database. The order can be marked as processed only after WMQ receives the message successfully; therefore, the interaction has to have transactional protection. See Figure 9-14 on page 335.

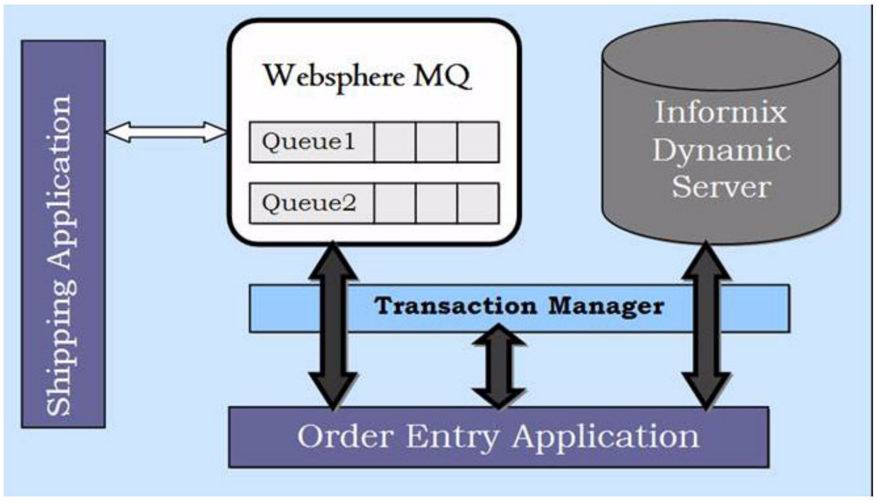


Figure 9-14 Applications using WMQ

The Order Entry application writes custom code to exchange messages from and to WMQ. Need for custom code development every time an application wants to interact with WMQ is costly – you need to train your programmers for it or hire consultants to develop, debug and maintain this code, modify it for new queues and applications. The data exchanged between the database and WMQ flows through application – not efficient for high volume data and necessitates transaction manager.

9.5.3 IDS support for WebSphere MQ

IDS provides SQL callable functions to read, receive, send, subscribe, unsubscribe and publish. See Figure 9-15 on page 335. These SQL callable functions expose WMQ features to IDS application and integrate the WMQ operations into IDS transaction; that is, fate of the WMQ operation is tied to fate of the transaction. If the transaction is rolled back, the operations made on WMQ – messages sent or received – will be rolled back. This is done by coordinating transactions at IDS and WMQ – not by compensating transaction. So, this is reliable with high performance.

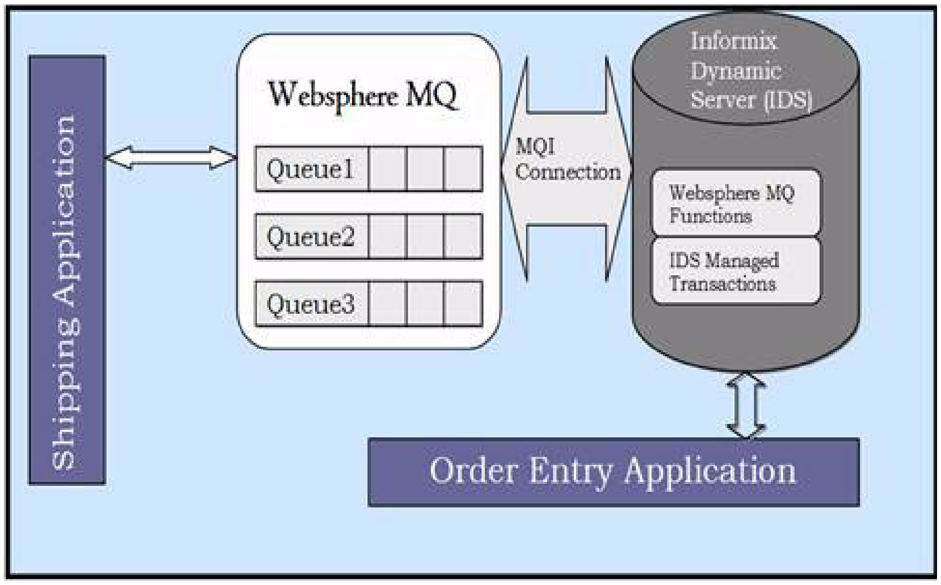


Figure 9-15 WMQ - Sending and receiving messages

Using IDS WMQ functionality, sending and receiving a message to and from an WMQ Queue is easy. See Example 9-19.

Example 9-19 WMQ - Sending and receiving

```
select MQSend("CreditService", customerid || ":" || address || ":" || product ":" 
|| orderid)
from order_tab
where customerid = 1234;

insert into shipping_tab(shipping_msg) values( MQReceive() );

create function get_my_order() returns int;

define cust_msg lvarchar(2048);
define customerid char(12);
define address char(64);
define product char(12);
define corderid char(12);
define snd_status int;

-- Get the order from Order entry application.
execute function MQReceive("OrderQueue") into cust_msg;
let customerid = substr(cust_msg, 1, 12);
let address = substr(cust_msg, 14, 77);
let product = substr(cust_msg, 79, 90);
let corderid = substr(cust_msg, 92, 103);

insert into shipping_table(custid, addr, productid, orderid)
   Values(customerid, address, product, corderid);

-- send the status to CRM application
execute function MQSend("CRMQueue", corderid || ":IN-SHIPPING") into snd_status;
return 1;
end function;
```

When you rollback the transaction as shown in Example 9-20, the message received will be restored in the queue book order and the row will also be removed from shipping_tab.

Example 9-20 Rollback the transaction

```
begin work;
INSERT into shipping_tab(shipping_msg)
                 values (MQReceive(“bookorderservice”));

rollback work;  -- Undo previous statement including WMQ operation
```

9.5.4 Programming for WMQ

IDS provides functions exposing each interface WMQ provides – read, receive, send, publish, subscribe, unsubscribe – and functions to send and receive large messages. The WMQ functions can be invoked any where a function can be used: values clause, projection list, query filters, stored procedures, and triggers. In addition, with IDS, you can map WMQ Queue into an IDS table. Insert on this table will translate to send operation to WMQ and select will translate to either read or receive.

While WMQ provides simple abstractions of queue and its operations, each operations comes with plenty of options like msg expiry time, retry count, and so on. So, IDS has abstracted these options into service, policy, and optionally correlation ID:

- Service

  Maps a queue, queue manager, code set of the messages into the service. The table “informix”.mqiservice stores the mappings. IDS.DEFAULT.SERVICE is mapped to system default queue manager, queue named IDS.DEFAULT.QUEUE and default code set.

- Policy

  The policy defines the attributes like priority, expiry date, and so on., for each operation. The table “informix”.mqipolicy stores 37 attributes for each policy. IDS.DEFAULT.POLICY is the default policy. Depending on your application environment, create one or more policies.

- Correlation ID

  When multiple applications share the same queue, you can control the interaction using Correlation ID of up to 48 bytes. Once the applications agree on the Correlation ID for their messages, they can simply get messages matching their correlation ID. They work similar to filters or predicates in SQL queries. Correlation ID is not mandatory and has no default value.

Basic programming functions

The MQ programming functions in IDS are listed in Table 9-1.

Table 9-1 MQ functions in IDS

Function name	Description
MQSend()	Send a string message to a queue
MQSendClob()	Send CLOB data to a queue
MQRead()	Read a string message in the queue into IDS without removing it from the queue
MQReadClob()	Read a CLOB in the queue into IDS without removing it from the queue
MQReceive()	Receive a string message in the queue into IDS and remove it from the queue
MQReceiveClob()	Receive a CLOB in the queue into IDS and remove it from the queue
MQSubscribe()	Subscribe to a Topic
MQUnSubscribe()	UnSubscribe from a previously subscribed topic
MQPublish()	Publish a message into a topic
MQPublishClob()	Publish a CLOB into a topic
CreateMQVTIRead()	Create a read VTI table and map it to a queue
CreateMQVTIReceive()	Create a receive VTI table and map it to a queue
MQTrace()	Trace the execution of MQ Functions
MQVersion()	Get the version of MQ Functions

Functions for sending messages from IDS to WMQ

- *MQSend(Service, Service_Policy, Message, CorrelationID)*
- *MQSendClob(Service, Service_Policy, ClobMessage, CorrelationID)*

You can send a message up to 32739 bytes to an WMQ queue. MQSendClob() behaves same as MQSend() except it takes CLOB as its message parameter instead of character type. Message and ClobMessage are the mandatory parameters. IDS will send the message to the queue managed by the queue manager using the policy in the Service record entry saved in “informix”.mqiservice table.

Parameter interpretation

Here is how the calls are interpreted for MQSend(). The other functions follow the same pattern.

When the four parameters are given, translation is straightforward and is executed as given:

```
MQSend(serviceparam, policyparam, messageparam, correlationparam)
```

Here is the translation when one or more parameters are missing.

- MQsend(messageparam) is translated to:

  ```
  MQSend(“IDS.DEFAULT.SERVICE”, “IDS.DEFAULT_POLICY”, messageparam, “”);
  ```

- MQsend(serviceparam, messageparam) is translated to:

  ```
  MQSend(serviceparam, “IDS.DEFAULT_POLICY", messageparam, “”);
  ```

- MQsend(serviceparam, policyparam, messageparam) is translated to:

  ```
  MQSend(serviceparam, policyparam, messageparam, "");
  ```

Examples

An example is:

```
SELECT MQSend(“myservice”, “mypolicy”, orderid || “:” || address)
FROM   tab
WHERE orderid = 12345;
```

All WMQ functions should be run in a transaction. In IDS SELECT, UPDATE, DELETE and INSERT automatically start a new transaction. Or, you can start a new transaction with BEGIN WORK statement.

Simply executing the function will give error:

```
EXECUTE FUNCTION MQSend(“MyService”,
        “<order><id>5</id><custid>6789</custid></order>”);
```

IDS does not implicitly start a new transaction for the EXECUTE statement. So, you’ve to start a transaction explicitly.

```
BEGIN WORK;
EXECUTE FUNCTION MQSend("MyService",
  "<order><id>5</id><custid>6789</custid></order>");
COMMIT WORK;
```

If the transaction gets rolled back, all operations on WMQ will be rolled back just like IDS rolls back its changes.

```
BEGIN WORK;
INSERT INTO resultstab(sendval)
      VALUES(MQSend("MyService",
```

```
           "<order><id>5</id><custid>6789</custid></order>")

ROLLBACK WORK;
```

Read and Receive functions

- *MQRead(Service, Policy, CorrelationID)* returns lvarchar(32739)
- *MQReadClob(Service, Policy, CorrelationID)* returns CLOB
- *MQReceive(Service, Policy, CorrelationID)* returns lvarchar(32739)
- *MQReceiveClob(Service, Policy, CorrelationID)* returns CLOB

The Read operation gets the message from the queue without deleting the message from the queue. Receive operation removes the message from the queue and get the message. These functions can be called with zero or more parameters. The parameters are interpreted similar to MQSend() above. The transactional behavior of receive functions is same as MQSend.

MQRead() and MQReceive() can return up to 32739 bytes. The maximum size of the message itself is a WMQ configuration parameter. The larger messages should be read or received as CLOB. For MQ, message is a message. Depending on the length, IDS differentiates between messages to map the messages to data types.

If correlation ID is given, WMQ get the next message in the queue matching correlation ID, otherwise a NULL message is returned. Policy determines the wait time when no applicable message is present on queue. So, using predefined correlation ID, multiple applications can share the same queue and for different purposes.

Examples are:

```
SELECT mqread('SHIPPING.SERVICE','My.DEFAULT.POLICY')
       FROM systables where tabid = 1;

SELECT mqreceive('SHIPPING.SERVICE','My.DEFAULT.POLICY')
       FROM systables where tabid = 1;
```

Publish and Subscribe functions

Publishing and subscribing to a queue is an effective configuration for exchanging information between multiple applications on multiple subjects. When an order entry has to go to credit card application, shipping, CRM, and partner application, order entry application publishes the order once to a queue and target applications can subscribe to the queue and obtain the message using either read or receive function. Within this scheme, WMQ also supports categorizing messages into topics for finer control. For example, the order entry message can categorize the order into books, electronics, clothing topics.

You have to configure the queue for publishing and define the topics. WMQ allows defining topics statically or dynamically. The message broker provides the publish and subscribe features and it has to be running in addition to queue manager. Message Broker component provides message routing, message translation easing business integration challenges.

Subscribers subscribe to a topic and *specify the queue on which you want to receive the messages*; when a publisher inserts a message on that topic into the queue, the WMQ broker routes the messages to all of the queues of each specified subscriber. The subscribers retrieve the message from the queue via read or receive functions.

Note: Before using the publish and subscribe services, you have to setup the *"informix".mqipubsub* table. See the Informix dynamic server documentation for its schema and examples.

The publish and subscribe functions are:

- *MQPublish(publisher_name, policyparam, message, topic, correlationid)*;
- *MQSubscribe(subscriber_name, policy_name, topic)*
- *MQUnsubscribe(subscriber_name, policy_name, topic)*

The publisher name and subscriber names have to be defined in "informix".mqipubsub table. We have discussed other parameters before.

Examples:

```
SELECT mqSubscribe(‘WeatherChannel’,"Weather")
        FROM systables WHERE tabid = 1;

SELECT mqPublish(‘WeatherChannel’,
"<weather><zip>94501</zip><date>7/27/2006</date><high>89</high><low>59</low></w
eather>","Weather")
FROM systables WHERE tabid = 1;

SELECT mqreceive('WeatherChannel',"Weather")
FROM systables WHERE tabid = 1;
```

MQ Utility functions

They are MQVersion() and MQTrace().

*MQVersion() r*eturns the current version of the WMQ blade in IDS.

MQTrace(trace_level, trace_file) enables you to trace execution path of WMQ functions and interaction between IDS and MQ. The tracing level can be from 10 to 50 – multiple of 10.

An example of Trace output is shown in Example 9-21.

Example 9-21 Trace output

```
14:19:38 Trace ON level : 50
14:19:47  >>ENTER : mqSend<<
14:19:47    status:corrid is null
14:19:47  >>ENTER : MqOpen<<
14:19:47   status:MqOpen @ build_get_mq_cache()
14:19:47  >>ENTER : build_get_mq_cache<<
14:19:47   status:build_get_mq_cache @ mi_get_database_info()
14:19:47   status:build_get_mq_cache @ build_mq_service_cache()
14:19:47  >>ENTER : build_mq_service_cache<<
14:19:47  <<EXIT : build_mq_service_cache>>
```

MQ Table mapping functions

Invoking the WMQ functions in IDS is easy, not the easiest way to use MQ. IDS can map a WMQ queue to IDS table. SELECT on the table will fetch the messages in the queue and INSERT on the table will send the message. We discuss its usage in this section. Other operations like UPDATE and DELETE are disallowed on the table.

The functions are:

- *MQCreateVtiRead(readtable, servicename, policy, maxMessage)*
- *MQCreateVtiReceive(receivetable, servicename, policy, maxMessage)*

SELECT on read table imitates MQRead() – fetches the message without deleting it from the queue where as SELECT on receive table deletes the message on the queue as well. The maxMessage parameter determines size of the column, but it also determines the size of the column and the type of message. Positive length creates a column Maximum length of the message is you can define is 32607. Use -1 as maxMessage to retrieve the message as a CLOB and -2 to retrieve the message as BLOB.

An example is shown in Example 9-22.

Example 9-22 Creating a READ table

```
-- Create a READ table with max message length 4096.
execute function MQCreateVTIREAD("myreadtable",
                          "myservice", "mypolicy", 4096);

-- Below is the table created by MQCreateVTIREAD() function.

create table myreadtab
   ( msg      lvarchar(4096),
     correlid varchar(24),
     topic    varchar(40),
     qname    varchar(48),
     msgid    varchar(12),
     msgformat varchar(8))
using "informix".mq (SERVICE = "myservice", POLICY = "mypolicy", ACCESS = "READ");

-- Get the top 10 messages from the queue.
SELECT first 10 * from myreadtable;
-- INSERT a message into the table
INSERT into myreadtable values("IBM:81.98;Volume:1020");
-- SELECT the first message matching correlation id
SELECT FIRST 1 * from myreadtable where correlid = 'abc123';
```

IDS is aware of the correlation id predicate and sends the correlation id request to MQ. WMQ matches the correlation ID and sends the matched message.

You create a table to transport BLOB data with:

```
execute function MQCreateVTIRECEIVE("mydoctable", "myservice", "mypolicy", -2);
```

The table created by MQCreateVTIRECEIVE() function is shown in Example 9-23.

Example 9-23 MQCreateVTIRECEIVE() created table

```
create table mydoctable
      ( msg      BLOB,
       correlid varchar(24),
       topic    varchar(40),
       qname    varchar(48),
       msgid    varchar(12),
       msgformat varchar(8))
   using "informix".mq (SERVICE = "myservice", POLICY = "mypolicy",
   ACCESS = "RECEIVE");
```

```
INSERT into mydoctable(msg) select blobcol from ordertab;

-- insert using blob, get through blob
INSERT into mydoctable(msg) values(filetoblob("/etc/passwd", "client"));

select lotofile(msg, '/tmp/blob.dat','client') from mydoctable;
```

How not to use this feature

Both INSERT and SELECT operations on RECEIVE tables and INSERT on READ tables change the data on MQ. You should be aware of these nuances.

See the statements listed in Example 9-24.

Example 9-24 SELECT on RECEIVE tables

```
-- Find all the golf orders in the message queue.
SELECT * from myrecvorder where msg matches '%golf%';
-- Find all the orders from zip 94501.
SELECT * from myrecvorder where msg[12,15] = '94501';
-- Find all the messages greater than a correlation ID
SELECT * from myrecvorder where correlid > "abc123";
-- Find the number of messages for my correlation id
SELECT count(*) from myrecvorder;
```

To complete the two first SELECT statements, IDS retrieves *all* the messages in the service myrecvorder, then applies the filters and returns the qualified rows. The unqualified messages are lost. Use READ tables if you want to apply these predicates, but be aware of the message fetch overhead.

9.5.5 Transactions

When you invoke any WMQ function exchanging message with MQ, you have to be in a transaction – implicit or explicit. To provide reliable interaction between IDS and MQ, transactions are necessary. When the commit is successful, application needs all changes to data at IDS and WMQ are persisted and when the application rolls back, any operation at WMQ are rolled back just like operations on IDS are rolled back. IDS implicitly starts a transaction when you issue DML (UPDATE, DELETE, INSERT or SELECT) and DDL statements (CREATE statements). Or you can explicitly start a new transaction with BEGIN WORK statements and APIs like JDBC start a new transaction when you set autocommit off. Note EXECUTE FUNCTION/PROCEDURE statement does not start one, so you need to start a transaction before invoking WMQ function in an EXECUTE statement. See Figure 9-16.

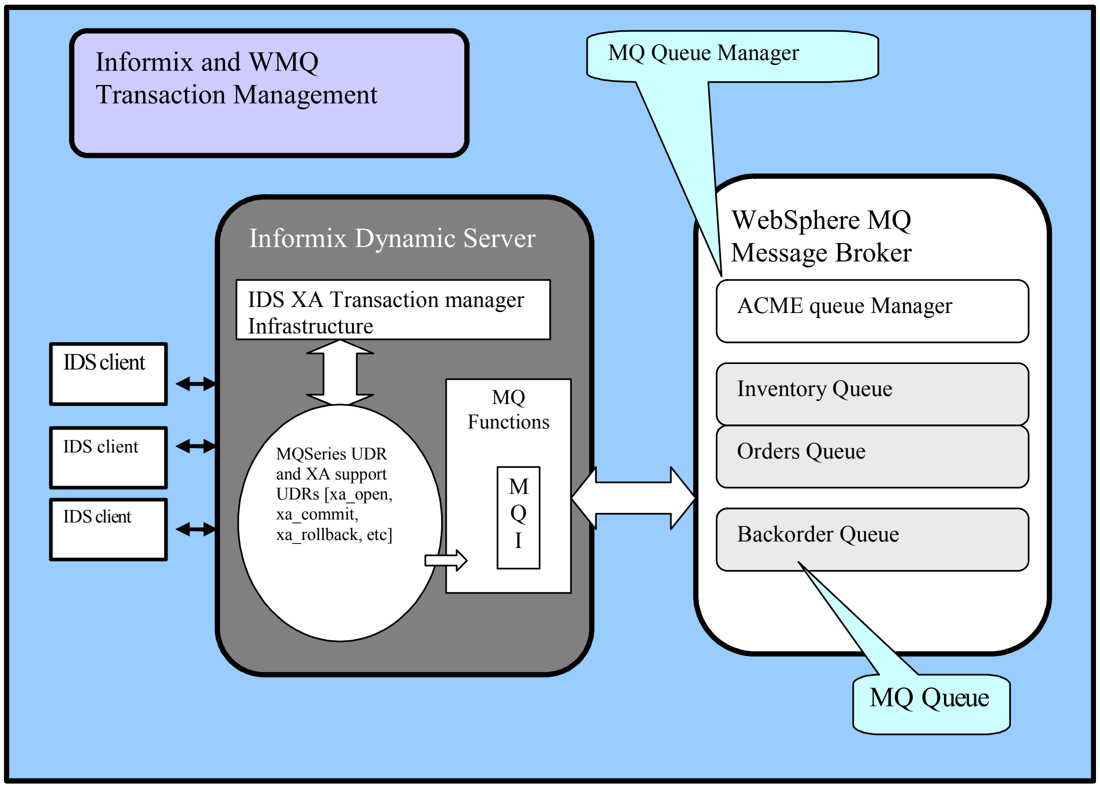

Informix and WMQ Transaction Management
MQ Queue Manager
Informix Dynamic Server
IDS XA Transaction manager Infrastructure
IDS client
IDS client
IDS client
MQSeries UDR and XA support UDRs [xa_open, xa_commit, xa_rollback, etc]
MQ Functions
MQI
WebSphere MQ Message Broker
ACME queue Manager
Inventory Queue
Orders Queue
Backorder Queue
MQ Queue

Figure 9-16 WMQ transaction management

The transaction management is transparent to application. Application simply uses WMQ functionality under a transaction and IDS handles the commit or rollback co-ordination between IDS and WMQ using the open 2-phase commit protocol. This is integrated into IDS transaction manager; IDS handles WMQ along with its distributed transactions involving other IDS instances. During IDS-MQ interaction, IDS opens a connection to WMQ and when application the application invokes an the fist WMQ function within a transaction, IDS begins a corresponding transaction at MQ. During commit or rollback, IDS transaction manager is aware of WMQ participation in the transaction and co-ordinates the transaction with it.

Environment

MQ functionality is provided with Informix Dynamic Server and the datablade is installed into $INFORMIXDIR/extend when you install IDS. You've to register the datablade in the database you to want to invoke MQ Functions. We currently support WMQ interaction Informix logged database. IDS communicates with WebSphere MQ using the server API and therefore WebSphere MQ needs to be installed on the same machine as the server. This WebSphere MQ can channel the messages to one or more remote WebSphere MQ servers. Each Dynamic Server instance can connect to only one WMQ queue manager.

Please send in the request the authors or your favorite IBM Informix support team if you want to use it in non-logged or ANSI mode database.

Platform support

Table 9-2 summarizes IDS and WMQ versions by supported platforms.

Table 9-2 IDS and WMQ platforms support

Informix Dynamic server Version	Support platforms	WebSphere MQ Version
10.00.xC3 and Higher	Solaris™ – 32 bit HP/UX (PA-RISC) – 32 bit AIX – 32bit Windows – 32bit.	Needs V5.3 and higher
10.00.xC4 and Higher	AIX – 64bit HP/UX (PA-RISC) – 64 bit	Needs v6.0 and higher
10.00.xC5 and Higher	Linux (Intel) – 32 bit Linux (pSeries®) – 64bit Solaris – 64 bit	Needs v6.0 and higher

If you are on a different platform, refer to the machine notes to see if the support has been added.

9.5.6 Summary

IDS WMQ functionality eliminates need for custom code development for IDS applications interacting with MQ. Once you setup the queues, services, and policies, developers can use WMQ functions like other built-in functions in the development environment of their choice. Even better, set up the READ and RECEIVE tables, get developers to SELECT from and INSERT into it.

9.6 Integrating I4GL applications with SOA through EGL

Informix 4GL (I4GL) has been (and still is) a very successful 4th generation programming language which has been initially developed by Informix in the mid 80’s.

At one point in time I4GL was the 3rd most important development language for database oriented applications worldwide. I4GL’s success was heavily related to the ease of programming around character based, Informix database server centric applications. Due to I4GL’s popularity, one still find literally thousands of I4GL applications in production with millions line of 4GL code.

Over the recent years I4GL became slowly out of date since more and more developers and users requested more modernized applications, specifically supporting graphical user interfaces (GUI) and Web deployment. In addition many I4GL developers and customers have been asking for an easy SOA integration of existing Informix 4GL applications.

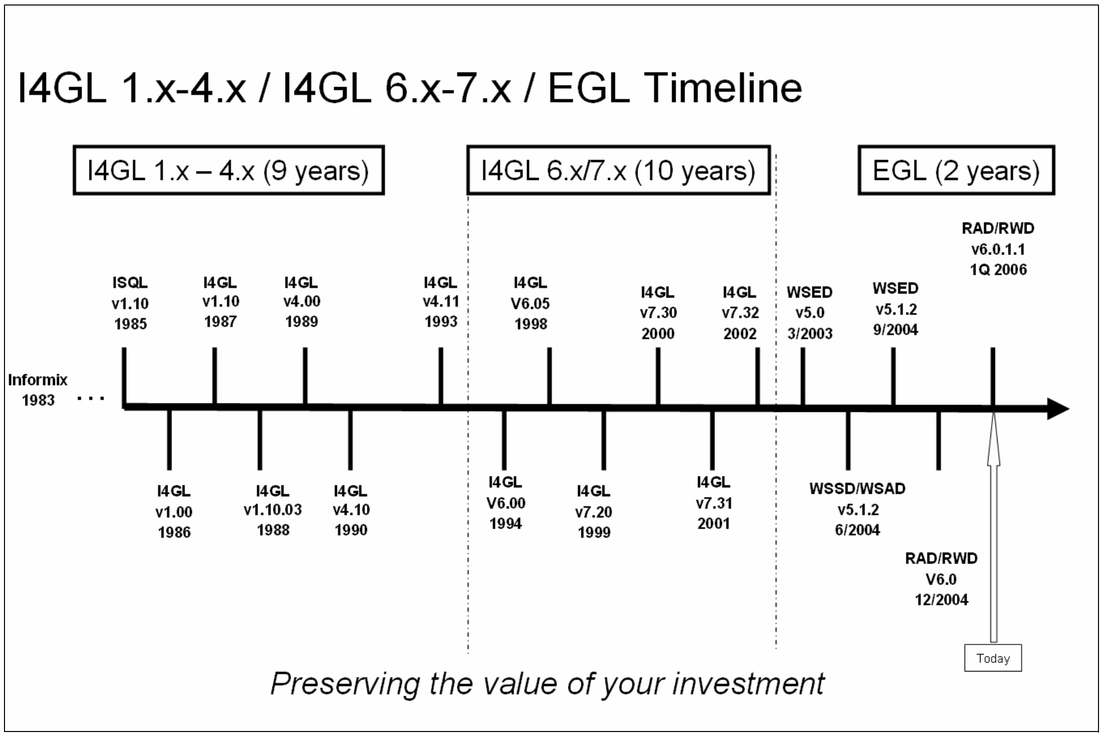


Figure 9-17 The 4GL / EGL development timeline

The Informix 4GL development team has been actively looking for ways to modernize existing 4GL applications without the need of rewriting existing code or directly converting the 4GL code into (for typical I4GL developers unfamiliar) development environments like pure Java or similar. Shortly after the Informix acquisition through IBM in 2001, the Informix 4GL lab team joined forces with the EGL development team to jointly develop an enhanced version of EGL in combination with a sophisticated I4GL to EGL conversion tool.

Starting with version 5.1.2 of EGL, I4GL began to heavily influence the EGL language and since version 6.0, the I4GL to EGL conversion tool comes bundled with the Rational development tools to make a conversion from existing I4GL applications to the new EGL language as easy as possible (see also Figure 9-17 on page 345).

In addition the recommended conversion path from I4GL to EGL, IBM still maintains the *classic* Informix 4GL with (at the time of writing this redbook) no stated end of life for the product. The latest version of I4GL is 7.32 which is also the only supported I4GL version for conversions into EGL.

9.6.1 Why convert 4GL to EGL for SOA integration?

The Informix 4GL language never provided the required technology on the I4GL language level to easily integrate with a SOA environment.

Some I4GL developers and customers came up with some very creative ideas of adding SOA support to existing applications by either extending their 4GL applications with external C-functions (which is supported through the I4GL language) for services consumption or to write complex wrappers around I4GL applications to be able to offer service interface to already existing valuable business logic written in I4GL.

On the other hand, EGL supports a very easy integration of Web services on an EGL language level (see 5.5.2, “EGL and Web services support” on page 130 for more details).

Since the conversion process from I4GL to EGL leaves the existing 4GL business logic as is and since the converted EGL application behaves like the original I4GL application, it makes lots of sense to use this approach for easy SOA integration.

In the next sections we will discuss the required steps to successfully approach such a solution and we will show based on a simple I4GL banking demo application what's needs to be done to the achieve the SOA integration goal.

9.6.2 What the required steps are

Before we start with the actual application example, let us briefly look at the required steps to a) successfully convert the existing I4GL application to EGL, and then b) how to further enhance that converted application towards Web services integration.

For a) you need to do the following:

1. Locate all the required Informix 4GL files (the actual .4gl modules and libraries, .per forms definition files, message files and so on.)
2. Make sure that the I4GL application compiles without any major problem with a recent I4GL 7.32 compiler version
3. Extract the database schema of the I4GL application database for the following conversion steps
4. Convert all existing I4GL libraries into EGL
5. Now convert the actual I4GL application to EGL

For b) you should follow these steps:

1. Create a new EGL Web Project to act as a container for your to be provided Web service.
2. Include a reference to the necessary EGL projects which have been create in part a) to allow the access to the converted I4GL business logic (which is now EGL coded).
3. Create a new EGL service part as a wrapper around your existing former I4GL functions that you want to expose as a Web service.
4. Create the Web services WSDL file.
5. And finally, deploy the newly created EGL Web service to your Web application server (for example, WebSphere Application Server).

That is basically it! Depending on the complexity of your existing I4GL business logic and your choice of which part of your application you want to expose as a Web service, this process can take only a few minutes without the need to do complex additions or changes to the converted code.

15.3, "Scenario exposing I4GL business logic as Web services" on page 442 leads you through a scenario which exposes a simplified banking application as a Web Service using EGL.

Part 4

Setting up the environment

In Part 4 we describe how to set up the environment for an SOA solution involving the platforms and the tools for our data servers.

© Copyright IBM Corp. 2006. All rights reserved.

10

The z/OS products for SOA

In this chapter we describe the z/OS products that IBM currently provides to support SOA implementation on a mainframe, and which products we used in this redbook for accessing the Data Servers. We explain what you need to set up to enable SOA and Web services in the z/OS environment.

This chapter contains these topics:

- The z/OS products for SOA implementation
- Setting up Web services in z/OS

© Copyright IBM Corp. 2006. All rights reserved.

10.1 The z/OS products for SOA implementation

A rich set of SOA-compliant products introduces service oriented offering and new process management capabilities to the enterprise-class mainframe customer. For example, WebSphere Asset Analyzer helps identify modules for reuse. WebSphere Developer for zSeries helps create Web service interfaces, rich new Web user interfaces and service flows for existing CICS and IMS applications. And the latest versions of CICS, IMS, DB2 and MQ are enabled for SOA to deliver legendary mainframe availability, security, and recoverability to an SOA solution.

Other System z/OS products featured in the announcement include: enhancements to the WebSphere Business Modeler (a simple to use process modeling for the business analyst); a new WebSphere Integration Developer (replacing WebSphere Application Developer Integrated Edition) that gives easy-to-use integration to simplify and speed the assembly of composite applications; a new WebSphere Process Server (replacing WebSphere Business Integration Server Foundation) for flexible deployment of business processes; and an enhanced WebSphere Business Monitor for real-time visibility into process performance enabling process intervention and continuous improvement. All of these elements are connected through the new WebSphere ESB, which provides connectivity infrastructure for integrating applications and services to power your SOA.

WebSphere Enterprise Service Bus (ESB) that connects applications with standards-based interfaces. It delivers a new version of WebSphere Message Broker providing universal connectivity along with advanced message routing, transformation, and enrichment. WebSphere Message Broker is unmatched in its ability to integrate virtually the entire enterprise whether they are standards-based or not.

Figure 10-1 shows the z/OS environment with the main products in general that IBM provides to enable an environment to use SOA inside mainframe.

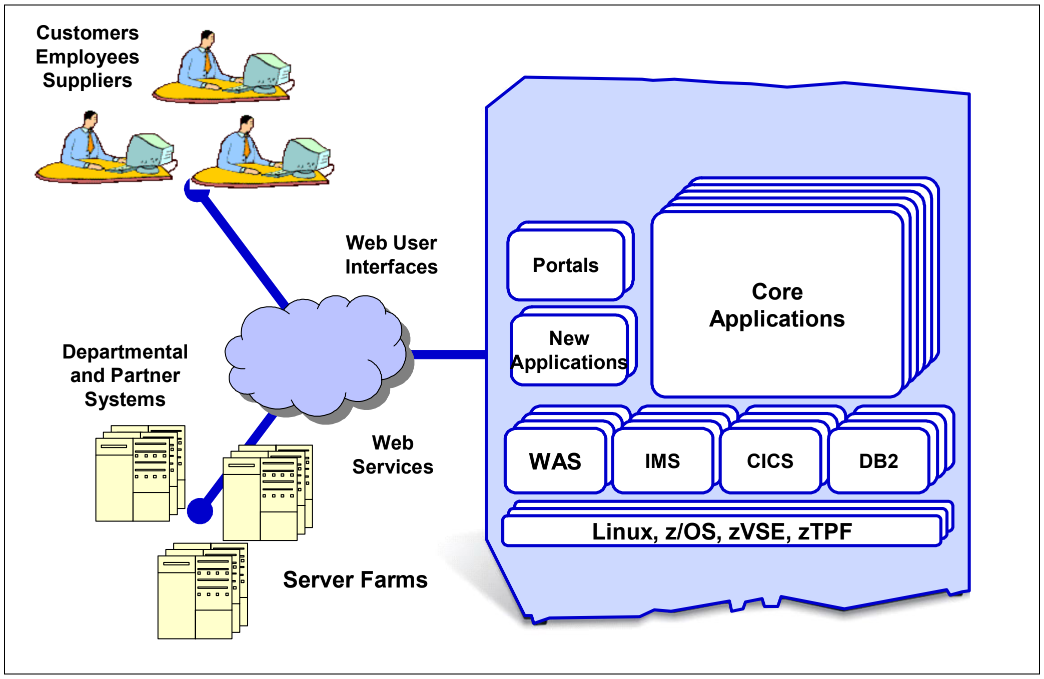


Figure 10-1 z/OS products providing SOA

IBM provides a general SOA foundation family with tools and features to z/OS.

Table 10-1 show these products.

Table 10-1 SOA z/OS products

Product Name	**Current Version**
Model	
WebSphere Business Modeler Advanced	5.1.1.2
Rational Software Architect	6.0.1
Assemble	
WebSphere Integration Developer	6.0
WebSphere Developer for zSeries	6.0
Deploy	
WebSphere Application Server for z/OS	6.0.2
WebSphere Process Server for z/OS	6.0
WebSphere Enterprise Service Bus for z/OS	6.0.1
WebSphere Message Broker for z/OS	6.0
CICS Transaction Gateway for z/OS	6.0
WebSphere Partner Gateway	6.0
WebSphere Adapters (on WPS & ESB for z/OS)	6.0
WebSphere Portal Server Enable for z/OS*	5.1.0.2
WebSphere Everyplace Deployment	6.0
Workplace Collaboration Services	2.5
WebSphere Extended Deployment for z/OS	6.0.1
WebSphere Information Integrator Classic Event Publisher	8.2.3
Manage	
WebSphere Business Monitor	4.2.4
Tivoli Composite Application Manager for SOA	6.0
Tivoli Composite Application Manager for WebSphere	6.0
Tivoli Federated Identity Manager	6.0
Tivoli Access Manager for e-business	5.1

10.1.1 z/OS products used in this redbook

In this section we discuss which products we used to enable z/OS to Web services.

Table 10-2 on page 351 shows the z/OS products used for this redbook.

Table 10-2 The z/OS products and versions

Product Name	**Version**
z/OS	1.7

Product Name	Version
Work Load Manager	
DB2 for z/OS	8 or higher
IMS	9.1
IMS JDBC Connector (IMS Java)	
DB2 Universal JDBC Driver Provider	
IBM XML Toolkit for z/OS, C++ Edition	1.8
XML Toolkit for z/OS and OS/390	1.6 or higher
IBM SDK Java 2 Technology Edition	1.3.1 or higher

10.2 Setting up Web services in z/OS

In this section we describe how to setup a Web service in the z/OS environment, see these sections to perform it:

- Enabling Web service provider in DB2 for z/OS
- Enabling Web service consumer in DB2 for z/OS
- Installing DB2 Universal JDBC in z/OS
- Installing IMS Java

10.2.1 Enabling Web service provider in DB2 for z/OS

Enabling Web service provide in DB2 allows you to create Web services on z/OS with your DB2 data and applications.

It has the following prerequisites:

- Enable JDBC in DB2
- Install WebSphere Application Server Version 5 or later
- Download mail.jar from JavaMail™ Version 1.2 or later
- Download activation.jar from JavaBeans Activation Framework Version 1.0.1 or later

Note: If mail.jar is not present, download JavaMail Version 1.2 or later. If #activation.jar is not present, download JavaBeans Activation Framework Version 1.0.1 or later. You can download these products from the following Web site:

http://java.sun.com

To use DB2 Web Services Object Runtime Framework (WORF), you need to make the runtime services available to WebSphere Application Server (WAS). By default, WORF is installed in the HFS directory:

```
/usr/lpp/db2/db2910_worf/
```

The base install directory contains the lib/ subdirectory that contains the runtime JAR file worf.jar.

To begin using WORF, copy worf.jar, mail.jar, and activation.jar to a WebSphere Application Server shared library directory that you have already set up and restart WAS.

WORF provides an application that contains sample Document Access Definition Extension (DADX) files in the following directory:

```
lib/services.war
```

To set up this application with sample DADX files:

1. Follow the instructions in the job prolog to customize and run job DSNTEJWS, #which is located in the prefix.SDSNSAMP directory. #DSNTEJWS creates the DB2 tables that are used by the sample application.
2. By default, the sample application uses the former or existing JDBC driver connectivity # to connect to a DB2 server. If you configure WebSphere Application Server with JDBC providers that #use the universal JDBC driver, you need to perform the following actions to #configure the sample application to use the universal JDBC driver:
   a. Copy services.war to a temporary directory.
   b. Unjar services.war by issuing the following command:
      ```
      jar -xvf services.war
      ```
   c. Open the group.properties files, which are located in the following #directories:
      ```
      WEB-INF/classes/groups/dxx_sample
      WEB-INF/classes/groups/dxx_travel
      ```
      Modify the dbDriver, dbURL userID, #and password fields to have the following values:
      ```
      dbDriver=com.ibm.db2.jcc.DB2Driver
      dbURL=jdbc:db2://server:port/database
      userID=DB2 userid
      password=DB2 userid password
      ```
   d. Jar the files by issuing the following command:
      ```
      jar -cvf services.war *
      ```
3. Use a Web browser to connect to your WebSphere Application Server Administrative Console.
4. Under Applications, select Install New Application.
5. Select the Server Path option, and type the location of the services.war file in the text box. If you installed WORF in the default location, the server path is:
   ```
   /usr/lpp/db2810_worf/lib/services.war
   ```
6. Fill in a context root for the application (for example, services). Click Next.
7. On the following screens, respond as necessary for your local setup. You #can accept the default settings.
8. On the last screen, click Finish. Click Save to Master Configuration to apply your changes.

To load the application:

1. On the WebSphere Application Server Administrative Console's main page, under Applications, select Enterprise Applications. Select the application and click **Start** to load the application.
2. After the application loads, point a browser to your server with the context root that you chose (for example, http://server/services/). The welcome page lists the sample DADX files that are provided in services.war.

Note: WORF and WebSphere setup are covered with more informations on Chapter 12, "WebSphere Application Server" on page 369.

10.2.2 Enabling Web service consumer in DB2 for z/OS

Enabling Web service consumer in DB2 allows you to receive Web service data in your DB2 applications.

It has the following prerequisites:

- Install z/OS V1R4 or later.
- Install IBM XML Toolkit for z/OS, C++ Edition 1.8.
- Install XML Toolkit for z/OS and OS/390 V1.6 or later.
- Configure TCP/IP.
- Installation job DSNTIJMV contains sample startup procedure DSNWLM, which you can use as a model for your startup procedures. Change the following items in each startup procedure:
  a. Change the procedure name from DSNWLM to the procedure name that you specified when you set up the WLM environment.
  b. Change the value of APPLENV to the name of the WLM environment that you set up for the Web services consumer user-defined functions.This name must match the name in the WLM ENVIRONMENT parameter in the CREATE PROCEDURE statement in DSNTIJIM.
  c. Change the value of DB2SSN to your DB2 subsystem name.
  d. Add the data set name of the XML Toolkit load library to the STEPLIB concatenation. If you used the default data set names when you installed the XML Toolkit, the load library data set name is userid.SIXMLOD1.
- After you set up the WLM application environment, create a JCL startup procedure for the stored procedure address space.
- To create the load modules to enables Web service consumer to DB2, customize job DSNTIJWL as indicated in its prolog and run the job.
- To define the Web service consumer user-defined functions to DB2, customize job DSNTIJWS as indicated in its prolog and run the job.

Attention: Be sure to apply Language Environment® APARs PQ63045 and PQ84190 on z/OS.

10.2.3 Installing the DB2 Universal JDBC driver

The procedures in this section describe what you need to do to install the DB2 Universal JDBC Driver. to provide DB2 Universal JDBC Driver type 2 connectivity and DB2 Universal JDBC Driver type 4 connectivity on a z/OS system that has a DB2 UDB for z/OS subsystem.

To install the DB2 Universal JDBC Driver as part of a DB2 UDB for z/OS installation, follow lthese steps:

1. Install Java 2 Technology Edition, SDK 1.3.1 or higher. If you plan to limplement Java stored procedures and user-defined functions on this DB2 subsystem, install Java 2 Technology Edition, SDK 1.3.1, SDK 1.4.1, or higher.
2. If you plan to use DB2 Universal JDBC Driver type 4 connectivity to connect to DB2 UDB for z/OS Version 7 servers, install OS/390 Support for Unicode or z/OS Support for Unicode on those servers.
3. On z/OS, enable TCP/IP.

4. When you allocate and load the DB2 UDB for z/OS libraries, include the steps |that allocate and load the DB2 Universal JDBC Driver libraries.
5. On DB2 UDB for z/OS, enable distributed data facility (DDF) and TCP/IP support.
6. On DB2 UDB for z/OS, set subsystem parameter DESCSTAT to YES. DESCSTAT corresponds |to installation field DESCRIBE FOR STATIC on panel DSNTIPF.
7. In z/OS UNIX System Services, edit your profile file to customize the environment variable |settings. You use this step to set the libraries, paths, and files that the DB2 Universal JDBC Driver |uses.
8. Optionally you can customize the DB2 Universal JDBC Driver configuration properties.
9. On DB2 UDB for z/OS, enable the DB2-supplied stored procedures and define the |tables that are used by the DB2 Universal JDBC Driver.
10. In z/OS UNIX System Services, run the DB2binder utility to bind the packages for the DB2 Universal JDBC Driver.
11. If you plan to use Universal Driver type 4 connectivity to implement distributed |transactions against DB2 UDB for OS/390 and z/OS Version 7 servers: In z/OS UNIX System Services, |run the DB2T4XAIndoubtUtil utility on the z/OS system on which you are |installing z/OS Application Connectivity to DB2 for z/OS. Run the utility once for each of the DB2 UDB for OS/390 |and z/OS Version 7 servers.
12. If you plan to use LOB locators to access DBCLOB columns in DB2 tables on DB2 UDB for z/OS servers: In z/OS UNIX System Services, run the |DB2LobTableCreator utility on each of those servers to create tables that |are needed for fetching LOB locators.
13. Verify the installation by running a simple JDBC application.

Important: See *DB2 UDB for z/OS Version 8 Installation Guide,* GC18-7418 for information about all these steps.

10.2.4 Binding the DB2 Universal JDBC driver packages

You must bind your JDBC driver packages on your z/OS systems.To bind the JDBC driver packages on z/OS systems, complete the following steps:

1. If you are not using the latest level JDBC driver, migrate your z/OS JDBC Driver to the new DB2 Universal JDBC Driver. Change the DB2_HOME environment variable in the JAVAENV data set to JCC_HOME. For example, change DB2_HOME=/u/DSN/DSN7/usr/lpp/db2/db2710 to JCC_HOME=/u/DSN/DSN7/usr/lpp/db2/db2710/jcc.
2. Bind the required DB2 Universal JDBC driver packages. The Universal JDBC driver does not use the same JDBC packages as the z/OS JDBC drivers. It also uses a different process for binding its packages. The Universal JDBC driver uses a DRDA-based connection to the target DB2 system when creating the required packages.

There are two ways to bind the JCC drivers. Run the supplied batch job AHXJBIND or complete the following steps, which require UNIX System Services (USS) access.

- Update your CLASSPATH and PATH.

  If you are using your own USS directory to perform the bind, then you need to make sure that you have the correct CLASSPATH and PATH settings to run the JCC Binder. You can avoid doing this manually by ensuring that the USS profile is updated correctly with the CLASSPATH and PATH information or by ensuring that your USERIDs profile contains the correct CLASSPATH, LIBPATH and PATH information.

```
CLASSPATH
CLASSPATH=/usr/lpp/db2/db2710/jcc/classes/db2jcc_license_cisuz.jar
CLASSPATH=/usr/lpp/db2/db2710/jcc/classes/db2jcc.jar:$CLASSPATH
CLASSPATH=/usr/lpp/Printsrv/classes:$CLASSPATH
PATH
PATH=/usr/lpp/java/IBM/J1.3/bin:$PATH
PATH=/usr/lpp/db2/db2710/jcc/bin:$PATH
```

- Invoke the JDBC bind from the command line; see Example 10-1.

Example 10-1 JDBC binding

```
java com.ibm.db2.jcc.DB2Binder
-url<url> -user<user> -password<pswd> -collection NULLID
```

The URL identifies the target DB2 system. Table 10-3 shows the variable parts of the URL.

Table 10-3 URL variables

Variable	Description
server_name	The IP address or domain name of the host system where the target DB2 system resides.
port_number	The server port number that is configured for the TCP/IP communications on the target DB2 system.
database_name	The database name of the target DB2 system. On DB2 for z/OS systems, this is referred to as a location name.
user	The user name that you use to connect to the target DB2 system. The user name that you specify must have bind authority.
password	The password that you use to connect to the target DB2 system.
collection	The collection ID for the JDBC packages. You must use NULLID.

See Example 10-2 for the invocation.

Example 10-2 JDBC binding

```
java com.ibm.db2.jcc.DB2Binder
-url jdbc:db2://stlmvs1.ibm.com:446/stlec1 -user SYSADM
-password PASSWRD -collection NULLID
```

Edit and run the sample DSNTIJSG bind job:

1. Update all BIND PACKAGE lines in the DSNTIJSG job (BIND PACKAGE(DSNUTIL)) to include the name of the location that you want to bind. For example:

   ```
   BIND PACKAGE(MYLOCATION.DSNUTIL)
   ```

Important: You must insert the location name in front of all the BIND PACKAGE statement packages.

2. Submit the updated DSNTIJSG job.

10.2.5 Installing IMS Java

IMS Java is delivered in a separate FMID. Before you can install the IMS Java FMID with SMP/E, you must prepare HFS, which is described in this topic.

Prerequisite:

Install IMS Version 9 and run the standard IMS IVPs.

Note: For details about how to run the IMS IVPs and all complete setup, see I*MS Version 9: Installation Volume 1: Installation Verification,*GC26-9429-05.

To install IMS Java:

1. Allocate a data set for HFS, see the Example 10-3 that shows the JCL.

Example 10-3 HFSALLOC Job

```
//HFSALLOC JOB parameters
//*************************************************************/
//* To run this job:                                          */
//* 1) Add JOB statement parameters to meet your requirements. */
//* 2) For DSNAME, change hfsdsn to the name of the new file   */
//*    system.                                                 */
//* 3) For VOLUME, change volid to the volser ID of the DASD   */
//*    that will contain the IMS Java HFS data set.            */
//*************************************************************/
//ALLOCATE EXEC PGM=IDCAMS,DYNAMNBR=200
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  ALLOCATE -
      DSNAME('hfsdsn') -
      RECFM(U)         -
      LRECL(0)         -
      BLKSIZE(32760) -
      DSORG(PO)        -
      VOLUME(volid)  -
      DSNTYPE(HFS)   -
      NEW CATALOG      -
      SPACE(15,5) CYL -
      DIR(200)    -
      UNIT(SYSALLDA)
/*
```

2. Define the mount point directory to mount the HFS, see Example 10-4 on page 357.

Example 10-4 HFSMOUNT job

```
//HFSMOUNT JOB parameters
//*************************************************************/
//* To run this job:                                          */
//* 1) Add JOB statement parameters to meet your requirements. */
//* 2) For FILESYSTEM, change hfsdsn to the name of the file   */
//*    system that you specified in the HFSALLOC job.          */
//* 3) For MOUNTPOINT, change /PathPrefix to the high-level    */
//*    directory name. The directory name must be preceded with*/
//*    a forward slash (/), for example, /apps or /ims/apps.   */
//*    This string must match the PathPrefix                  */
//*    string in the DFSJSMKD job.                            */
//**********************************************************/
//MOUNT    EXEC PGM=IKJEFT01
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
  MOUNT FILESYSTEM('hfsdsn')    /* MOUNT  HFS    */ +
  MOUNTPOINT('/PathPrefix') TYPE(HFS) MODE(RDWR)
/*
```

3. Run the sample installation job DFSJSMKD. DFSJSMKD runs the DFSJMKDR REXX script, which creates the HFS paths for IMS Java.
4. Using SMP/E, install the IMS Java FMID.

The next step varies. It depends on the environment that your application will run in:

- JMP region: Running the IMS Java IVP in a JMP Region
- JBP region: Running the IMS Java IVP in a JBP Region
- WebSphere Application Server for z/OS
- WebSphere Application Server on a non-z/OS platform
- DB2 UDB for z/OS stored procedure
- CICS

11

The Linux, UNIX, and Windows products for SOA

In this chapter, we look at the steps to prepare and set up a Web services environment in DB2 V9.1 for Linux, UNIX, and Windows. We also highlight certain new system requirements and discuss important changes from previous versions of DB2 that you need to be aware of.

This chapter contains these topics:

- DB2 V9.1 for Linux, UNIX, and Windows installation and its development software support
- Connecting Web services to DB2 via JDBC
- Preparing the installation of the Web services provider
- Installation of the Web services consumer UDFs
- Migrating from XML Extender

© Copyright IBM Corp. 2006. All rights reserved.

11.1 DB2 V9.1 for Linux, UNIX, and Windows installation and its development software support

DB2 V9.1 for Linux, UNIX and Windows now allows coexistence of multiple DB2 versions and fix packs on Windows, and concurrent copies of DB2 database systems on Linux and UNIX. You can install multiple copies of DB2 database systems on Linux or UNIX operating systems without the need for alternate FixPaks. Key benefits of this feature include:

- Install anywhere: You can install DB2 database systems using any path that you choose.
- Install any number of times: You can install two or more copies of the same database system on one computer. The code levels can be different as well.
- Service each copy independently: You can update one copy without affecting any of the other copies.

This means that database administrators can have independent copies of DB2 database systems for different purposes. This allows different databases on the same computer to run at different fix pack levels. Therefore, you can install DB2 V9.1 for Linux, UNIX and Windows on a system that is already running DB2 V8. Please check DB2 V9.1 for Linux, UNIX and Windows Information Center for prerequisites and system requirements:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp`

Current DB2 system requirements are also available here:

`http://www-306.ibm.com/software/data/db2/udb/sysreqs.html`

11.1.1 Linux and UNIX operating systems

For AIX, HP-UX, Solaris Operating Environment, the default installation path is:

```
/opt/IBM/db2/V9.1
```

For Linux systems, the default installation path is:

```
/opt/ibm/db2/V9.1
```

If you are installing on a system where this directory is already being used, the DB2 product installation path will have _xx added to it, where _xx are digits, starting at 01 and increasing depending on how many DB2 copies you have installed. You can also specify your own DB2 product installation path.

On supported Linux and UNIX operating systems, a new command, **db2ls**, provides information about DB2 database systems and features installed on your system. You can use this command to first list where DB2 database systems are installed and which level of the DB2 database system is installed. Figure 11-1 shows you an example of the **db2ls** output where we have installed DB2 V9.1 for Linux, UNIX and Windows in an installation path we specified.

```
/usr/local/bin> ./db2ls

Install Path         Level    Fix Pack  Special Install Number  Install Date
--------------------------------------------------------------------------------
/usr/opt/db2_09_00   9.0.0.0       0                        0     Fri Mar 31 16:09:56 2006 PST
```

Figure 11-1 db2ls output

Important: In Version 9.1, the `db2ls` command is the only method for querying a DB2 product. You can no longer query DB2 products using Linux or UNIX operating system native utilities such as `pkgadd`, `rpm`, `SMIT`, or `swinstall`. You must change any scripts containing a native installation utility that you use to interface with and query DB2 installations. You cannot use the `db2ls` command on Windows operating systems.

11.1.2 Windows operating systems

Similar to DB2 Version 9.1 on Linux and UNIX systems, you can have multiple DB2 copies on the same Windows system.

When only one DB2 copy is installed, a unique Default DB2 copy name is generated during installation, which you can change later. Applications use the Default DB2 copy in an environment similar to the DB2 Version 8 environment.

When multiple DB2 copies are installed on the same system, DB2 Version 8 can coexist with DB2 Version 9.1, with certain restrictions described below.

- DB2 Version 8 and DB2 Version 9.1 can coexist, but DB2 Version 8 is set as the Default DB2 copy. This cannot be changed unless you uninstall Version 8.
- Each DB2 copy must have unique instance names.

Restriction: For Win x64 platforms, DB2 does not support multiple DB2 32-bit and 64-bit versions installed on Windows system. If you install the DB2 64-bit version, the 32-bit version will be removed from the system.

11.2 Connecting Web services to DB2 via JDBC

To develop and deploy Java applications that run against DB2 databases, you need to use supported development software and operating systems. During the installation process for DB2 for Linux, UNIX and Windows, the installation process also installs the latest SDK for Java that is available. Table 11-1 lists the DB2 V9.1 for Linux, UNIX and Windows supported SDK levels for Java.

Table 11-1 SDK for Java support for the IBM DB2 Driver for JDBC and SQLJ

Operating system	31-bit, 32-bit, or 64-bit operating system	Supported levels of SDK for Java for the IBM DB2 Driver for JDBC and SQLJ
AIX 5	32-bit/64-bit	1.4.2, 5
HP-UX 11i	32-bit/64-bit	1.4.2
Linux on Intel x86	32-bit	1.4.2, 5
Linux on IA64	64-bit	1.4.2
Linux on AMD64/EM64T	32-bit/64-bit	1.4.2, 5
Linux on PowerPC®	32-bit/64-bit	1.4.2, 5
Linux on zSeries	31-bit/64-bit	1.4.2, 5
Solaris	32-bit/64-bit	1.4.2

Windows on Intel x86	32-bit	1.4.2, 5
Windows on IA64	64-bit	1.4.2
Windows on x64	32-bit/64-bit	1.4.2, 5

Any JVMs that run Java applications that access DB2 databases must include native threads support. You can specify native threads as the default thread support for some JVMs by setting the THREADS_FLAG environment variable to "native".

Before you begin using the DB2 JDBC and SQLJ Driver, ensure you have setup the following:

- Include db2jcc.jar and sqlj.jar in your CLASSPATH
- Include license files for the IBM DB2 Driver for JDBC and SQLJ

Table 11-2 IBM DB2 Driver for JDBC and SQLJ license file

License File	Server to which license file permits a connection	Product that includes license file
db2jcc_license_c.jar	Cloudscape(TM)	Cloudscape Network Server
db2jcc_license_cu.jar	Cloudscape All DB2 for Linux, UNIX, and Windows servers	All DB2 for Linux, UNIX, and Windows products
db2jcc_license_cisuz.jar	Cloudscape All DB2 for Linux, UNIX, and Windows servers DB2 for z/OS DB2 UDB for iSeries	All DB2 Connect products

  On Windows: The files are located in %DB2PATH%\sqllib\java (where %DB2PATH% is a variable that determines where DB2 is installed)

  On UNIX: The files are located in $HOME/sqllib/java (where $HOME is the home directory of the instance owner)

- Install IBM DB2 Driver for JDBC and SQLJ native libraries for support of IBM DB2 Driver for JDBC and SQLJ type 2 connectivity.

  For AIX, HP-UX on IPF, Linux, and Solaris: libdb2jcct2.so

  For HP-UX on PA-RISC: libdb2jcct2.sl

  For Windows: db2jcct2.dll

  On Windows: The files are located in %DB2PATH%\sqllib\bin (where %DB2PATH% is a variable that determines where DB2 is installed)

  On UNIX: The files are located in $HOME/sqllib/lib (where $HOME is the home directory of the instance owner)

If you use the Type 4 connectivity, it is especially important to configure TCP/IP:

```
db2set DB2COMM=TCPIP
```

```
db2 update DBM CFG using SVCENAME <TCP/IP service name>
```

If you plan to run Java procedures and UDFs then also ensure the JDK_PATH in the DBM CFG is setup to include the PATH where the SDK for Java is located.

For example on Windows:

```
db2 update dbm cfg using JDK_PATH c:\Program Files\jdk142
```

And on UNIX:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk142
```

If you plan to run Java stored procedures that work with XML data on DB2 for Linux, UNIX, and Windows servers, you need to set the IBM DB2 Driver for JDBC and SQLJ as the default JDBC driver for running stored procedures. To do that, set the *DB2_USE_DB2JCCT2_JROUTINE* environment value to YES, yes, ON, on, TRUE, true, or 1.

To set the IBM DB2 Driver for JDBC and SQLJ as the default driver at the instance level:

db2set DB2_USE_DB2JCCT2_JROUTINE=YES -i ***instance-name***

Where ***instance-name*** is the actual instance name.

To set the IBM DB2 Driver for JDBC and SQLJ as the default driver at the global level:

```
db2set DB2_USE_DB2JCCT2_JROUTINE=YES -g
```

11.2.1 Changes to development software support

DB2 V9.1 for Linux, UNIX and Windows supports C, C++, Java, COBOL, Fortran, REXX, Perl, PHP and .NET languages. Refer to the following URL for more information about the latest development software support.

http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.rn.doc/doc/c0023012.htm

11.3 Preparing the installation of the Web services provider

The Web services provider is part of DB2 Version 9.1. The Web services provider is in the following path in DB2 Version 9.1:

On Windows: %DB2PATH%\sqllib\samples\webservices\dxxworf.zip (where %DB2PATH% is a variable that determines where DB2 is installed)

On UNIX: $HOME/sqllib/samples/webservices/dxxworf.zip (where $HOME is the home directory of the instance owner)

The DB2 Web Service provider is an extension to Java application servers such as WebSphere Application Server and Jakarta Tomcat. After you have located dxxworf.zip, you will unzip the zip file to extract worf.jar and copy into your WebSphere Server or Tomcat environment.

You can install the Web services provider on any of the following operating systems:

- Windows NT®
- Windows 2000
- Linux
- AIX
- Solaris Operating Environment
- OS/390 Version 2.8 or later
- z/OS Version 1.1 or later

You can use the following database environments:

- IBM DB2 Version 91 or later
- DB2 Universal Database for OS/390 Version 8
- DB2 Universal Database for z/OS
- Informix Dynamic Server (IDS) Version 9.3

See Chapter 10, "The z/OS products for SOA" on page 349 for detail specific to the z/OS platform. Also see Chapter 9, "Informix IDS and SOA" on page 297 for detail specific to Informix Dynamic Server (IDS).

And supported Web servers are:

- WebSphere Application Server Advanced Edition Version 6
- Apache Jakarta Tomcat Version 3.3.1 through 4.0.3 or later
- Application server for DB2

For detail instructions on how to setup Web service provider on the Web servers, refer to the following URLs for specific Web server information:

Application Server for DB2:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.websphere.ii.db2udb.foundation.appdev.fed.ws.doc/developing/applicationserverfordb2.html`

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

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.websphere.ii.db2udb.foundation.appdev.fed.ws.doc/installing/tiiwrfin.html`

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

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.websphere.ii.db2udb.foundation.appdev.fed.ws.doc/installing/tiiapins.html`

11.4 Installation of the Web services consumer UDFs

The **`db2enable_soap_udf`** command enables your database to use the SOAP requester functions and installs the Web services consumer user-defined functions (UDFs) to the particular database. You should install the following software before running the SOAP UDFs:

- DB2 Version 8
- DB2 V9.1 for Linux, UNIX and Windows

On DB2 V9 you may *optionally* enable DB2 XML Extender in order to continue accessing XML Extender features, or for backward compatibility with applications that used to run on DB2 V8. Refer to further information at 11.5, "Migrating from XML Extender" on page 366.

You *must additionally* enable DB2 XML Extender with the **`dxxadm enable_db`** command on DB2 V8.

To enable, or install, and disable the Web service consumer issue the following:

```
db2enable_soap_udf -n dbName [-u uID] [-p password] [-force]
```

To disable, or uninstall the Web service consumer issue the following:

```
db2disable_soap_udf -n dbName [-u uID] [-p password]
```

Where:

dbName

A database name

uID

Optional: User ID

password

Optional: The password associated with the user ID

–force

Attempts to drop any existing functions.

You can also enable the Web services consumer UDFs from DB2 Control Center as shown in Figure 11-2 on page 365.

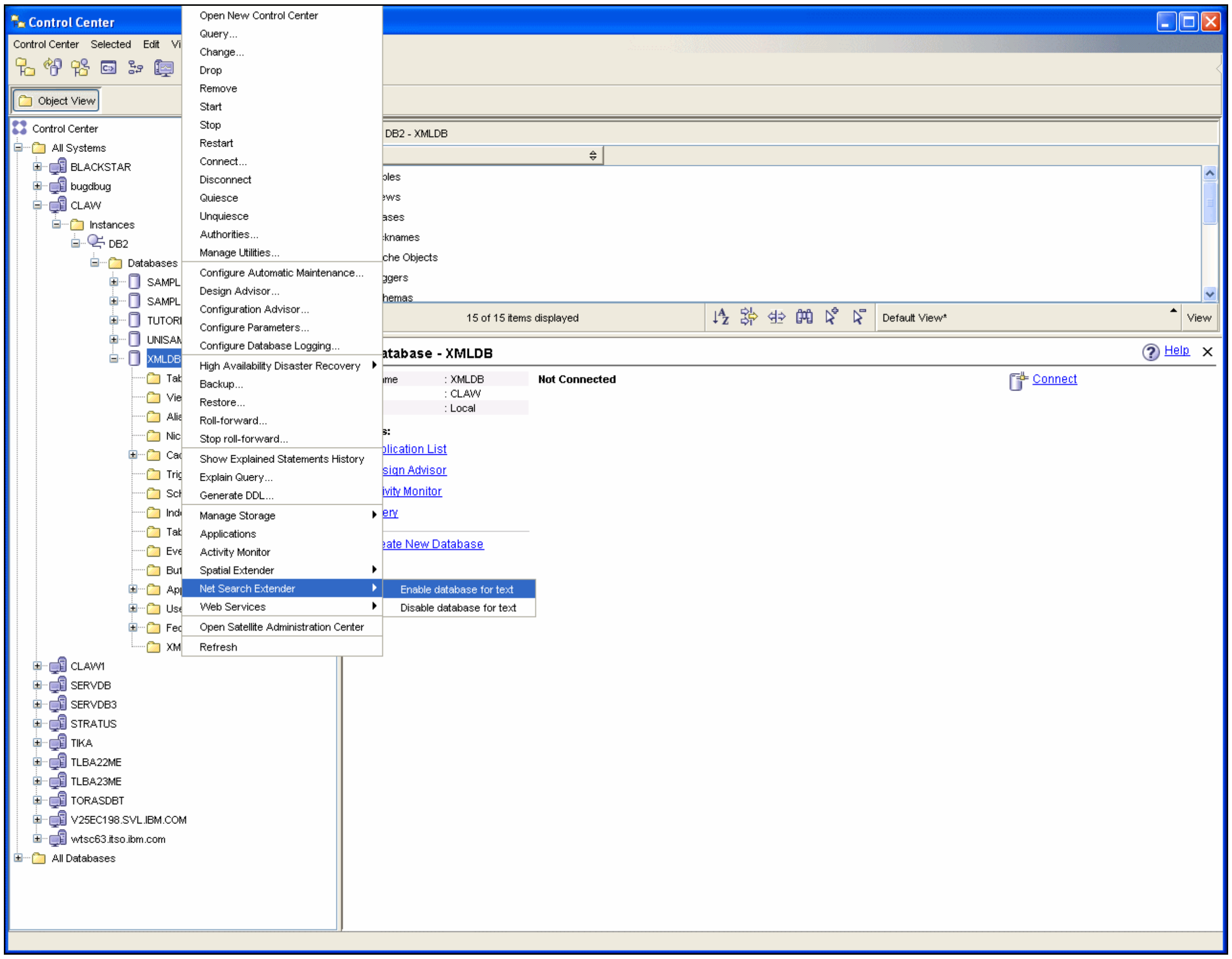


Figure 11-2 Enabling Web service in DB2 Control Center

11.5 Migrating from XML Extender

DB2 Version 9.1 supports native XML storage in an annotated tree form similar to that of the XML Document Object Model (DOM). This support includes a new XML type, XML indexes, SQL/XML functions and XQueries. You can migrate your database application from XML Extender to use the native XML storage in DB2 Version 9.1 to take full advantage of the native XML store.

Important: Although XML native support is introduced in DB2 Version 9.1 for Linux, UNIX and Windows, you can still use XML Extender. To enable DB2 V8 XML Extender functions for a particular database, issue:

```
dxxadm enable_db <database>
```

Where **`<database>`** is the database name.

The steps to migrate from XML Extender are:

1. Migrate to DB2 Version 9.1 for Windows or migrate to DB2 Version 9.1 for Linux and UNIX.

   On Linux and UNIX, you can migrate to DB2 Version 9.1 from a DB2 server that has multiple copies of DB2 Enterprise Server Edition (ESE) Version 8. If you have installed several alternate fixpaks as a completely new copy of DB2 ESE, you could have multiples copies of DB2 ESE on the same DB2 server. You need to manually migrate any DB2 UDB Version 8 instances to a DB2 Version 9.1 copy of your choice.

   You can manually migrate a DB2 UDB Version 8 instance at any fixpak level by executing the **`db2imigr`** command from the target DB2 Version 9.1 copy of your choice. Once an instance is migrated to a DB2 Version 9.1 copy, you cannot migrate to another DB2 Version 9 copy. Also you cannot migrate to DB2 UDB Version 8. However, you can upgrade an instance between different DB2 copies of DB2 Version 9.1 using the **`db2iupdt`** command.

   On Windows, migration is required if you have instances and databases running on DB2 UDB Version 8 that you want to run on DB2 Version 9.1. If you choose to automatically migrate your existing DB2 UDB Version 8 copy during the DB2 Version 9.1 installation, your instances and DB2 administration server (DAS) are migrated but you still need to migrate your databases after installation. If you choose to install a new DB2 Version 9.1 copy, you must manually migrate your instances, your DAS and databases.

   Restriction: Migration is supported only from DB2 UDB Version 8. Direct migration is not supported from DB2 UDB Version 7 or earlier. You must migrate first to DB2 UDB Version 8.

2. Convert your databases to Unicode databases. If you created your databases in DB2 UDB Version 8 as a Unicode database, you can start using the XML functionality with your migrated database. Otherwise, you must export your database, create your database again by running the CREATE DATABASE with the clause USING CODESET utf-8 TERRRITORY *territory*, and then load your data. This is due to the restriction of the native XML functionality is only supported on Unicode databases. For example:

   ```
   CREATE DATABASE xmldb USING CODESET UTF-8 TERRITORY US
   ```

3. Add XML type columns to your tables. Use the ALTER TABLE command:

   `ALTER TABLE` ***table_name*** **`ADD`** ***column_name*** **`XML [NOT NULL]`**

   Where:

table_name

Is the name of the existing table with data.

column_name

Is the name of the new XML column you want to add.

You only need to perform this step if you store intact your XML documents in a column of data type CLOB, VARCHAR, XMLCLOB, XMLVARCHAR, or XMLFILE.

4. Register your XML schemas in the XML Schema repository (XSR). This is analogous to the step of **`dxxadm enable_collection`** in XML Extender. If you have document type definitions (DTDs), you must convert them to XML schemas and then register them in the XSR. You only need to perform this step only if you want to validate your XML documents.
5. Import XML documents into the table with the new XML data type column.
6. Convert your application to use annotated XML schema decomposition to store content from XML documents in table columns, and the new SQL/XML functions to construct or publish XML using the new XML data type.

Details on all these migration steps and examples of application migration are available in a white paper series published at developerWorks Information Management. You can check this URL for more information. At the time of writing this redbook, DB2 V9.1 for Linux, UNIX and Windows is still a beta version, thus the migration paper is not yet available. Please check this URL frequently for updates.

`http://www-1.ibm.com/support/docview.wss?rs=73&uid=swg21200005`

The DeveloperWorks article *From DAD to annotated XML schema decomposition* at the following URL which also includes a DAD to annotated XML schema converter tool which will help you convert your XML schemas to annotated XML schemas based on the mapping rules described in the DAD.

`http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604pradhan/`

12

WebSphere Application Server

This chapter introduces the IBM WebSphere Application Server (WAS) Version 6. We provide a general overview of the products and talk in detail about the WebSphere Application Server for z/OS-specific product functions.

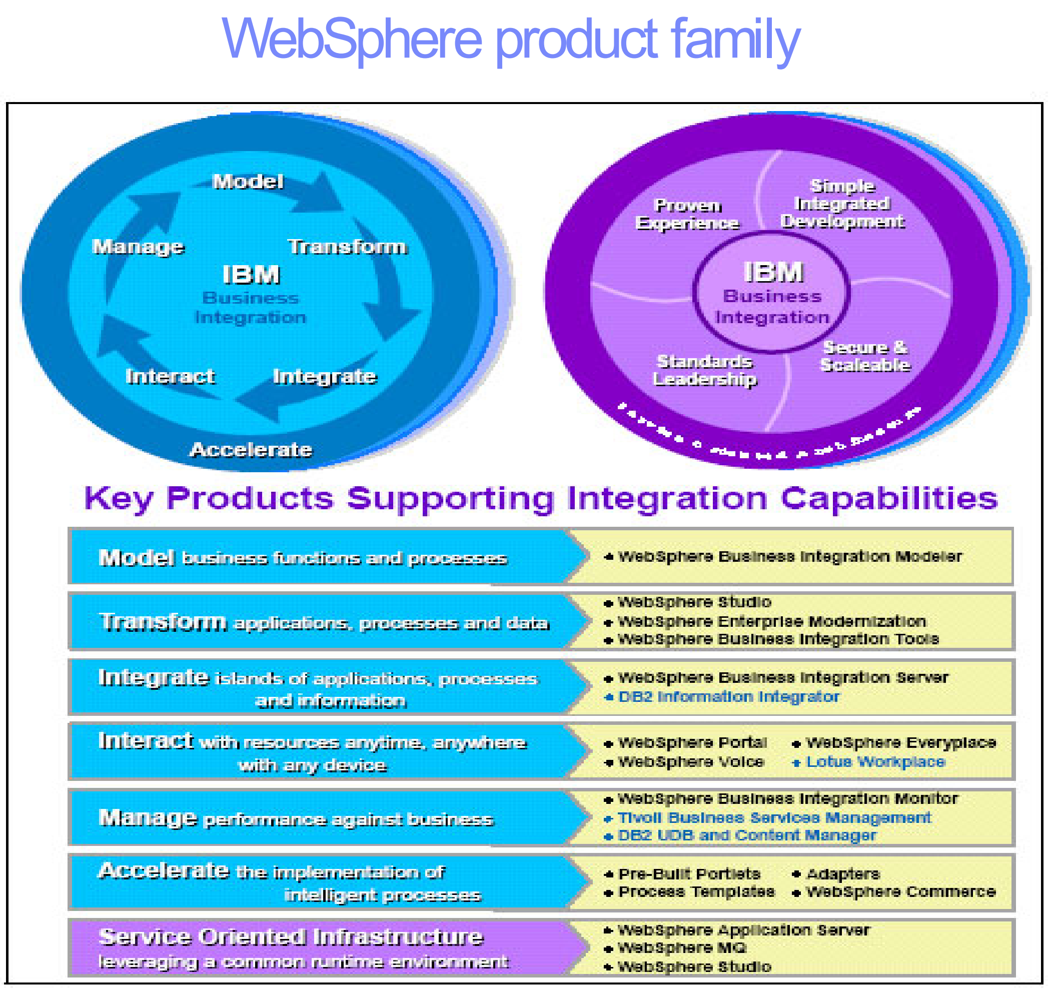
This chapter contains these topics:

© Copyright IBM Corp. 2006. All rights reserved.

12.1 Introduction to WebSphere Application Server Version 6

As the foundation of the WebSphere software platform, WebSphere Application Server Version 6.0 is one of the industry's premier Java-based application platforms, integrating enterprise data and transactions for the dynamic e-business world. Each configuration available delivers a rich application deployment environment with application services that provide enhanced capabilities for transaction management, as well as the security, performance, availability, connectivity, and scalability expected from the WebSphere family of products.

Figure 12-1 shows a high-level overview of the WebSphere platform.



Model business functions and processes	• WebSphere Business Integration Modeler
Transform applications, processes and data	• WebSphere Studio • WebSphere Enterprise Modernization • WebSphere Business Integration Tools
Integrate islands of applications, processes and information	• WebSphere Business Integration Server • DB2 Information Integrator
Interact with resources anytime, anywhere with any device	• WebSphere Portal • WebSphere Voice • WebSphere Everyplace • Lotus Workplace
Manage performance against business	• WebSphere Business Integration Monitor • Tivoli Business Services Management • DB2 UDB and Content Manager
Accelerate the implementation of intelligent processes	• Pre-Built Portlets • Process Templates • Adapters • WebSphere Commerce
Service Oriented Infrastructure leveraging a common runtime environment	• WebSphere Application Server • WebSphere MQ • WebSphere Studio

Figure 12-1 WebSphere product family

WebSphere Application Server is J2EE 1.4 compatible and provides Web services support above and beyond the specification. Further, there are new features for rapid development and deployment, which help reduce development cycle time and maximize the ability to use existing skills and resources.

WebSphere Application Server is available in multiple packages to meet specific business needs. They also serve as the base for other WebSphere products, such as WebSphere Commerce, by providing the application server required for running these specialized applications. WebSphere Application Server is available on a wide range of platforms, including UNIX-based platforms, Microsoft operating systems, IBM z/OS, and IBM eServer™ iSeries.

Figure 12-2 shows WebSphere Application Server product overview by functions.

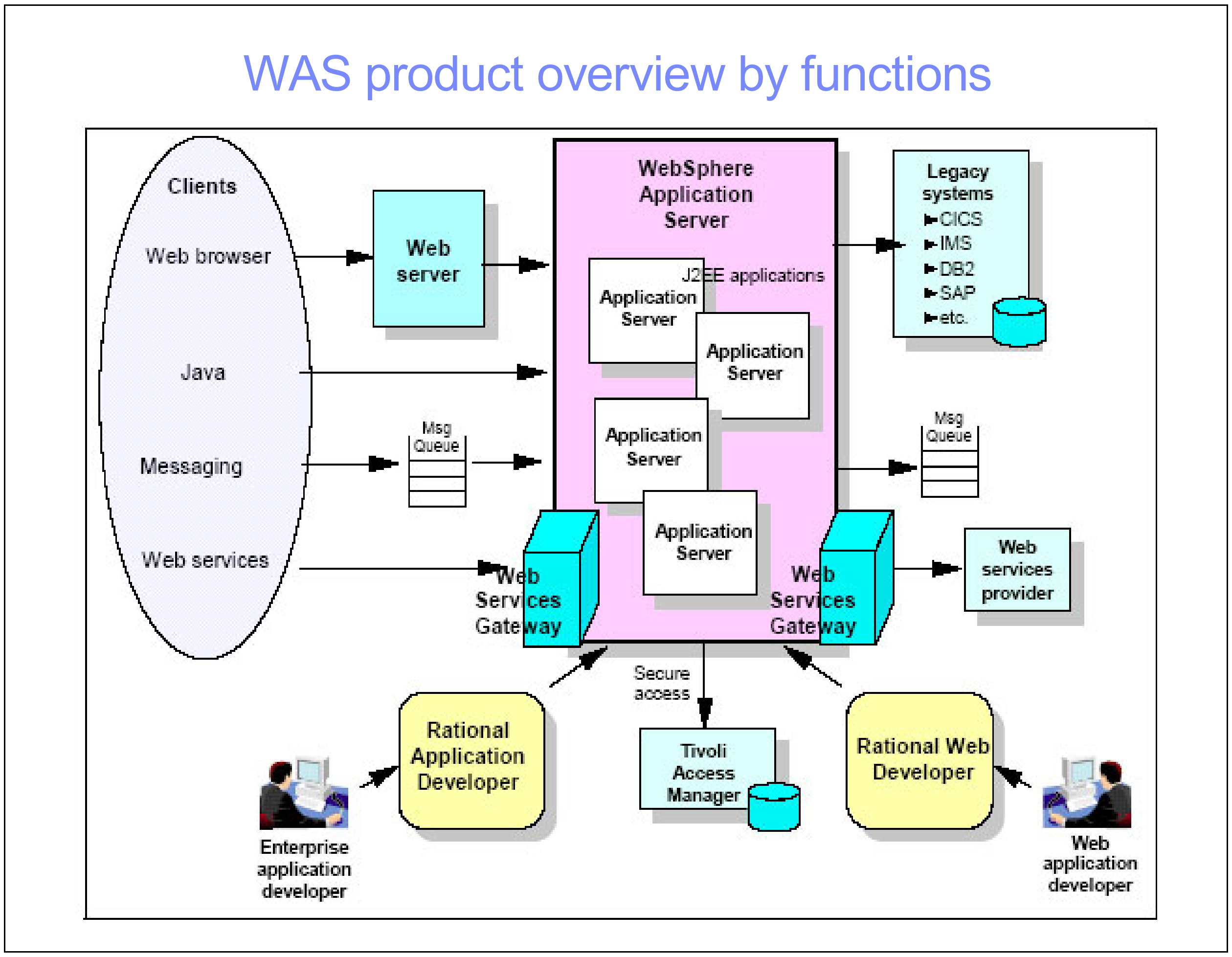


Figure 12-2 WebSphere Application Server product overview by functions

WebSphere Application Server Version 6 has three product offerings. At the heart of each package is a WebSphere Application Server that provides the runtime environment for enterprise applications:

WebSphere Application Server V6 Express - Offers the entry point to e-business, an out-of-the-box solution for managing simple, dynamic Web sites with an easy-to-use Web application server and a development environment. The Express product is fully J2EE 1.4 compliant.

WebSphere Application Server V6 - The core of the WebSphere portfolio, this product is an industry-leading J2EE and Web services application server, delivering a high-performance and extremely scalable transaction engine for dynamic e-business applications.

WebSphere Application Server V6 Network Deployment - Provides an operating environment with advanced performance and availability capabilities in support of dynamic application environments. In addition to all of the features and functions within the base WebSphere Application Server, this configuration delivers advanced deployment services that include clustering, edge-of-network services, Web services enhancements, and high availability for distributed configurations.

In addition, the Network Deployment package contains the Web services gateway. In WebSphere Application Server Version 6, the gateway is now fully integrated in the service integration technology.

12.1.1 What is new in WebSphere Application Server Version 6?

Version 6 includes support for J2EE 1.4, Web services enhancements (WS-AtomicTransaction, WS-Addressing, WS-I Basic Profile 1.1), WebSphere platform messaging with fully integrated Java messaging engine, JavaServer Faces (JSR 127), and standardized logging (JSR 47).

Other enhancements provide the solid base for a robust infrastructure, including a unified clustering framework and fail-over for stateful session EJBs. Additional programming model extensions include Service Data Objects (SDO), activity session, asynchronous beans, dynamic query, application profiling, JTA support, WorkArea service, and scheduler, among others. These features have been part of the WebSphere Server Foundation V5.1 product.

WebSphere Application Server Version 6 supports and complies with the Web service specifications and standards listed in Table 12-1.

Table 12-1 Web services standards in WebSphere Application Server Version 6

Specification or standard	Description
JAX-RPC (JSR-101) 1.1	Additional type support xsd:list Fault support Name collision rules New APIs for creating services isUserInRole()
Web Services for J2EE 1.1 (JSR109, JSR921)	Moved to J2EE 1.4 schema types Migration of Web services client DD moving to appropriate container DDs Handlers support for EJBs Service endpoint interface (SEI) is a peer to LI/RI
SAAJ 1.2	APIs for manipulating SOAP XML messages SAAJ infrastructure now extends DOM (easy to cast to DOM and use)
WS-Security	WSS 1.0 WS-I Security Profile
WS-I Basic Profile 1.1	Attachments support
WS-Transaction	WS-AtomicTransaction
JAXR 1.0	Java client API for accessing UDDI (Version 2 only) and ebXML registries
UDDI V3	Includes both the registry implementation and the client API library

12.2 Highlights and benefits

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. An application server functions as Web middleware or a middle tier in a three-tier e-business environment. The first tier is the HTTP server that handles requests from the browser client. The third tier is the business database (for example, DB2 UDB for iSeries) and the business logic (for example, traditional business applications such as order processing). The middle tier is WebSphere Application Server, which provides a framework for a consistent and structured link between the HTTP requests and the business data and logic.

For WebSphere Application Server for z/OS, possibly in the same logical partition as ultra-high-performance CICS or IMS applications and IMS/DB2 data sources, it is possible to achieve the tightest levels of integration between new and existing corporate systems, with communication at channel speeds, unhampered by network bottlenecks. This has a number of advantages. First and foremost, the performance is unparalleled by running WebSphere Application Server and DB2 together on the mainframe rather than on distributed systems. Secondly, since the application no longer needs to cross a TCP/IP network to retrieve data, there are security and reliability benefits. The data does not need to be encrypted, users require less regular authentication, and the whole end-to-end communication between applications remains unaffected by the prevalence of network failure. Sysplex-enabled mainframe systems consistently offer 99.999% availability and beyond; but if an application needs to interoperate with a network-attached distributed server, this level of availability can decline dramatically.

WebSphere Application Server is intended for organizations that want to take advantage of the productivity, performance advantages, and portability that Java provides for dynamic Web sites. It includes:

- J2EE 1.4 support.
- High-performance connectors to many common back-end systems to reduce the coding effort required to link dynamic Web pages to real line-of-business data.
- Application services for session and state management.
- Web services that enable businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically.
- The WebSphere Platform Messaging infrastructure to complement and extend WebSphere MQ and application server. It is suitable for those that are currently using the WebSphere Application Server V5 embedded messaging and for those that need to provide messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

12.3 Supported platforms and software

The following tables illustrate the platforms, software, and versions that WebSphere Application Server V6 supports at the time of the writing of this document. For the most up-to-date operating system levels and requirements, refer to the following:

`http://www.ibm.com/software/webservers/appserv/`

12.3.1 Operating systems

Table 12-2 shows the supported operating systems and versions for WebSphere Application Server V6.

Table 12-2 Supported operating systems and versions

Operating Systems	Versions
Windows	Windows 2000 Advanced Server with SP4, Windows 2003 Server
AIX	AIX 5.1 Maintenance Level 5100-05 AIX 5.2 Maintenance Level 5200-02, 5200-03
Sun	Solaris Solaris 8 with the latest patch Cluster Solaris 9 with the latest patch Cluster
HP-UX	HP-UX 11i v1 with the latest Quality Pack
Linux (Intel)	Linux (Intel) Red Hat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3, SuSE Linux Enterprise Server 9.0
Linux (Power PC®)	Red Hat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
zLinux (Supported for WebSphere Application Server Network Deployment V6 only)	Red Hat Linux Enterprise 3.0 Update 1 UnitedLinux 1.0 SP3 SuSE Linux Enterprise Server 9.0
i5/OS® and OS/400®	OS/400 5.3, 5.2
z/OS (Supported for WebSphere Application Server Network Deployment V6 only)	z/OS 1.6, 1.5, 1.4 z/OS.e 1.6, 1.5, 1.4

12.3.2 Database servers

Table 12-3 shows the operating systems and versions that WebSphere Application Server V6 supports.

Table 12-3 WebSphere Application Server and operating systems

Databases	Versions
IBM DB2	IBM DB2 Enterprise Server Edition 8.2, 8.1 with FP5 IBM DB2 Workgroup Server Edition 8.2, 8.1 with FP4a, FP5 IBM DB2 Information Integrator 8.2, 8.1 with FP5 IBM DB2 Connect 8.2, 8.1 with FP5 IBM DB2 Express 8.2, 8.1 with FP5 IBM DB2 for zSeries 8, 7 IBM DB2 for iSeries 5.3, 5.3
Cloudscape	Cloudscape 5.1.6x
Oracle	Oracle Standard/Enterprise Edition 10g Release 1 - 10.1.0.2 Oracle 9i Standard/Enterprise Edition Release 2 - 9.2.0.4 Oracle 8i Standard/Enterprise Edition Release 3 - 8.1.7.4

Databases	Versions
Sybase	Sybase Adaptive Server Enterprise 12.5.1, 12.0.0.8
Microsoft SQL Server	SQL Server Enterprise 2000 SP 3a
Informix	Informix Dynamic Server 9.4, 9.3

12.4 WebSphere Application Server V6 architecture

WebSphere Application Server is the implementation by IBM of the Java 2 Enterprise Edition (J2EE) platform. It conforms to the J2EE 1.4 specification. WebSphere Application Server is available in three unique packages that are designed to meet a wide range of client requirements. This discussion centers on following topics:

- Architecture configurations
- z/OS base infrastructure

12.4.1 Architecture configurations

At the heart of each member of the WebSphere Application Server family is an application server with essentially the same architectural structure. While the application server structure is identical for WebSphere Application Server and Express, there are differences in licensing terms, the development tool provided, and platform support. With WebSphere Application Server and Express, you are limited to stand-alone application servers. Each stand-alone application server provides a fully functional J2EE 1.4 environment. Network Deployment adds additional elements that allow for more advanced topologies and that provide workload management, scalability, high availability, and central management of multiple application servers.

Stand-alone server configuration

Express, WebSphere Application Server, and Network Deployment all support a single stand-alone server environment. However, with Express and WebSphere Application Server, this is your only option. In a single stand-alone server environment, each application server acts as a unique entity. An application server runs one or more J2EE applications and provides the services that are required to run those applications.

Multiple stand-alone application servers can exist on a machine, either through independent installations of the WebSphere Application Server code or through the creation of multiple configuration profiles within one installation. However, there is no common management or administration provided for multiple application servers. Stand-alone application servers do not provide workload management or fail-over capabilities.

Figure 12-3 on page 376 shows an architectural overview of a stand-alone application server.

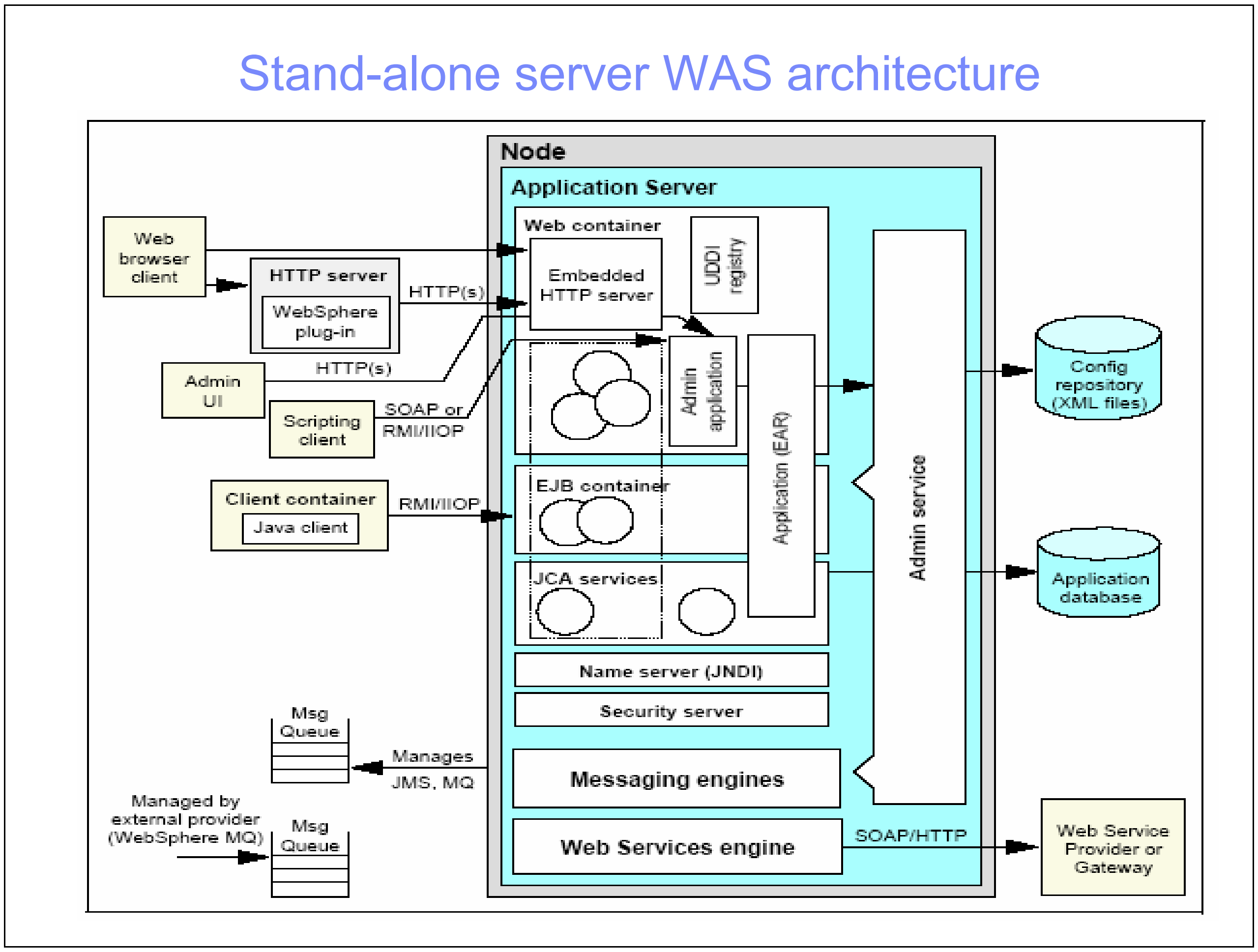


Figure 12-3 Stand-alone server WebSphere Application Server architecture

Distributed server configuration

With Network Deployment, you can build a distributed server configuration, which enables central administration, workload management, and fail-over. One or more application servers are federated (integrated) into a cell and managed by a deployment manager. The application servers can reside on the same machine as the deployment manager or on multiple separate machines. Administration and management is done centrally from the administration interfaces via the deployment manager.

With this configuration, you can create multiple application servers to run unique sets of applications, and you can manage these servers from a central location. However, more importantly, you can cluster application servers for workload management and fail-over capabilities. Applications installed to the cluster are replicated across the application servers. When one server fails, another server in the cluster can continue processing. Workload is distributed among Web containers and Enterprise JavaBean (EJB) containers in a cluster using a weighted round-robin scheme.

Figure 12-4 on page 377 illustrates the basic components of an application server in a distributed server environment.

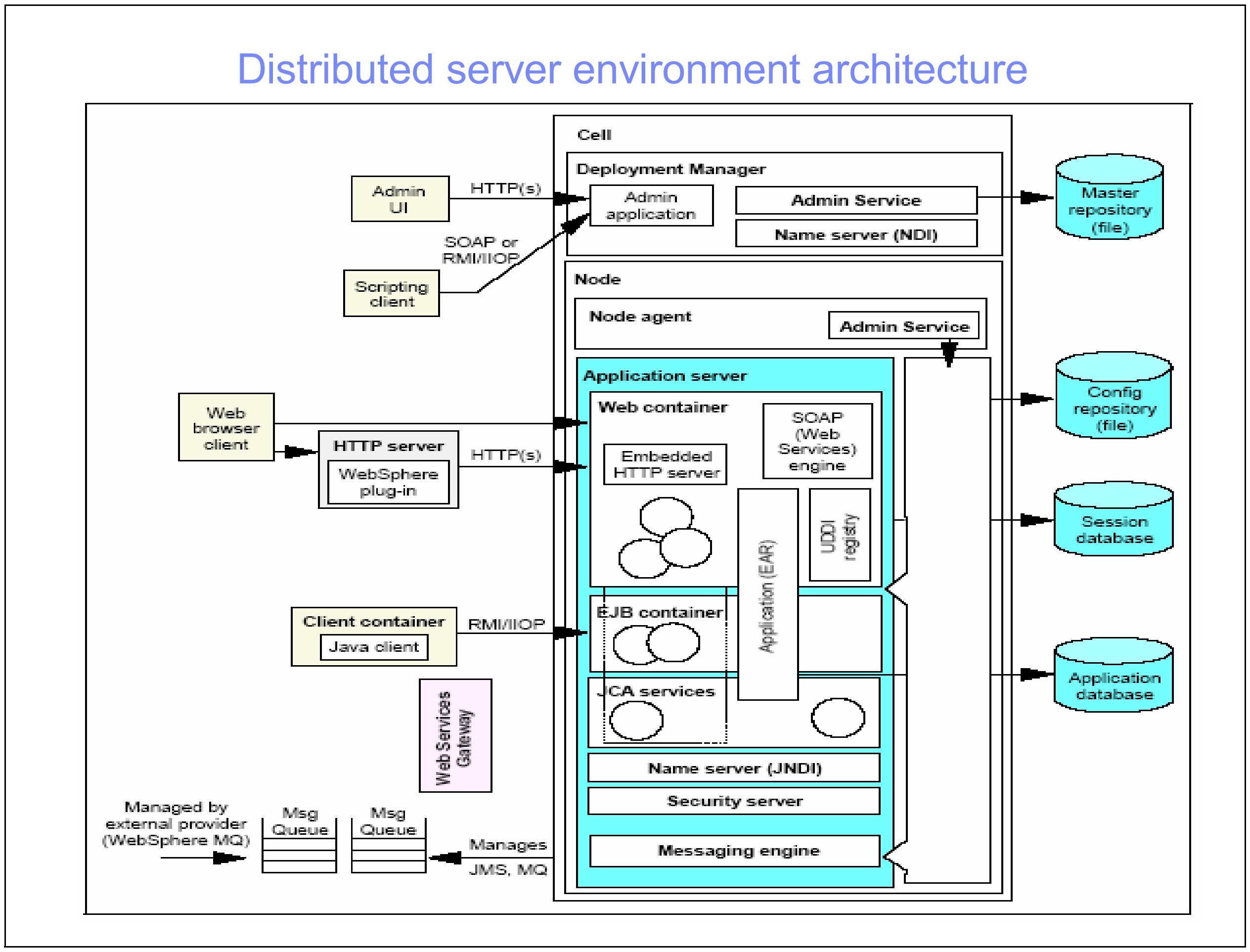


Figure 12-4 Distributed server environment architecture

Cells, nodes, and servers

Regardless of the configuration, the WebSphere Application Server is organized based on the concept of cells, nodes, and servers. While all these elements are present in each configuration, cells and nodes do not play an important role until you take advantage of the features that are provided with Network Deployment.

Application servers

The application server is the primary runtime component in all configurations and is where the application actually executes. All WebSphere Application Server configurations can have one or more application servers. In the Express and WebSphere Application Server configurations, each application server functions as a separate entity. There is no workload distribution or common administration among application servers. With Network Deployment, you can build a distributed server environment that consists of multiple application servers that are maintained from a central administration point. In a distributed server environment, application servers can be clustered for workload distribution. In WebSphere Application Server for z/OS, a server may span multiple systems in a Parallel Sysplex®. Also A representation of a server that is localized to a single z/OS operating system in a Parallel Sysplex. There may be more than one instance of a given server in a single z/OS operating system.

Nodes, node groups and node agents

A node is a logical grouping of server processes, managed by WebSphere, that share common configuration and operational control. A node is associated with one physical installation of WebSphere Application Server. In a stand-alone application server configuration, there is only one node. With Network Deployment, you can configure multiple nodes to be managed from one common administration server. In these centralized management configurations, each node has a node agent that works with a deployment manager to manage administration processes.

Node group is a new concept with V6. A node group is a grouping of nodes within a cell that have similar capabilities. It validates that the node is capable of performing certain functions before allowing them. For example, a cluster cannot contain both z/OS and non-z/OS nodes. In this case, multiple node groups would be defined, one for the z/OS nodes and one for the non-z/OS nodes. A DefaultNodeGroup is created automatically and is based on the deployment manager platform. For WebSphere Application Server for z/OS, a set of servers within a Parallel Sysplex sharing a common WebSphere Application Server for z/OS and OS/390 configuration database. Each server hosts a suite of applications and supports its own operational characteristics. There may be more than one WebSphere Application Server for z/OS and OS/390 node in a Parallel Sysplex, but a single z/OS system within the Parallel Sysplex can support only server instances from a single node. This node group contains the deployment manager and any new nodes with the same platform type.

Cells

A cell is a grouping of nodes into a single administrative domain. In the WebSphere Application Server and Express configurations, a cell contains one node. That node can have multiple servers, but the configuration files for each server are stored and maintained individually.

In a distributed server configuration, a cell can consist of multiple nodes, all administered from a single point. The configuration and application files for all nodes in the cell are centralized into a cell master configuration repository. This centralized repository is managed by the deployment manager process and synchronized out to local copies held on each of the nodes.

12.4.2 z/OS base infrastructure

WebSphere Application Server, when configured in Network Deployment mode, can take full advantage of the underlying z/OS Parallel Sysplex and Workload Manager capabilities to provide for fully available and scalable infrastructures. If you are to build a WebSphere Application Server infrastructure that is scalable and highly available, then the underlying z/OS infrastructure and Enterprise Integration Subsystems (EIS) also need to be scalable and highly available. On z/OS this means making full and redundant use of Parallel Sysplex technology. The Parallel Sysplex should be set up with at least two LPARs to provide redundancy for fail-over situations. Each aspect of the Parallel Sysplex architecture should contain redundancy. This includes the Couple DataSets, Coupling Facilities, Coupling Facility structures and Coupling Facility Links.

All subsystems connected to the WebSphere Application Server infrastructure should also be configured with redundancy. This can be achieved by creating redundant subsystem servers on different LPARs or by configuring the subsystems as sysplex-aware. Sysplex-aware subsystems, such as DB2 in Data Sharing Mode, CICS in a CICSplex configuration, and WebSphere MQ using Shared Queues, can take advantage of z/OS advanced workload management concepts, as well as providing availability during fail-over situations.

Figure 12-5 on page 379 shows a simple application server cluster configured over two LPARs in a sysplex. If one of these LPARs were to go down, the cluster could still process requests through the remaining server clone.

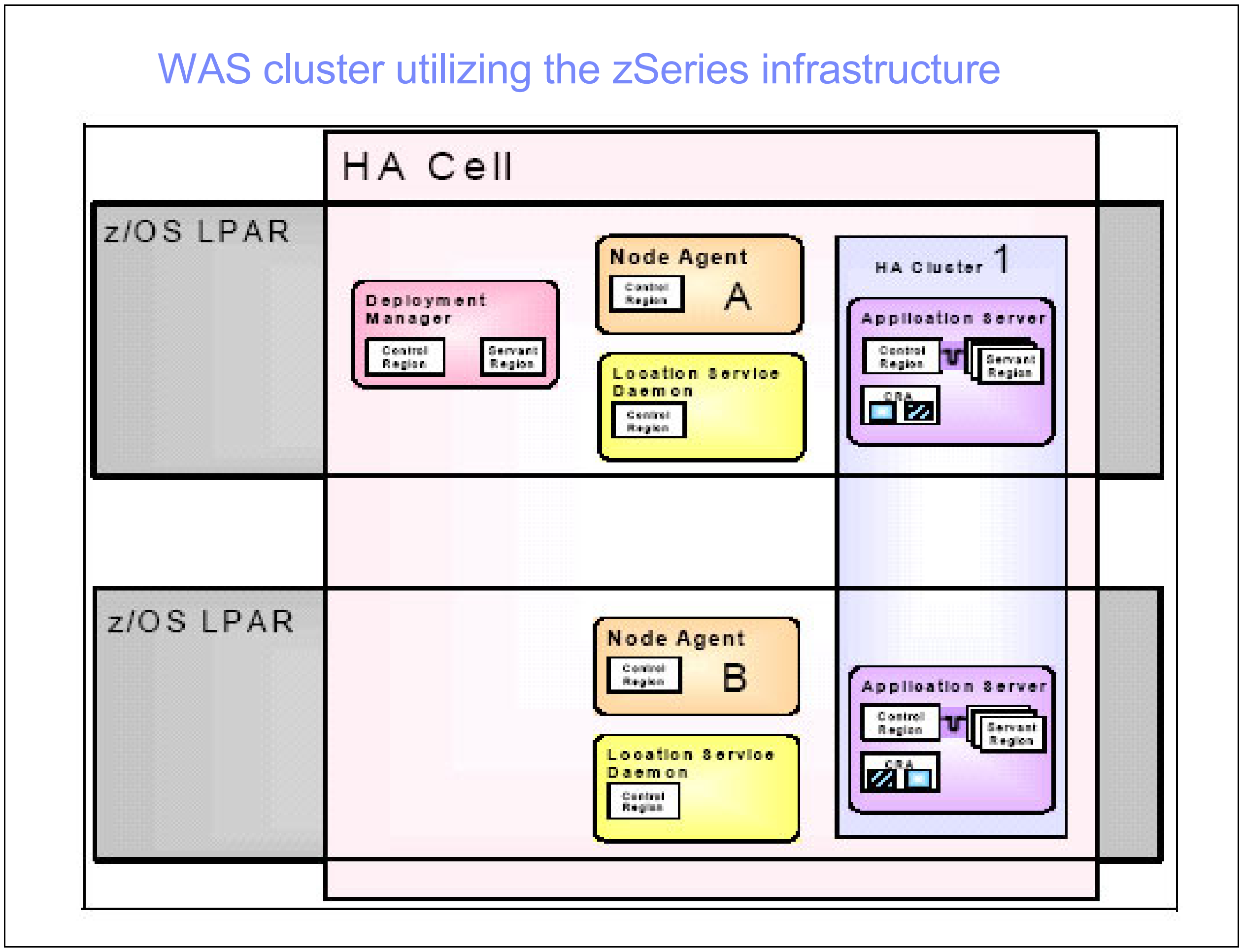


Figure 12-5 WebSphere Application Server cluster utilizing the zSeries infrastructure

12.5 Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. WebSphere Application Server can act as both a Web service provider and as a requester. As a provider, it hosts Web services that are published for use by clients. As a requester, it hosts applications that invoke Web services from other locations.

WebSphere Application Server supports SOAP-based Web service hosting and invocation. Table 12-4 shows details about WebSphere Application Server Web service support.

Table 12-4 WebSphere Application Server Web service support

Web service components	WebSphere Application Server Express and WebSphere Application Server	WebSphere Application Server Network Deployment
Web services support	Yes	Yes

Web service components	WebSphere Application Server Express and WebSphere Application Server	WebSphere Application Server Network Deployment
Private UDDI v3 Registry	Yes	Yes
Web Services Gateway	No	Yes
Enterprise Web services	Yes	Yes

The Web services support includes the following:

- Web Services Description Language (WSDL), an XML-based description language that provides a way to catalog and describe services. It describes the interface of Web services (parameters and result), the binding (SOAP, EJB), and the implementation location.
- Universal Discovery Description and Integration (UDDI), a global, platform-independent, open framework to enable businesses to discover each other, define their interaction, and share information in a global registry. UDDI support in WebSphere Application Server V6 includes UDDI V3 APIs, some UDDI V1 and V2 APIs, UDDI V3 client for Java, and UDDI4J for compatibility with UDDI V2 registries. It also provides a UDDI V3 Registry, that is integrated in WebSphere Application Server.
- Simple Object Access Protocol (SOAP), a lightweight protocol for exchange of information in a decentralized, distributed environment.
- Extensible Markup Language (XML), which provides a common language for exchanging information.
- JAX-RPC (JSR-101), which provides the core programming model and bindings for developing and deploying Web services on the Java platform. It is a Java API for XML-based RPC and supports JavaBeans and enterprise beans as Web service providers.
- Enterprise Web services (JSR-109), which adds EJBs and XML deployment descriptors to JSR-101.
- WS-Security, the specification that covers a standard set of SOAP extensions that can be used when building secure Web services to provide integrity and confidentiality. It is designed to be open to other security models, including PKI, Kerberos, and SSL. WS-Security provides support for multiple security tokens, multiple signature formats, multiple trust domains, and multiple encryption technologies. It includes security token propagation, message integrity, and message confidentiality. The specification is proposed by IBM, Microsoft, and VeriSign for review and evaluation. In the future, it will replace existing Web services security specifications from IBM and Microsoft including SOAP Security Extensions (SOAP-SEC), Microsoft's WS-Security and WS-License, as well as security token and encryption documents from IBM.
- JAXR, an API that standardizes access to Web services registries from within Java. The current JAXR version, 1.0, defines access to ebXML and UDDI V2 registries. WebSphere Application Server provides JAXR level 0 support, meaning it supports UDDI registries.
- JAXR does not map precisely to UDDI. For a precise API mapping to UDDI V2, IBM provides UDDI4J and IBM Java Client for UDDI v3.
- SAAJ, the SOAP with Attachments API for Java (SAAJ) defines a standard for sending XML documents over the Internet from the Java platform.

12.5.1 Web Services Gateway

The Web Services Gateway bridges the gap between Internet and intranet environments during Web service invocations. The gateway builds upon the Web services Definition Language (WSDL) and the Web Services Invocation Framework (WSIF) for deployment and invocation.

With V6, the Web Services Gateway is fully integrated into the integration service technologies which provides the runtime. The administration is done directly from the WebSphere administrative console.

The primary function of the Web Services Gateway is to map an existing WSDL-defined Web service (target service) to a new service (gateway service) that is offered by the gateway to others. The gateway thus acts as a proxy. Each target service, whether internal or external is available at a service integration bus destination.

Exposing internal Web services to the outside world

Web services hosted internally and made available through the service integration bus are called inbound services. Inbound services are associated with a service destination. Service requests and responses are passed to the service through an endpoint listener and associated inbound port.

From the gateway's point of view, the inbound service is the target service. To expose the target service for outside consumption, the gateway takes the WSDL file for the inbound service and generates a new WSDL file that can be shared with outside requestors. The interface described in the WSDL is exactly the same, but the service endpoint is changed to the gateway, which is now the official endpoint for the service client.

Figure 12-6 diagrams how a Web service is exposed to the outside world.

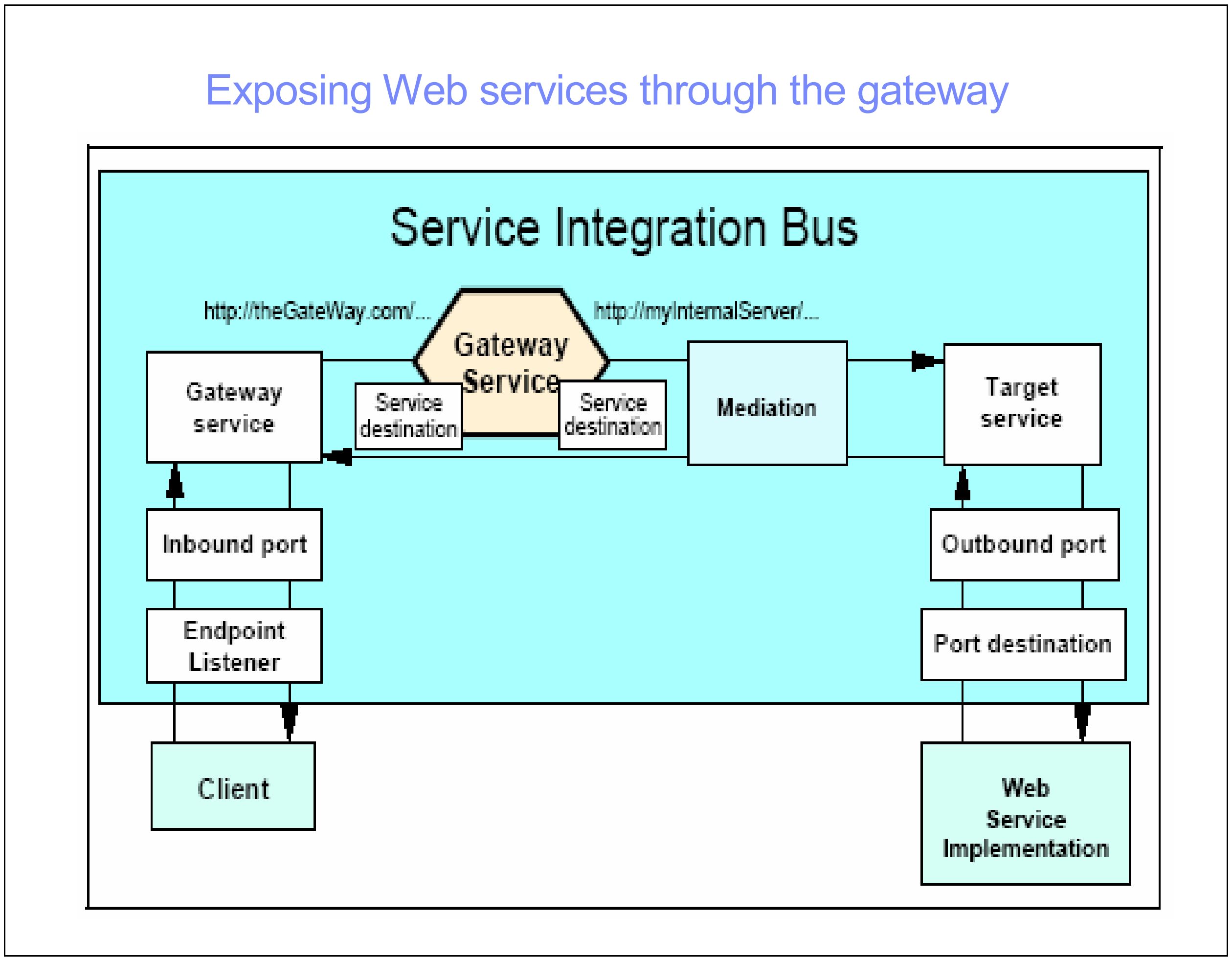


Figure 12-6 Exposing Web services through the gateway

Externally hosted Web services

A Web service that is hosted externally and made available through the service integration bus is called an outbound service. To configure an externally-hosted service for a bus, you first associate it with a service destination, then you configure one or more port destinations (one for each type of binding, for example SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service.

From the gateway's point of view, the outbound service is the target service. Mapping a gateway service to the target service will allow internal service requestors invoke the service as though it were running on the gateway. Again, a new WSDL is generated by the gateway showing the same interface but naming the gateway as service provider rather than the real internal server. All requests to the gateway service are rerouted to the actual implementation specified in the original WSDL.

Of course, every client could access external Web services by traditional means, but if you add the gateway as an additional layer in between, clients do not have to change anything if the service implementor changes. This scenario is very similar to the illustration in Figure 12-6 on page 382, with the difference that the Web service implementation is located at a site on the Internet.

UDDI publication and lookup

The gateway facilitates working with UDDI registries. As you map a service for external consumption using the gateway, you can publish the exported WSDL in the UDDI registry. When the services in the gateway are modified, the UDDI registry is updated with the latest changes.

12.5.2 Service integration bus

The service integration bus provides the communication infrastructure for messaging and service-oriented applications, thus unifying this support into a common component. The service integration bus provides a JMS 1.1 compliant JMS provider for reliable message transport and has the capability of intermediary logic to intelligently adapt message flow in the network. It also supports the attachment of Web services requestors and providers.

The service integration bus capabilities have been fully integrated within WebSphere Application Server, enabling it to take advantage of WebSphere security, administration, performance monitoring, trace capabilities, and problem determination tools.

The service integration bus is often referred to as just a bus. When used to host JMS applications, it is also often referred to as a messaging bus.

Figure 12-7 on page 384 illustrates the service integration bus and how it fits into the larger picture of an Enterprise Service Bus (ESB).

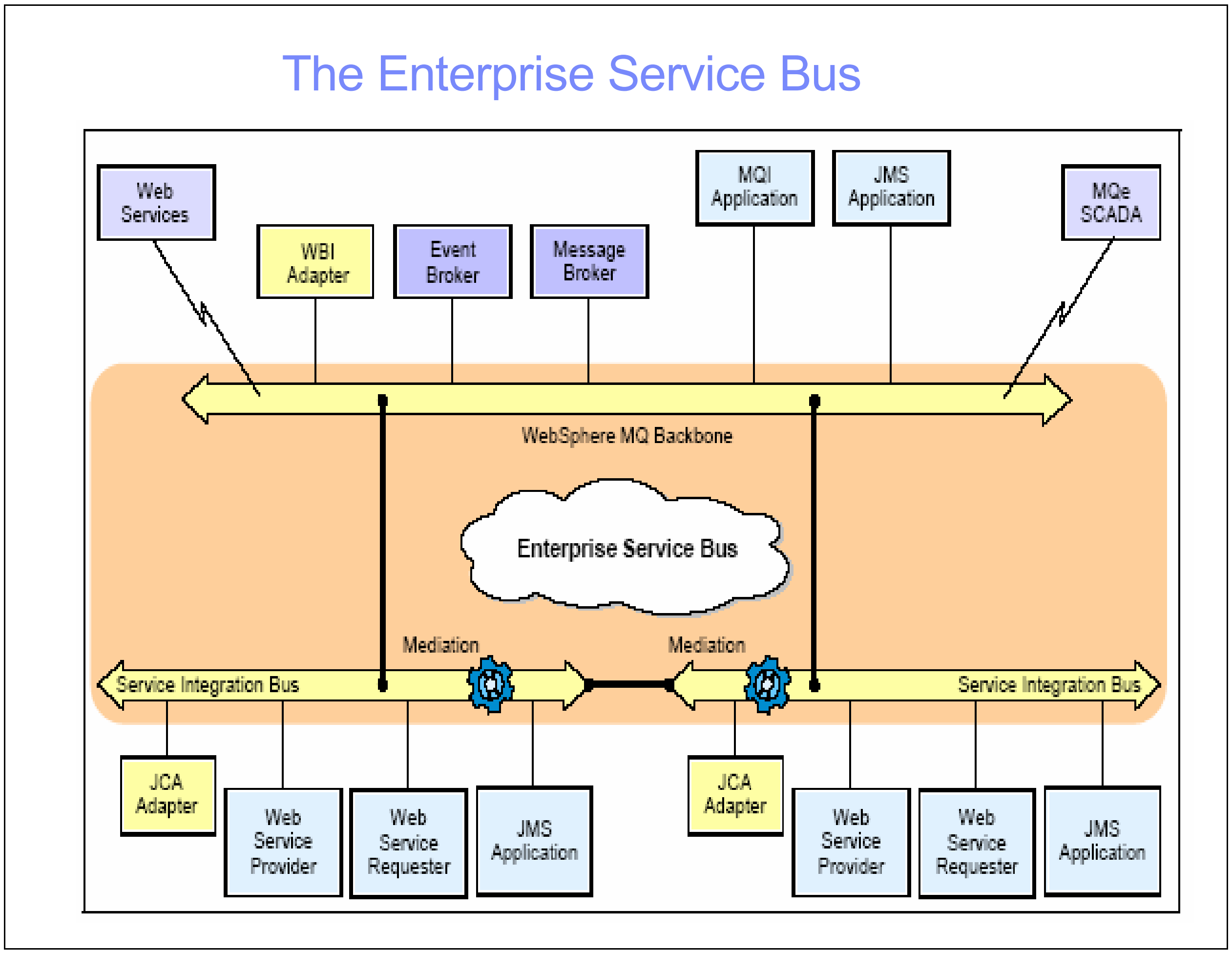


Figure 12-7 Enterprise Service Bus

A service integration bus consists of the following:

- Bus members

  Application servers or clusters that have been added to the bus.

- Messaging engine

  The application server or cluster component that manages bus resources. When a bus member is defined, a messaging engine is automatically created on the application server or cluster. The messaging engine provides a connection point for clients to produce or consume messages from.

  An application server will have one messaging engine per bus it is a member of. A cluster will have at least one messaging engine per bus and can have more. In this case the cluster owns the messaging engine(s) and determines on which application server(s) they will run.

- Destinations

  The place within the bus that applications attach to exchange messages. Destinations can represent Web service endpoints, messaging point-to-point queues, or messaging publish/subscribe topics. Destinations are created on a bus and hosted on a messaging engine.

- Message store

  Each messaging engine uses a set of tables in a data store (JDBC database) to hold information such as messages, subscription information, and transaction states. Messaging engines can share a database, each using its own set of tables. The message store can be backed by any JDBC database supported by WebSphere Application Server.

12.5.3 Summary

In this chapter, we introduced the IBM WebSphere Application Server products. We talked about the products in general, basic configurations and provided an overview of the Web services-specific product features.

We use most of the Web services-specific product functions in the sample applications that we develop in this book.

More information

The WebSphere family of products are regularly updated to cope with a business environment that changes at a high pace. Therefore, the Internet represents one of the best sources of up-to-date information.

For information about the IBM WebSphere family of products, refer to:

`http://www.ibm.com/software/websphere/`

Sources of additional information about particular products can be found at the end of the corresponding chapters.

13

WebSphere Information Server

Information Services are a critical part of an enterprise’s overall service-oriented architecture. In this chapter we introduce IBM WebSphere Information Server which is the product for the functions required to integrate, enrich and deliver information you can trust for key business initiatives. With rich functionality, broad connectivity to heterogeneous sources, and unified metadata, it provides a strong foundation for an enterprise information architecture.

This chapter provides these topics:

- Information as a service
- A closer look at information services
- Introducing the IBM WebSphere Information Server
- WebSphere Information Services Director architecture

© Copyright IBM Corp. 2006. All rights reserved.

13.1 Information as a service

In large organizations it is inevitable that different parts of the enterprise use different information management systems to store and search their critical data. Competition, evolving technology, mergers, acquisitions, departmental initiatives, product capabilities and application requirements all contribute to an unwieldy set of systems, each carrying overlapping information managed using various technologies and data formats. As these information systems become more complex, they become more difficult to change and increasingly expensive to maintain or develop.

SOA principles are being applied to many types of system infrastructure to manage interactions among people, processes and information. In the realm of enterprise information management, SOA principles are being applied to the way information is created and distributed throughout the organization. An Information Service is a type of service that allows the managers of shared information assets, such as customer addresses or product descriptions, a consistent, auditable and secure way to share this asset while maintaining control over how it is used. For the service consumer, the Information Service is a trusted information source provisioned by people who understand the meaning of the data and have responsibility for maintaining it. By separating the interface from the implementation, service providers are free to change how and where the data is produced and managed internally.

13.2 A closer look at information services

Since information is at the core of just about everything in business processing, how do you tell an *information service* from a *process service* that uses data or a *partner service* that exchanges data within an external application? While many of these services involve providing data to the applications, many of the services discussed in the previous section are themselves consumers of information services. Information services focus on the unified management of data from diverse data sources.

Invoking an information service is one of many ways of programmatically accessing data. It is not appropriate for all types of data access. A typical Information Service will have one or more the following characteristics:

- Require pre-processing to prepare the data – The difference between data and information is that information is data that has been prepared for a specific purpose. Information in an On Demand world must be assembled from preprocessed ingredients. Customers need to assemble key Information Services from data that has been prepared for that purpose. Data preparation might include aggregation, synchronization, standardization, de duplication, reformatting, conversion, and so on. These combinations of data preparation tasks form data integration patterns. Choosing the right pattern depends on the amounts of data, where the data came from, when it needs to be delivered and in what form.
- Integrate multiple data sources – Many business processes require information that results from processing large sets of data, often from multiple sources. For example, if you purchase an item online from an e-business, you notice that every interaction you have with the site carries the information about what you have bought, where you live, what you have looked at in the past, things you are likely to want in the future. While a business transaction such as a single purchase might seem a simple database transaction, the context for that transaction is much broader. Bringing together data to provide this context can be a data intensive process that calls for specialized data management tools.
- Provide virtualized views –Virtualization provides transparency that masks the differences, idiosyncrasies and implementations of underlying data sources from users. Ideally, it

allows data from multiple heterogeneous sources to appear to the user as a single system. The service consumer does not have to be aware of where the data is stored (location transparency), what language or programming interface is supported by the data source (invocation transparency), if SQL is used, what dialect of SQL the source supports (dialect transparency), how the data is physically stored, whether it is partitioned and/or replicated (physical data independence, fragmentation and replication transparency) or what networking protocols are used (network transparency). The user should see a single uniform interface, complete with a single set of error codes (error code transparency).

- Address a repeatable need – One of the chief tenets of service orientation is reusability. To be successful, Information Services must be at the right level of granularity to encapsulate an information need that is repeatable. An information service should be designed such that it is not so specific that it is only used in a certain circumstance or so general that the result set requires significant additional programming by the service consumer to be useful.

Most of the capabilities of an information service can be developed by integrating multiple point products and developing custom code. The key advantage of the IBM Information Server is that it provides these capabilities through point and click assembly rather than coding. It allows users to write applications as though all of the data were in a single database, when, in fact, the data may be stored in a heterogeneous collection of data sources. It also eliminates the need to bind application logic to data implementation that is likely to change with organizational growth/acquisition and/or new business challenges.

13.3 Introducing the IBM WebSphere Information Server

IBM WebSphere Information Server is software offering designed to integrate, enrich and deliver information for key business initiatives. With rich functionality, broad connectivity to heterogeneous sources, and unified metadata, it provides a strong foundation for an enterprise information architecture. By providing a SOA interface to these capabilities the Information Server gives users real-time, access to integrated business information across and beyond the enterprise by publishing reusable services that represent valuable business information. As shown in Figure 1, the Information Server leverages many data-centric operations to prepare the data and deliver it as an Information Service. These operations include the ability to:

1. **CONNECT** to any data or content, wherever it resides.It provides direct, native access to relevant information sources
2. **UNDERSTAND** and analyze information, including its meanings, relationships and lineage.
3. **CLEANSE** information to ensure its quality and consistency.
4. **TRANSFORM** information to provide enrichment and tailoring for its specific purposes.
5. **FEDERATE** information to provide a unified view to people, processes, and applications.

Figure 13-1 on page 390 illustrates the steps for turning the data into information:

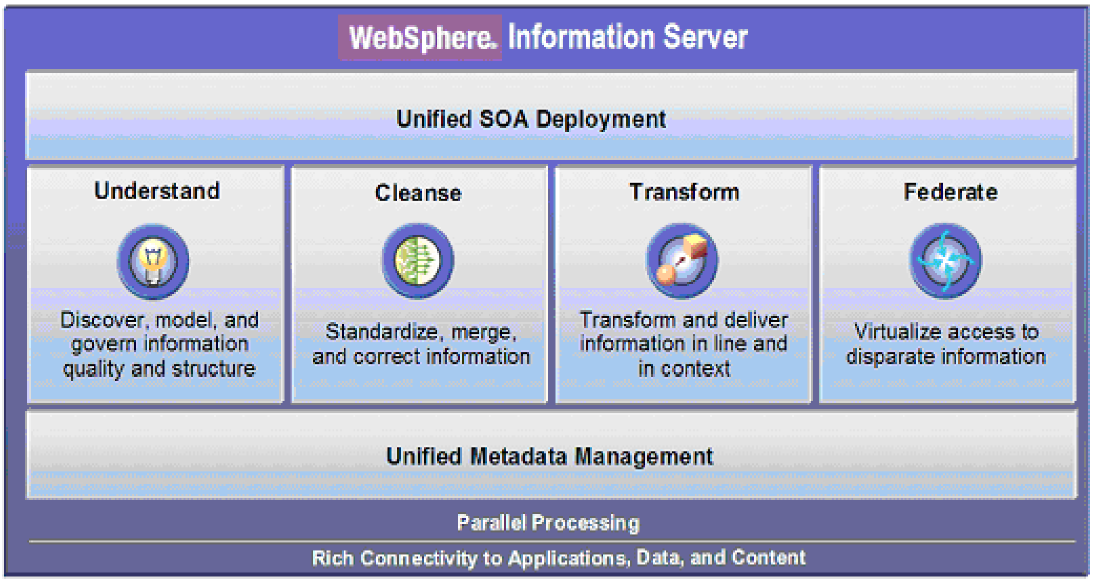

WebSphere. Information Server
Unified SOA Deployment
Understand
Discover, model, and govern information quality and structure
Cleanse
Standardize, merge, and correct information
Transform
Transform and deliver information in line and in context
Federate
Virtualize access to disparate information
Unified Metadata Management
Parallel Processing
Rich Connectivity to Applications, Data, and Content

Figure 13-1 IBM WebSphere Information Server turns data into information

13.3.1 Unified SOA deployment

The IBM Information Server delivers information services using an SOA framework called the IBM WebSphere Information Services Director (WISD). This product module provides a seamless SOA integration platform allowing organizations to unlock information directly from multiple disparate data and content sources. The WISD provides the following capabilities:

- **Facilitates re-use** - By ensuring consistent definitions, packaging, and rules applied to the data, the WebSphere Information Services Director, provides information services that can be reused easily across processes, and maintained independently to make the business more flexible.
- **Productivity aligned with how businesses are structured** – The information services are maintained by the people who know and understand the information best, allowing the process developers to focus their efforts, knowing they have the best available information.
- **Facilitates governance and control** – By centralizing control and management of information services within the WebSphere Information Services Director, the WebSphere Information Server's integrated service monitoring and linked metadata provide strong control, visibility, and traceability of how information is being utilized across the organization.
- **Open and Standards-based** – The WebSphere Information Services Director is deployed on a J2EE-based foundation framework that provides flexible, distributable and configurable interconnections among the many parts of the architecture through accepted SOA standards. It provides the unique ability to publish services using the following bindings:
  - Web services – Any XML Web Service-compliant application can invoke an Information Service as a Web service.
  - SOAP over JMS – In a messaging environment, the Information Services Director can automatically generate an asynchronous JMS queue listener (message-driven bean) and route incoming messages into Information Services.
  - EJB – For Java-centric development, the Information Services Director can generate a J2EE-compliant EJB (stateless session bean) where each Information Service is instantiated as a separate synchronous EJB method call.

- Infrastructure services delivered with the IBM WebSphere Information Services Director include:
  - Integrated Metadata Services
  - Service catalog
  - Logging services
  - Security services
  - Load balancing and availability services
  - Reporting services
  - Installation and configuration

13.4 WebSphere Information Services Director architecture

The WebSphere Information Services Director (WISD) is the critical product module that allows a user to service-enable information management tasks created within the WebSphere Information Server. Its extensible architecture allows the WISD to service-enable DB2 and other products not strictly part of the Information Server product offering. In this section we shall review this architecture.

13.4.1 Design concepts

Rather than hand-coding each service, WISD's powerful point and click graphical interface enables rapid deployment of information services. To better understand how the Information Services are created you must first understand how Information Services are organized. Figure 13-2 shows the object hierarchy that consists of Projects, Applications, Services and Operations which define how services are designed and deployed.

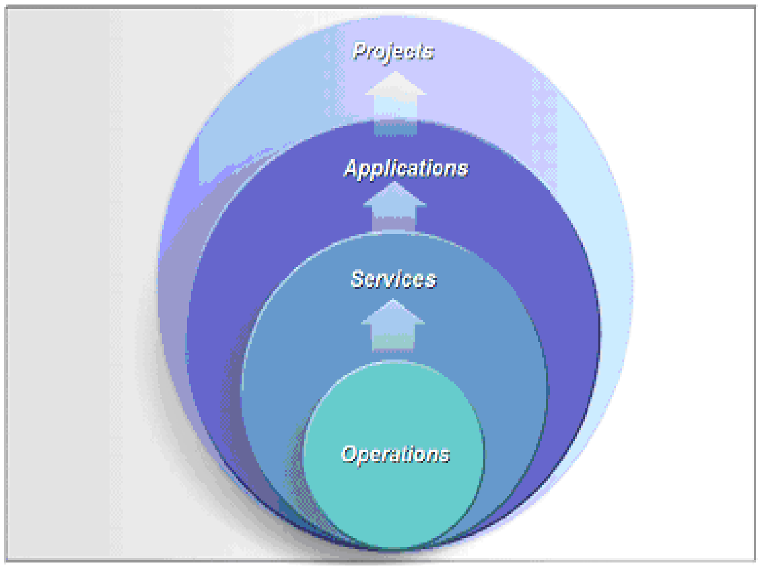


Figure 13-2 Information Services are organized into hierarchies

The following core objects describe how Information Services are organized:

- **Operations** – An operation is a unit of work that is done on behalf of the Information Service. For example, invoking a federated query using the Federation Server or an ETL job using DataStage is an operation. The designer needs to link an operation, to a particular behavior (an implementation). This implementation is delegated to one of the operation providers available (DataStage, SQL Federation, and so on).

- **Services** – A service is what an external user will see after deployment. A Web Service Description Language (WSDL) document is generated for each service (when using the Web Service binding) that is part of a deployed application and the WSDL documents are available through the service catalog.
- **Applications (Service Groups)** – An application object provides a way of grouping information services for deployment. It allows all the services to be deployed together and can be managed as a unit. For example, an application object might have 100 services that are commonly used together. An administrator can start or stop all of the services contained within the application definition. The design environment provides some information about applications to the user (is it deployed, has it been changed since deployment) and also allows users to perform some level of validation.
- **Projects** - A Project is an organization and collaboration artifact. It exists only in the design environment (not at runtime). This is the place where users and permissions are defined for all the applications created in the project. A project has a dashboard that contains important information about it. A project can be exported (to an XML document) and imported to another design environment. When importing/exporting, a subset of the entire project can be specified.

13.4.2 Product architecture

Figure 13-3 shows the components that make up the WISD.

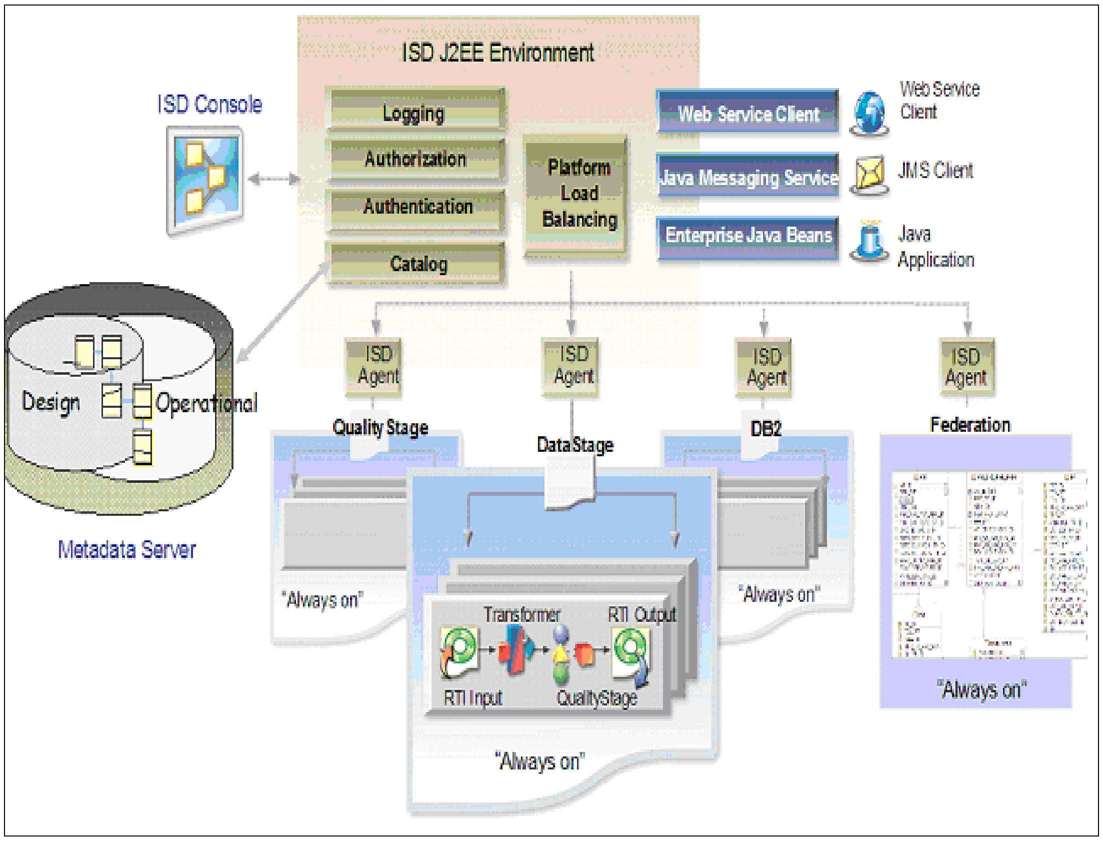


Figure 13-3 WebSphere Information Services Director Architecture

These components include:

- Infrastructure Services – The WISD is deployed on a J2EE-based Service Backbone that provides flexible, distributable and configurable interconnections among the many parts of the architecture.   These infrastructure services include:
  - **Logging Services** – Provide a central place for a user to record service events. Logs go into the common repository with each service provider defining relevant logging categories for that service. Configurations determine what categories of logging messages are actually saved in the repository.
  - **Security Services** – Support role-based authentication of users, access-control services and encryption appropriate for compliance with many privacy and security regulations.
  - **Service Catalog** – Provides users with the means search and browse services by category and to view descriptions available to be defined by the WISD.
  - **Load balancing and availability services** – Support routing requests to multiple servers to provide optimal loading and a high availability environment that can recover from any individual server failure.
- **Operation Provider Handlers** – As described in the object model, each service is performed by defining operations performed by an Operations Provider. As shown in Figure 13-3 on page 392, ISD agents contain handlers to process service requests from the following operations providers:
  - **WebSphere DataStage** – Transforms data of any complexity and delivers it to target applications. WebSphere DataStage provides built-in connectivity for easy access to any source or target system, advanced development tools to define and deploy transformation integration processes and a scalable platform to process massive volumes of data.
  - **WebSphere DataStage TX** – Extends the data transformation capabilities of WebSphere DataStage by providing data synchronization of complex, hierarchical data formats across transactional and operational environments. This functionality is essential in many industries that rely on specific document formats such as EDI, HL7 for health care and the financial services SWIFT formats for conducting business with customers, suppliers and partners across the global business environment.
  - **WebSphere QualityStage** – Prepares your data for integration by providing a powerful framework for developing and deploying data matching, standardization, enrichment and survival operations, simplifying the process of integrating similar data from multiple sources.
  - **IBM** DB2 UDB – Provides a native interface to IBM's flagship relational database system for development and deployment of critical enterprise data.
  - **WebSphere Federation Server** – The Federation server presents a single virtual view of the data that may exist in many forms: structured and unstructured; mainframe and distributed; public and private. This data may reside in diverse source systems (such as Oracle databases, enterprise applications, Microsoft spreadsheets, flat files, the Web, news groups, and more) and be distributed across a variety of operating environments (such as Windows, Linux, UNIX and z/OS)
- **Service Bindings** – Service consumers are able to access Information Services using multiple technologies for program interoperability (bindings). The WISD allows the same service to support multiple protocol bindings, all defined within the WSDL file. This improves the utility of services and therefore increases the likelihood of reuse and adoption across the enterprise. The WISD provides the unique ability to publish the same service using the following bindings including:

  - **SOAP over HTTP (Web services)** – Any XML Web service–compliant application can invoke a Information Service as a Web service. These Web services support the generation of “literal document-style” and “SOAP-encoded RPC-style” Web services.
  - **SOAP over** JMS – In a message queue environment, the Information Server can automatically generate an asynchronous JMS queue listener (message-driven bean) and route incoming SOAP messages into Information Services. As an option, it can adapt the output of an Information Service into a SOAP message that can be posted to one or more JMS queues or topics.
  - EJB – For Java-centric development, the Information Server can generate a J2EE-compliant EJB (stateless session bean) where each Information Service is instantiated as a separate synchronous EJB method call.
  - **Service Component Architecture (SCA)** – This future binding provides a client programming model and consistent way of describing components as services available over different protocols. SCA is supported by IBM Enterprise Service Bus product.

13.4.3 Conclusion

IBM WebSphere Information Server delivers all of the functions required to integrate, enrich and deliver information you can trust for key business initiatives. With rich functionality, broad connectivity to heterogeneous sources, and unified metadata, it provides a strong foundation for an enterprise information architecture.

Information Services are a critical part of an enterprise’s overall service-oriented architecture. The WebSphere Information Services Director is the component of the WebSphere Information Server that enables Web services. Users derive three main business benefits from the WebSphere Information Server:

- **Reducing cost** – By reproposing existing data, every request to see data in a different format or combined in a different way does not require a new database. Furthermore, applications are more resilient to change because structure, location or size of the databases and data files can change without impacting production applications.
- **Saving time** – Reusable services can be created that transparently process data from multiple sources in the background. By using preexisting components, time is saved due to assembly rather than programming.
- **Controlling access** – An SOA based framework for Information Services allows users to factor complex processing into single shared information services that improve accessibility while ensuring consistency in the way the information is managed.

With the capabilities of the WebSphere Information Server, On Demand businesses can leverage the power of SOA to deliver Information as a Service.

For more information about Information Services and the IBM WebSphere Information Server refer to:

http://www-306.ibm.com/software/data/integration/info_server/

Part 5

Assembling and developing a scenario

In Part 5 we show how to implement an SOA scenario involving the data servers. We start from the business case and show how to quickly develop SOA access services.

This part includes the following chapters:

We used these IBM tools to develop the RAD/WID/Developers Workbench:

- Provider
  - Develop Web services using Data Access Objects
  - Develop DADx/WORF using RAD
- Consumer
  - Develop Portlets as the SOA user interface
  - SOAP UDF/SOAP SP as consumer
- Developing information services
  - Information Server

© Copyright IBM Corp. 2006. All rights reserved.

SOA scenario

This chapter describes a typical problem and strategy space for a data server enterprise customer and illustrates the value of an incremental SOA solution using a real scenario.

This chapter provides these topics:

- Problem space
- Strategy space
- Solution space usage scenario

The implementation of most of the scenario is described in Chapter 15, “Developing SOA access services” on page 401.

© Copyright IBM Corp. 2006. All rights reserved.

14.1 Problem space

ITSO bank (however, a fictional name) was established in the 1970's as a savings bank. It provided checking accounts, saving accounts and mortgage accounts. Over time, the account data was moved to DB2 for z/OS to make it more accessible for reporting purposes. However, the customer record that contains address information still resides in an IMS database. There exists a COBOL/DLI program that, given a customer number, can return all billing and mailing addresses.

For the core banking application there existed COBOL/SQL programs to list all the account IDs for a customer number, get the account details for an account ID and create an account for a customer number. This account can either be a checking, savings or mortgage account. In order to make these accessible to remote programs, these COBOL programs have been converted into three stored procedures.

During the 1990's the bank was trying to diversify its portfolio and bought a smaller bank that was focussing on personal lines of credits, issuing credit cards. This company uses Informix as a database and 4GL applications to manage credit card accounts. There exist 4GL applications to list accounts (credit card numbers) for a customer number, get detailed account info for a credit card number. The customer record that existed in Informix was merged into the IMS-based customer record.

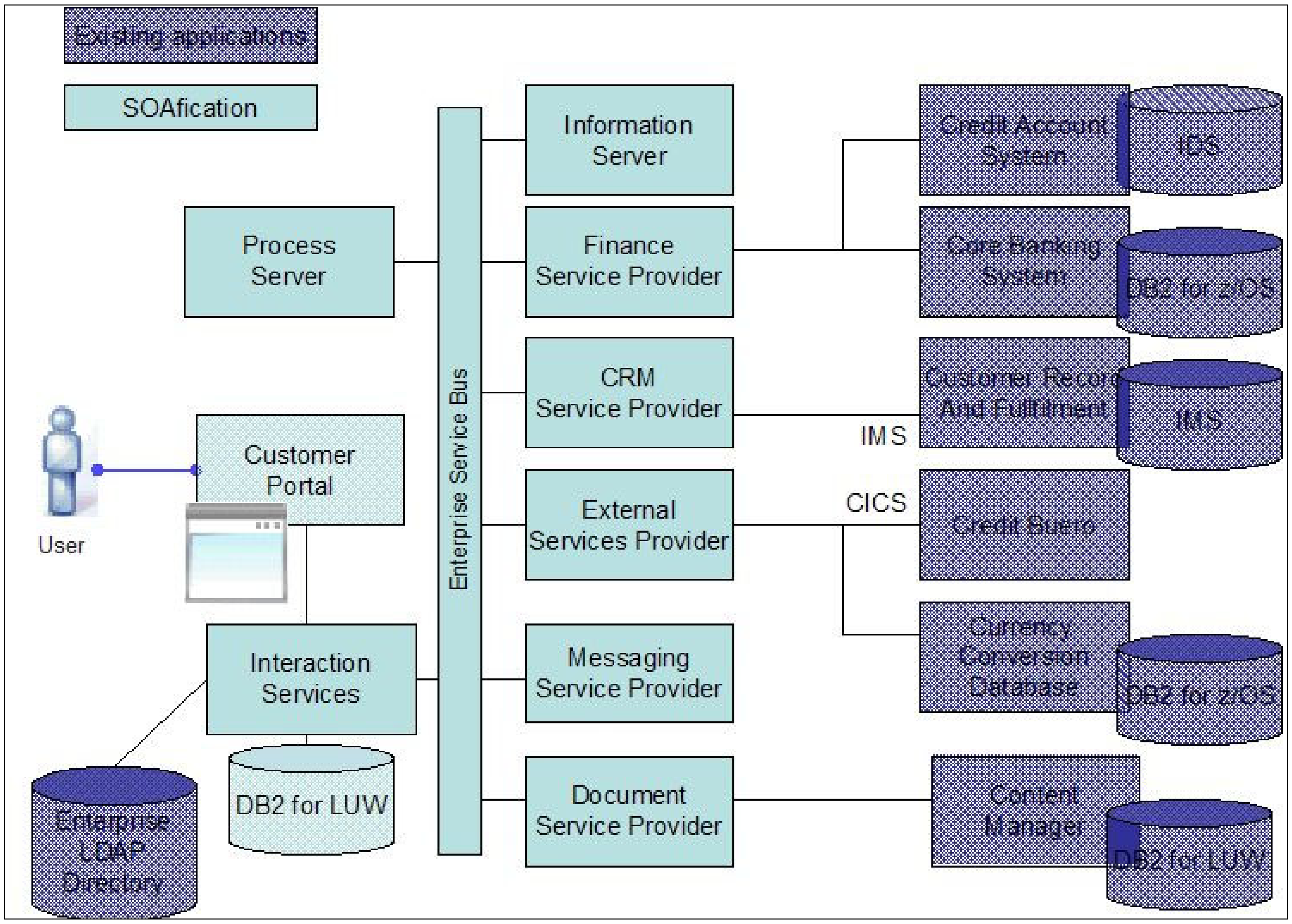


Figure 14-1 Project ISOA

The bank interacts with two external services. It can check FICO scores (a number from 450 to 850) for a social security number associated with a customer number through calling a

CICS transaction. It get's billed for that use, so the bank wants to minimize using that service. It also receives currency exchange rates into a table through a proprietary interface and has written a simple SQL stored procedure that accepts the amount, currency, date and target currency as a parameter and has an output parameter for the amount in the target currency.

Very recently, it has implemented DB2 Content Manager from an IBM partner, where all incoming mail, returned checks, and so on is scanned and associated with a customer number. In the branch offices, the Content Manager portlets are used to view documents for a customer. Security information for bank employees (user ID and credentials) is stored in a central LDAP directory which is used with Content Manager.

14.2 Strategy space

The ITSO Bank has heard about SOA and wants to understand how it can use SOA to tie everything together and get new value out of it's existing services without having to re-write, re-invent everything that is already working well. They embark on a project internally called ITSOA that will provide mortgage banking services to their customers on the internet. Their strategy is to make their existing functions available as services, so that individual customers on the Web can use it, but also so that mortgage brokers can programmatically access it.

Their CTO recommends to implement HTTP-based Web services that use SSL as the transport layer and basic authentication. The credentials to use services are stored in the LDAP directory (service userids). To achieve highest interoperability, the CTO recommend a document messaging style using literal serialization mechanism. Once a dominant standard for message-level Web service security has emerged, the CTO will look into adopting that to have security flow with the Web service through the Enterprise service bus eventually.

14.3 Solution space usage scenario

1. Customer logs on to the customer portal application of his bank using his customer number and a password.

   The self-banking application has two portlets. One portlet that shows all accounts (checking, saving, mortgage, credit cards and their details) as well as the current FICO score. The second portlet makes personalized offers to the customer.

2. The FICO is stored in the Viper database along with a time stamp when it was last updated. If the update was greater than 1 month, the credit bureau Web service (see additional material) is called for that customers SSID which is retrieved from the customer record Web service method that retrieves the customer details.

3. While the customer is looking at his accounts, a Web service call with the customer number is issued asynchronously to the information server to gather mortgage offerings for the customer and show them in the offering portlet. IBM Information server is used to gather, clean, consolidate offering data from multiple sources within the bank infrastructure (including the credit account, core banking system as well as the FICO score in the Viper database). The information server then returns a number of mortgage account offerings. It also accesses the IMS address data to determine where the person lives and offers a number of mortgages in and around the median house price for the area code.

   ```
   $400 000       5%
   $600 000       6%
   ```

4. The customer is interested and clicks on one link. A form is displayed and a Web service call from the CRM systems provider goes to the IMS customer record of the bank to retrieve address information and prefills that form. If the CRM system is unavailable, the address data is used from the local database table, if it is available, the local table is refreshed.
5. The customer reviews and then clicks on the Apply button. The finance service provider Web service is called to create a new account.

   Also an approved mortgage approval form is added to content manager and the customer gets a message on the portlet saying `Your mortgage application has been approved, we will mail you further information shortly`.
6. On the regular Bank Web-site, the ITSO bank also wants to provide a currency conversion calculator because the travel season is starting. Since currency conversion is available through a stored procedure on z/OS, they ask one of the PHP savvy interns to prototype a solution for them.

What have we demonstrated:

- IMS, CICS, DB2 for z/OS, IDS access services
- Content Manager client
- Portlets that call Web services and access DB2 Viper data, Viper SOAP WS consumer.

15 Developing SOA access services

This chapter contains a number of sample scenarios which demonstrate the SOA capabilities of the IBM data servers (DB2 for z/OS, DB2 for Linux, UNIX and Windows, Informix Dynamic Server). We cover different aspects of SOA, like exposing data and database functions as Web Services, or consuming Web Services within data servers. But we also include sample scenarios which put the different pieces together, and provide a single Web portal interface aggregating the different SOA components and services. We use the IBM Rational Application Developer 6.0 to implement the sample scenarios because of its features it relieves the you from performing many tedious configuration and setup tasks, and allows you concentrate on the implementation of the actual solution.

In this chapter, you will develop the following sample scenarios:

- Scenario exposing DB2 business logic as Web Services
- Scenario using DB2 as Web Service consumer
- Scenario exposing I4GL business logic as Web services
- “Scenario aggregating services as portlets”

15.1 Scenario exposing DB2 business logic as Web Services

This scenario illustrates the usage of DB2 technologies to easily expose business logic, which is contained in stored procedures executed in a DB2 database as Web Services.

15.1.1 Overview

The ITSO Bank company stores all of its accounts data in a DB2 for z/OS database. Most programs and jobs operating on this database are written in COBOL. To allow remote programs to access the database, shared COBOL business logic has been encapsulated into stored procedures.

To integrate this DB2 system into our new SOA environment, the business logic needs to be exposed as Web Services. The usage of stored procedures as interface to the business logic is already a first step towards a service-oriented architecture. The most cost-effective way to enable the system for SOA is to create a one-to-one mapping of existing stored procedures to Web Services as shown in Figure 15-1 (allow access to each stored procedure using a single Web Service operation).

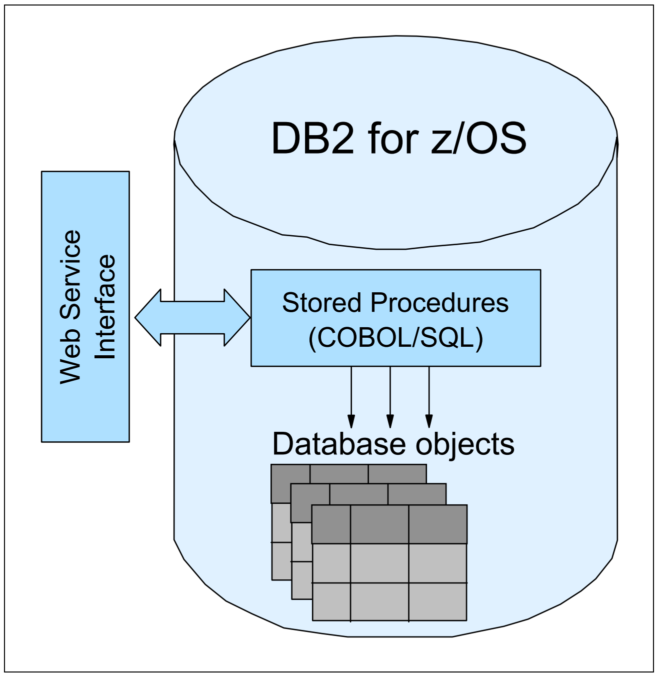


Figure 15-1 ITSO Bank account management with DB2 for z/OS

We can follow a number of different approaches to expose the existing stored procedures. In this scenario we focus on two representative approaches, and compare the advantages and disadvantages of each of them:

- Show how to create a WORF DADX group which allows configurable access to stored procedures.
- Create a Java bean-based Web Service that calls the DB2 stored procedures through JDBC.

Both approaches have in common that they need to run within a J2EE environment (the first one because WORF is based on a Java implementation of SOAP, and the second one because it is a native J2EE application). We use the IBM Rational Application Developer to create the Web Services. This tool provides wizard-based support which speeds up application development.

Use Cases

The DB2 for z/OS database containing the accounting data of the ITSO Bank company is named DB2ACCTD. It offers a complete catalog of stored procedures. Showing how to expose the complete catalog as services is beyond the scope of our book, so we're going to perform this action only for a few selected procedures which are part of our overall scenario. These procedures are as follows:

LSTACNBR	Returns a list of all accounts of a customer (providing an existing customer number).
GETMRACD	Returns the attributes (mortgage amount, interest rate, payments balance, and so on) of a given mortgage account.
CRTMRACT	Creates a new mortgage account for an existing customer.

The signatures of these stored procedures are shown in Example 15-1.

Example 15-1 Mortgage account stored procedure signatures

```
-- SP:         LSTACNBR
-- Purpose:    Returns a list of account numbers for a given customer number.
-- Parameters: IN  CUSTNUM      Customer number
--             OUT PSQLCODE     Returned SQL code
--             OUT PSQLSTATE    Returned SQL state
--             OUT PSQLERRMC    Returned error message
-- Result set: ACCTNUM  CHAR(10) Account number
--             ACCTTYPE CHAR(1)  Account type: S=Savings, C=Credit, M=Mortgage
CREATE PROCEDURE ACCTDB.LSTACNBR
(
 IN  CUSTNUM      CHAR(10)
,OUT PSQLCODE     INTEGER
,OUT PSQLSTATE    CHAR(5)
,OUT PSQLERRMC    VARCHAR(250)
)
DYNAMIC RESULT SETS 1
EXTERNAL NAME LSTACNBR
LANGUAGE COBOL
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
NO DBINFO
WLM ENVIRONMENT DB2ACCTD
;

-- SP:         GETMRACT
-- Purpose:    Returns the attribute of a mortgage account.
-- Parameters: IN  ACCTNUM      Account number
--             OUT PSQLCODE     Returned SQL code
--             OUT PSQLSTATE    Returned SQL state
--             OUT PSQLERRMC    Returned error message
-- Result set: CUSTNUM  CHAR(10) Customer number
--             AMOUNT   DEC(10,2)Mortgage amount
--             INTRATE  DEC(4,2) Interest rate
--             LIFETIME DEC(3,0) Mortgage lifetime
--             OPENDATE DATE     Open date of mortgage
--              PAYMENT  DEC(10,2)Monthly mortgage payment
--              BALANCE  DEC(10,2)Accumulated payments balance
CREATE PROCEDURE ACCTDB.GETMRACT
(
 IN  ACCTNUM       CHAR(10)
,OUT PSQLCODE      INTEGER
,OUT PSQLSTATE     CHAR(5)
,OUT PSQLERRMC     VARCHAR(250)
)
DYNAMIC RESULT SETS 1
EXTERNAL NAME GETMRACT
LANGUAGE COBOL
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
NO DBINFO
WLM ENVIRONMENT DB2ACCTD
;

-- SP:          CRTMRACT
-- Purpose:     Creates a new mortgage account with the given attributes, and
--              returns an account number.
-- Parameters:  IN  CUSTNUM  CHAR(10) Customer number
--              IN  AMOUNT   DECIMAL  Mortgage amount
--              IN  INTRATE  DECIMAL  Interest rate
--              IN  LIFETIME DECIMAL  Mortgage lifetime
--              IN  PAYMENT  DECIMAL  Monthly mortgage payment
--              OUT PSQLCODE      Returned SQL code
--              OUT PSQLSTATE     Returned SQL state
--              OUT PSQLERRMC     Returned error message
-- Result set:  ACCTNUM  CHAR(10) (New) account number
CREATE PROCEDURE ACCTDB.CRTMRACT
(
 IN  CUSTNUM       CHAR(10)
,IN  AMOUNT        DECIMAL(10,2)
,IN  INTRATE       DECIMAL(4,2)
,IN  LIFETIME      DECIMAL(3,0)
,IN  PAYMENT       DECIMAL(10,2)
,OUT PSQLCODE      INTEGER
,OUT PSQLSTATE     CHAR(5)
,OUT PSQLERRMC     VARCHAR(250)
)
DYNAMIC RESULT SETS 1
EXTERNAL NAME GETMRACT
LANGUAGE COBOL
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
NO DBINFO
WLM ENVIRONMENT DB2ACCTD
;
```

15.1.2 Implementation of the Web Services using WORF

The DB2 Web Services Object Runtime Framework (WORF) has already been introduced in 6.3, “Web services object runtime framework (WORF)” on page 142 for DB2 for z/OS and in

7.2, “Web services provider” on page 170 for DB2 for Linux, UNIX and Windows. The implementation of services and applications based on this framework is identical for both database products. WORF incorporates many capabilities, like creating a WSDL based on SQL statements, or storing and retrieving XML documents. It is especially applicable to our scenario, having business logic defined in existing stored procedures which need to be service-enabled without much development overhead.

WORF architecture

Before we can start going through the steps, we spend a few words about WORF concepts. WORF is a Java Web application which runs in a J2EE container, and allows to expose SQL statements and stored procedures as Web Services without requiring development effort. The Web Services interfaces are defined by XML configuration files. The building blocks of WORF are:

- A DADX file
- A DADX group configuration
- A J2EE Web application

Figure 15-2 shows the logical structure of a WORF-based application, positioning the building blocks (it lists three DADX groups exemplarily, but actually any number of DADX groups can be included within one application).

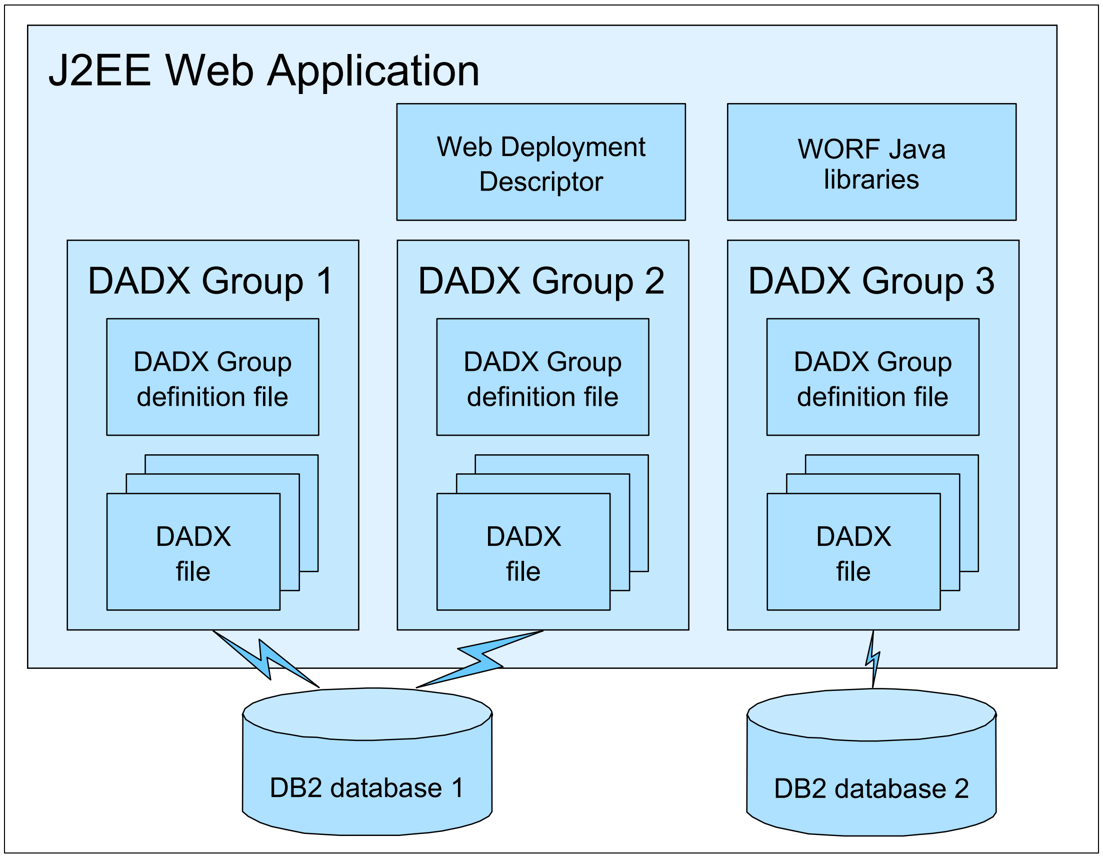


Figure 15-2 Example of the logical structure of a WORF-based J2EE Web application

DADX file

The concept and structure of a DADX file is explained in 6.3, “Web services object runtime framework (WORF)” on page 142 and in 7.2, “Web services provider” on page 170. A DADX file maps one or multiple SQL statements or stored procedures to Web Services operations, that is, each SQL statement and each stored procedure map to exactly one Web Services operation. WORF generates a single WSDL document from each DADX file contained in an application.

DADX group

DADX files are combined into DADX groups, each group containing one or more DADX files. Each DADX group has a file named `group.properties` which defines the properties of the group. The most important property of a DADX group is the database connection information (which means that all DADX files belonging to one group access the same DB2 database, as demonstrated in Figure 15-2 on page 405). Another property specifies if the DADX Web Services of this group use the RPC/encoded or document/literal SOAP encoding style (see "SOAP messaging mechanisms" on page 56).

J2EE Web application

A WORF application is deployed as a J2EE Web application. Such an application contains the WORF runtime environment, one or multiple DADX groups and their corresponding DADX files. WORF supports different SOAP engines (supported engines in the WORF package bundled with DB2 version 8.2 are IBM SOAP and Apache Axis 1.0), and the actual engine used is selected on application level (by setting a parameter in the application's Web deployment descriptor). If you want to enable multiple databases for SOA and you have to use different SOAP runtimes, create one WORF J2EE application for each SOAP runtime.

DB2 for Linux, UNIX and Windows comes with the *Embedded Application Server*, which can also host WORF, but deployment and maintenance features are limited.So in order to use WORF it is often required to use a separate server product. Any Application Server supporting J2EE applications can be used, but it is recommended to use either Apache Tomcat (`http://tomcat.apache.org/`) which is a cost-free open source product, or the IBM WebSphere Application Server (WAS). We use WebSphere Application Server version 6 for our examples.

For the development of the WORF example we use the IBM Rational Application Developer (RAD), since this tool includes wizards for WORF-based applications. In our example we're going to create a J2EE application which is deployed to a WebSphere Application Server in our production environment.

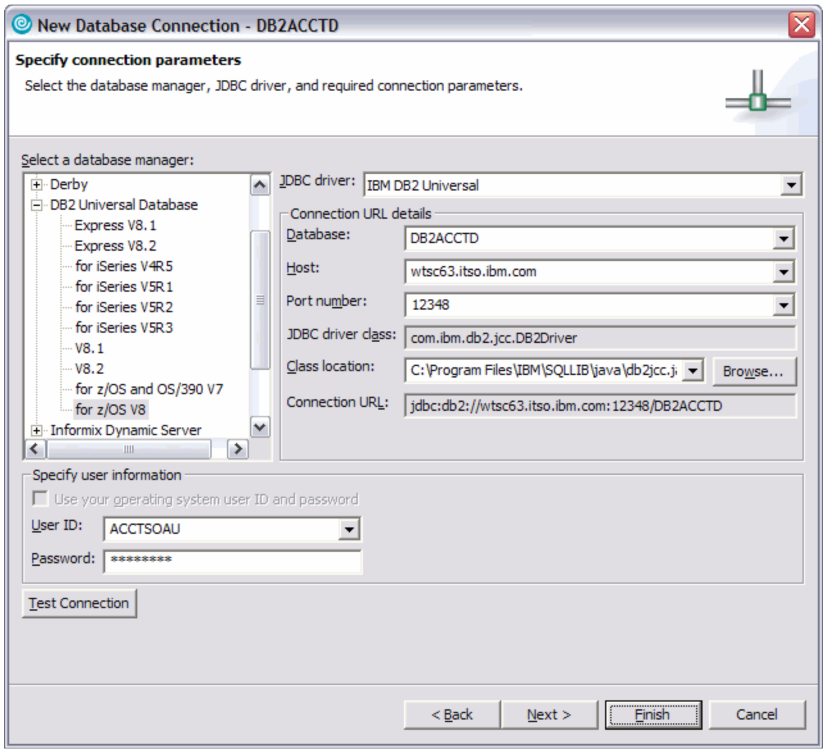
The following steps are shown in this scenario:

1. Create a database project for DB2ACCTD in RAD and import the database meta information (RAD wizards use this meta information to automate many steps).
2. Create a J2EE Web application which contains the WORF runtime environment and the DADX definitions.
3. Add a DADX group to the J2EE application and configure the access to the DB2ACCTD database.
4. Create a DADX file which contains the mappings of our DB2 stored procedures to Web Services operations.
5. Deploy the J2EE application into a WebSphere Application Server.

Import the DB2ACCTD database into a RAD database project

In our scenario the DB2ACCTD DB2 for z/OS database contains the ITSOBank accounting system. If you want to go through the scenario yourself, you can find instructions to create the DB2ACCTD database in Appendix E, "Additional material" on page 685.

We have to import the definition (we're interested mainly in the stored procedures) of the DB2ACCTD database into our RAD workspace. For this purpose, we create a database connection pointing to DB2ACCTD. Open the RAD tool and switch to the Data perspective (**Window** → **Open Perspective** → **Other** → **Data**). Then open the context menu of the Database Explorer view and select **New connection** to open the New Database Connection wizard. At the first page, enter an arbitrary name identifying the database connection and select **Choose a database manager and JDBC driver**. Change to the next page and enter the connection attributes of the DB2ACCTD database (Figure 15-3).


New Database Connection - DB2ACCTD

Specify connection parameters

Select the database manager, JDBC driver, and required connection parameters.

Select a database manager:
- Derby
- DB2 Universal Database
  - Express V8.1
  - Express V8.2
  - for iSeries V4R5
  - for iSeries V5R1
  - for iSeries V5R2
  - for iSeries V5R3
  - V8.1
  - V8.2
  - for z/OS and OS/390 V7
  - for z/OS V8
- Informix Dynamic Server

JDBC driver: IBM DB2 Universal

Connection URL details

Database: DB2ACCTD

Host: wtsc63.itso.ibm.com

Port number: 12348

JDBC driver class: com.ibm.db2.jcc.DB2Driver

Class location: C:\Program Files\IBM\SQLLIB\java\db2jcc.j

Connection URL: jdbc:db2://wtsc63.itso.ibm.com:12348/DB2ACCTD

Specify user information

Use your operating system user ID and password

User ID: ACCTSOAU

Password: ********

Figure 15-3 Database connection details of DB2ACCTD in RAD

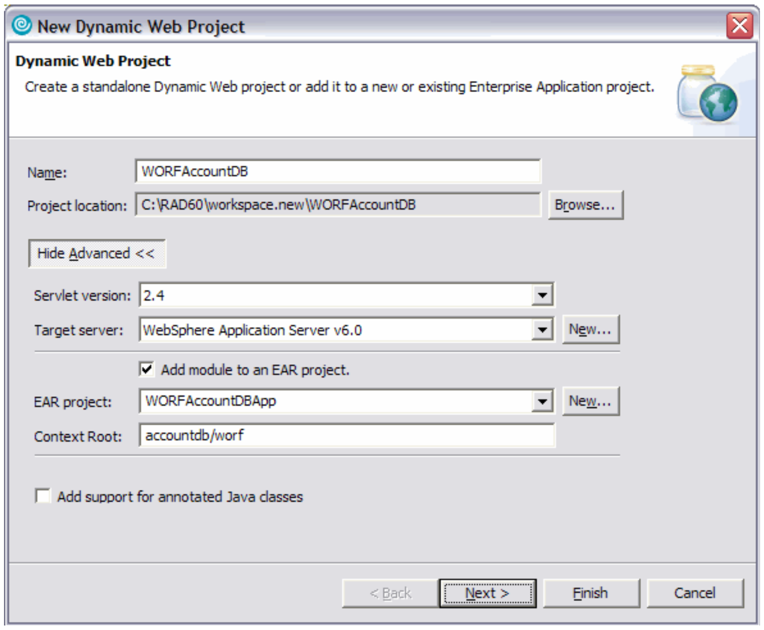
To check if the connection attributes are entered correctly, click **Test Connection**. Click Finish to have the new connection created.

The next step is important: RAD asks you if the database metadata shall be copied to a project folder. Confirm this question and enter a project name. Once the wizard finishes, the Database Explorer view contains the new database connection and the Data Definition view contains details (database objects like schemas, tables, views, stored procedures and user-defined functions) about the DB2ACCTD database.

Create a dynamic Web project

The next step is to create a dynamic Web project which will contain the WORF libraries as well as the XML configuration files specifying the Web Services interfaces to the stored procedures of the DB2ACCTD.

Open the Rational Application Developer and switch to the Web perspective (**Window** → **Open Perspective** → **Other** → **Web**). Select **File** → **New** → **Dynamic Web Project** and set the project properties as shown in Figure 15-4 on page 408. Click **Finish** to create the project.



Name: WORFAccountDB
Project location: C:\RAD60\workspace.new\WORFAccountDB
Servlet version: 2.4
Target server: WebSphere Application Server v6.0
☑ Add module to an EAR project.
EAR project: WORFAccountDBApp
Context Root: accountdb/worf
☐ Add support for annotated Java classes

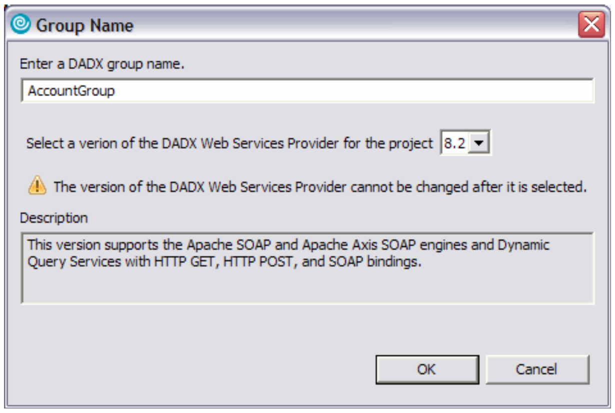
Figure 15-4   Create Web project for WORF application

The new Web project is added to the Dynamic Web Projects folder in the Project Explorer view. Also, a new EAR project named WORFAccountDBApp is added to the Enterprise Applications folder.

Setup DADX group configuration

Select **File** → **New** → **Other** → **Web Services** → **Web Services DADX Group Configuration**. A dialog comes up listing all Web projects. Select the WORFAccountDB project and click **Add group**. Enter the name of the new group (see Figure 15-5) and click **OK**. Select **Finish** on the next dialog. Since this is the first group added to this Web project, the DADX group creation wizard performs the following actions:

- Adds the WORF JAR files to the project's WEB-INF/lib folder.
- Creates a Web content folder named WORF, adding some WORF management pages.
- Adds the DADX group servlet and WORF management servlets to the Web deployment descriptor (file WEB-INF/Web.xml).
- Creates a folder with the DADX group name and adds a group.properties file to this folder.



Enter a DADX group name.
AccountGroup
Select a verion of the DADX Web Services Provider for the project 8.2
The version of the DADX Web Services Provider cannot be changed after it is selected.
Description
This version supports the Apache SOAP and Apache Axis SOAP engines and Dynamic Query Services with HTTP GET, HTTP POST, and SOAP bindings.

Figure 15-5   Add a new DADX group to project

If groups were added to the same project subsequently, only a new DADX group folder would be added and a new servlet for this group would be included in the project's Web.xml file.

As shown in Figure 15-5 on page 408, the wizard has an additional option to select the version of the DADX Web Services provider (you can choose between 8.1 and 8.2). Since version 8.1 only supports the Apache SOAP engine, whereas version 8.2 adds support for the Apache Axis engine as well as for Dynamic Query Services, it is recommended to use the second option. We do not go into the details of the Dynamic Query Services in this book.

Important: The version of the Rational Application Developer does not include support for the DADX Web Services provider version 9 (which is part of DB2 Version 9.1). Future versions of the RAD tool will include support for this version, so the dialogs and options shown in this chapter are subject to change.

Figure 15-6 shows the new DADX group in the Project Explorer view. The group creation wizard created new folder groups.AccountGroup, and added the files `group.properties` and `namespacetable.nst` to this folder. The second file is not important for our example (it is used together with the XML Extender capabilities of WORF), but the property file has to be updated with our database connection settings.

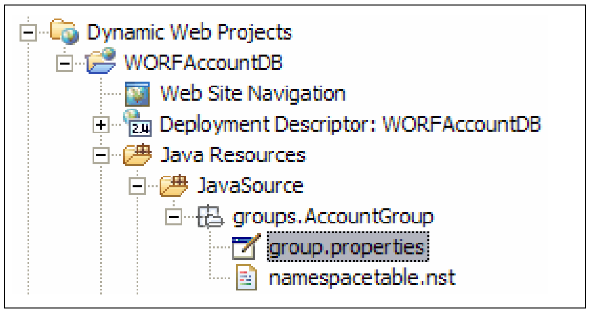


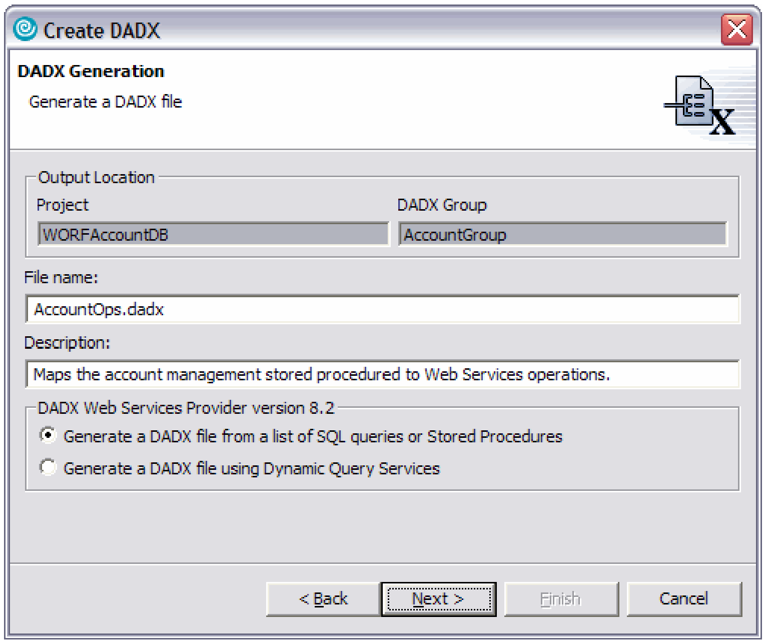
Figure 15-6 New DADX group in Project Explorer view

Open the `group.properties` file and set the properties dbDriver, dbURL, userID and password (read "Setup DADX data source" on page 414 to learn more about WORF database connectivity). The defaults of the other properties do not need to be changed. The WORF documentation contains complete information about the properties.

Create DADX file with stored procedures operations

Once the DADX group configuration is finished, we continue with defining the DADX operations for our stored procedures. The creation of the DADX file is to a large extent automated.

Select **File** → **New** → **Other** → **Web Services** → **DADX File** to open the DADX file creation wizard. On the first page of the wizard, select the correct project and DADX group and enter a name for the new DADX file (Figure 15-7 on page 410).


Create DADX

DADX Generation

Generate a DADX file

Output Location

Project	DADX Group
WORFAccountDB	AccountGroup

File name: AccountOps.dadx

Description: Maps the account management stored procedured to Web Services operations.

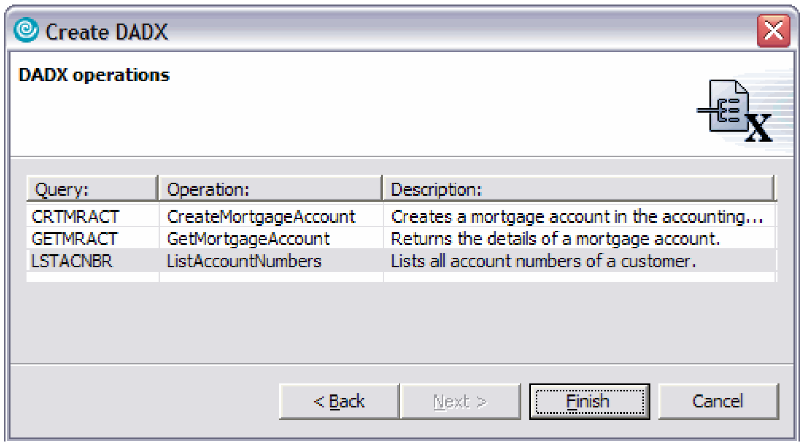
DADX Web Services Provider version 8.2

(●) Generate a DADX file from a list of SQL queries or Stored Procedures

( ) Generate a DADX file using Dynamic Query Services

Figure 15-7 Enter name of new DADX file

The next wizard page shows all database objects contained in any of the database definitions of the RAD workspace. Browse to the stored procedures of the DB2ACCTD database and select all the procedures which are added to the DADX file (multiple selections are supported). You can skip the following wizard page. The final wizard page lists all selected stored procedures and corresponding operation names. Update the operation names to more meaningful names as shown in Figure 15-8 and click Finish to complete this step.


Create DADX

DADX operations

Query:	Operation:	Description:
CRTMRACT	CreateMortgageAccount	Creates a mortgage account in the accounting...
GETMRACT	GetMortgageAccount	Returns the details of a mortgage account.
LSTACNBR	ListAccountNumbers	Lists all account numbers of a customer.

Figure 15-8 Review and update DADX operation names

The wizard analyzes the stored procedure signatures to create the correct syntax in the DADX file. It sets the `<SQL_call>` command and adds input/output parameter definitions. The structure of the result sets of a stored procedure are not declared in the procedures interface, so the wizard is not able to generate the `<result_set_metadata>` definitions, so they must be added manually.

Example 15-2 on page 411 shows the DADX file after the result set information has been added. The bold text marks the actual stored procedure calls; the remaining definitions define the types and structure of the parameters and result sets.

Example 15-2 Generated DADX file

```
<?xml version="1.0" encoding="UTF-8"?>
<dadx:DADX xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx dadx.xsd">
  <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
    Maps the account management stored procedured to Web Services operations.
  </dadx:documentation>
  <dadx:result_set_metadata  name="accountNumbersResult" rowName="accountNumber">
    <dadx:column name="ACCTNUM" type="CHAR" nullable="false" />
    <dadx:column name="ACCTTYPE" type="CHAR" nullable="false" />
  </dadx:result_set_metadata>
  <dadx:result_set_metadata  name="createAccountResult"
                             rowName="newAccountNumber">
    <dadx:column name="ACCTNUM" type="CHAR" nullable="false" />
  </dadx:result_set_metadata>
  <dadx:result_set_metadata  name="mortgageAccountResult"
                             rowName="mortgageAccountDetailsRow">
    <dadx:column name="CUSTNUM" type="CHAR" nullable="false" />
    <dadx:column name="AMOUNT" type="DECIMAL" nullable="false" />
    <dadx:column name="INTRATE" type="DECIMAL" nullable="false" />
    <dadx:column name="LIFETIME" type="DECIMAL" nullable="false" />
    <dadx:column name="OPENDATE" type="DATE" nullable="false" />
    <dadx:column name="PAYMENT" type="DECIMAL" nullable="false" />
    <dadx:column name="BALANCE" type="DECIMAL" nullable="false" />
  </dadx:result_set_metadata>
  <dadx:operation name="CreateMortgageAccount">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
      Creates a mortgage account in the accounting system.
    </dadx:documentation>
    <dadx:call>
      <dadx:SQL_call>
        CALL ACCTDB.CRTMRACT(:CUSTNUM, :AMOUNT, :INTRATE, :LIFETIME,
                             :PAYMENT, :PSQLCODE, :PSQLSTATE, :PSQLERRMC)
      </dadx:SQL_call>
      <dadx:parameter name="CUSTNUM" type="xsd:string" kind="in"/>
      <dadx:parameter name="AMOUNT" type="xsd:decimal" kind="in"/>
      <dadx:parameter name="INTRATE" type="xsd:decimal" kind="in"/>
      <dadx:parameter name="LIFETIME" type="xsd:decimal" kind="in"/>
      <dadx:parameter name="PAYMENT" type="xsd:decimal" kind="in"/>
      <dadx:parameter name="PSQLCODE" type="xsd:int" kind="out"/>
      <dadx:parameter name="PSQLSTATE" type="xsd:string" kind="out"/>
      <dadx:parameter name="PSQLERRMC" type="xsd:string" kind="out"/>
      <dadx:result_set name="createAccountReturn" metadata="createAccountResult"/>
    </dadx:call>
  </dadx:operation>
  <dadx:operation name="GetMortgageAccount">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
      Returns the details of a mortgage account.
    </dadx:documentation>
    <dadx:call>
      <dadx:SQL_call>
        CALL ACCTDB.GETMRACT(:ACCTNUM, :PSQLCODE, :PSQLSTATE, :PSQLERRMC)
      </dadx:SQL_call>
      <dadx:parameter name="ACCTNUM" type="xsd:string" kind="in"/>
      <dadx:parameter name="PSQLCODE" type="xsd:int" kind="out"/>
      <dadx:parameter name="PSQLSTATE" type="xsd:string" kind="out"/>
      <dadx:parameter name="PSQLERRMC" type="xsd:string" kind="out"/>
      <dadx:result_set name="mortgageAccountDetails"
                       metadata="mortgageAccountResult"/>
    </dadx:call>
  </dadx:operation>
  <dadx:operation name="ListAccountNumbers">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
      Lists all account numbers of a customer.
    </dadx:documentation>
    <dadx:call>
      <dadx:SQL_call>
        CALL ACCTDB.LSTACNBR(:CUSTNUM, :PSQLCODE, :PSQLSTATE, :PSQLERRMC)
      </dadx:SQL_call>
      <dadx:parameter name="CUSTNUM" type="xsd:string" kind="in"/>
      <dadx:parameter name="PSQLCODE" type="xsd:int" kind="out"/>
      <dadx:parameter name="PSQLSTATE" type="xsd:string" kind="out"/>
      <dadx:parameter name="PSQLERRMC" type="xsd:string" kind="out"/>
      <dadx:result_set name="accountNumbers" metadata="accountNumbersResult"/>
  </dadx:operation>
</dadx:DADX>
```

Now the development of the WORF-based application is finished. Additional DADX groups to connect to other databases, or new DADX files to include other stored procedures or SQL statements, can be added using the steps described in the last two chapters.

Deploy the J2EE application into a WebSphere Application Server

The last step is to deploy the application to an Application Server. In our scenario this server is a WebSphere Application Server 6.0 running in the intranet of the ITBO Bank company. The J2EE application is exported into an EAR file, which is installed on the WebSphere Application Server. To create the EAR file, go to the Project Explorer view in RAD (select **Window** → **Show View** → **Project Explorer**) and select the application from the Enterprise Applications folder. Right-click to open the context menu and select **Export** → **EAR file**. Enter a local filename (ending with a .ear suffix) into the Destination field and click Finish. Make sure the specified file has been created.

Go to the Web administration console of the WebSphere Application Server and select **Applications** → **Install New Application**. Enter the path to the exported .ear file under Local path. In the installation options, make sure the **Use Binary Configuration** option is selected[1], for the other settings the default values can be kept.

Important: The WORF support in RAD as explained here causes a problem with applications deployed to WebSphere Application Server 6.0 servers. The WORF libraries cannot load the SOAP engine classes because of a class loader problem. Follow the steps below after the application is installed.

Due to a class loader problem with WebSphere Application Server 6.0, the WORF libraries need to be moved from the application installation folder to the global WebSphere Application

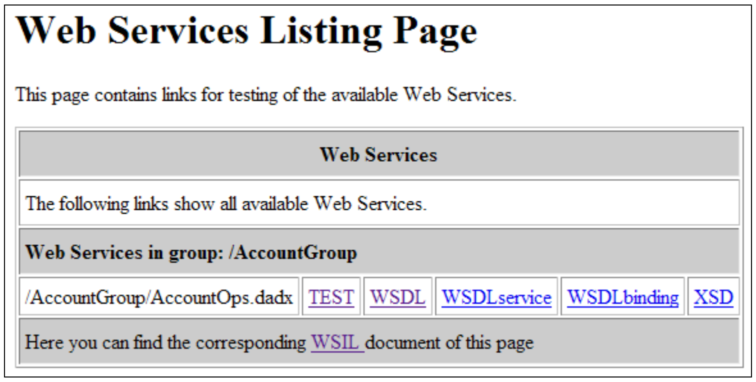
[1] If this option is selected, you are able to modify the application even after it is deployed, by updating the files in the installedApps folder. The application will pick up the changes after the time specified in the group.properties file. Supported changes are e.g. the modification of a DADX file, or the addition of a DADX file to the current DADX group.

Server libraries folder. Locate the files `worf.jar` and `worf-servlets.jar` files in the application install folder (`*WAS_install_folder*/profiles/default/installedApps`) and move them to `*WAS_install_folder*/lib`. You need to restart the WebSphere Application Server so that the change takes effect.

Once the WebSphere Application Server is restarted, check the server log to make sure the application started without errors. If the log contains error messages, check the "Troubleshooting" section for further information. Otherwise, open the following page with your Web browser.

```
http://Your_WAS_server/accountdb/worf/AccountGroup/LIST
```

The output of this page needs to look similar to Figure 15-9. The page contains a list of all DADX files of the AccountGroup group and also provides links to the files' WSDL and XSD definitions.



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: /AccountGroup					
/AccountGroup/AccountOps.dadx	TEST	WSDL	WSDLservice	WSDLbinding	XSD
Here you can find the corresponding WSIL document of this page					

Figure 15-9 WORF application account Web Services listing

The WORF library includes a test facility which allows you to call the DADX Web Service operations without the need to create a Web Service client proxy. Follow the TEST link in the Web Services Listing page to test the service we just created.

We completed creating a WORF-based Web Service which calls stored procedures contained in our accounting database. Web Service clients can use the WSDL of this Web Service to manage mortgage accounts.

Troubleshooting

This section lists a number of common problems encountered when trying to access a DADX Web Service, or when using one of the WORF services (like the test facility). In general, when you face a problem, look into the server log which contains additional information written by the WORF framework. It may help locating the cause of the problem.

DADX file is not listed in Web Services Listing page

If a DADX file which is included in a DADX group does not show up in the Web Services listing page of this group, it is most likely caused by syntax problems in the DADX file. Validate the file against the DADX XML Schema document.

Database connectivity problems

In case your WORF application has problems connecting to the DB2 database (for example, the result of a DADX Web Service call is a database failure, or the server log reports a database related problem), first check if the database connection properties in your `group.properties` file are correct.

If you use a JDBC Type 2 driver, it is possible that the DB2 JDBC drivers cannot be loaded or the DB2 environment is not initialized. In this case, setup the DB2 environment by calling the `<db2_instance_home>/sqllib/db2profile` before starting the WebSphere Application Server process.

WORF library problems

Sometimes the following error is reported when you try to access a DADX file:

```
Error 400: Can&apos;t load provider
&apos;com.ibm.etools.webservice.rt.framework.apache.ApacheServiceProvider&apos;
```

In this case you need to copy the WORF libraries to the WebSphere Application Server global libraries folder as described in "Deploy the J2EE application into a WebSphere Application Server" on page 412.

15.1.3 Additional WORF capabilities

In this section we explain additional features of the WORF framework and details which did not fit into the previous section.

Setup DADX data source

On DADX group level you can configure WORF to use either a JNDI data source defined in the application server to connect to the DB2 database, or use the JDBC connection attributes defined in the DADX group.properties file instead. The JNDI data source option is only supported for WebSphere Application Server. This configuration happens by setting the correct properties in the group.properties file. Table 15-1 shows which properties are relevant for either of these options.

Table 15-1 DADX group properties used for the data source JNDI lookup versus the JDBC connection

DADX group property	Used for JNDI data source lookup	Used for direct JDBC connection
initialContextFactory	Yes	No
datasourceJNDI	Yes	No
dbDriver	No	Yes
dbURL	No	Yes
userID	Yes	Yes
password	Yes	Yes

The purpose and usage of these properties is as follows:

initialContextFactory	Is the fully qualified Java class name of the JNDI context factory of the application server. For WebSphere Application Server version 6 it is `com.ibm.websphere.naming.WsnInitialContextFactory`.
datasourceJNDI	Specifies the JNDI data source used for DADX database connections, e.g. `jdbc/accountDBSource`.
dbDriver	The fully qualified Java class name of the JDBC driver (used for direct JDBC connections). If you use the DB2 Universal Driver, set this property to `com.ibm.db2.jcc.DB2Driver`.
dbURL	The JDBC URL of the data source, e.g. `jdbc:db2://wtsc63.itso.ibm.com:50000/DB2ACCTD`.

userID, **password**	Username and password used to authenticate when connecting to the DB2 database. You need to set these properties also when using a JNDI data source (the authentication data specified in the data source configuration is not used). Although this information can be left empty (in which case the Web server's runtime user is taken) we recommend to create and use a dedicated DB2 user which is granted permissions to access the database resources that are exposed through this interface.

Depending on the information provided in the properties file, WORF either uses the JNDI data source or the JDBC information to connect to the DB2 database. If both, JNDI and JDBC information is given, the JNDI data source has precedence.

Tip: We recommend to use a JNDI data source. JNDI data sources support features like connection pooling or distributed transactions. Also, you can manage the data source settings using the WebSphere Application Server administration console (as compared to updating the `group.properties` file).

Select the SOAP engine of a WORF application

The WORF library which is bundled with DB2 version 8.2 supports two SOAP engines, the one included in the Apache SOAP library (`http://ws.apache.org/soap/`) and the SOAP engine of Apache Axis (`http://ws.apache.org/axis/`). Per default, the Apache Axis engine is used. In case you need to use a different SOAP engine (for example, because the server does only support one of the engines, or because of SOAP compatibility issues) you can specify the SOAP engine in the Web deployment descriptor.

Open the Web deployment descriptor of your application in RAD, change to the Source tab and add a servlet parameter to the servlet of your DADX group (note that each DADX group has a servlet, and the name of the servlet is the group name). The name of the servlet parameter is `soap-engine`, and the parameter value is either `apache-soap` or `apache-axis`, depending on which engine you want to use. Example 15-3 shows how to set the Apache SOAP engine for our AccountGroup DADX group. The section in bold has been added to the web.xml file.

Example 15-3 SOAP engine selection in web.xml

```
...
<servlet>
  <servlet-name>AccountGroup</servlet-name>
  <servlet-class>
      com.ibm.etools.webservice.rt.dxx.servlet.DxxInvoker</servlet-class>
  <init-param>
    <param-name>soap-engine</param-name>
    <param-value>apache-soap</param-value>
  </init-param>
</servlet>
...
```

This change needs to happen before the application is deployed to the Application Server.

Information: The WebSphere Application Server product includes both, the Apache SOAP and Apache Axis engines, so you do not need to install these engines when running your WORF application in WebSphere Application Server. If you use a different Application Server, the engine libraries have to be installed. The *IBM Information Integrator Application Developer's Guide* contains instructions to install the SOAP engine on a Apache Jakarta Tomcat server.

Use of document/literal SOAP encoding

You can use a DADX group property to define whether the Web Services SOAP encoding style has to be RPC-based or document-based (the SOAP encoding styles are compared in section "SOAP messaging mechanisms" on page 56). If you want to use the document/literal style (which is compliant with the WS-I profile), set the `useDocumentStyle` in the property file to `yes`, if you want to use RPC/encoded style, set it to `no`.

Note that the Apache SOAP engine does only support RPC/encoded (so this setting does not have an effect when using this engine); Apache Axis supports both encoding styles.

Security considerations

The fine-grained DB2 authentication and authorization mechanisms (restricting access to database objects and stored procedures on a user level or group level) can be leveraged in WORF applications only in a limited manner. All database accesses of Web Services belonging to a DADX group are performed with the user credentials contained in the property file. Hence the security context of the Web Service caller cannot be mapped to DB2 authorization levels.

If you need to implement an authorization concept, you can utilize the J2EE security model instead. At example, assume a simplified model which restricts the access to stored procedures of our accounting database based on different roles:

- Front-desk clerks can only query account details and create loan requests.
- Accountant managers are allowed to additionally approve loan requests and generate quarterly reports.

The roles of each of our employees are stored in an LDAP directory together with the other employee data.

The J2EE security model has the capability to define a role-based access to Web Services. So we put the operations which can be accessed by both roles into a different DADX file than the operations only available for accountant managers. In the Web deployment descriptor of our J2EE application we create two security roles, one for clerks and one for accountant managers. Also, we define a security constraint which allows to access the first Web Service by both roles, and access to the second one only for accountant managers.

Finally, when we deploy our application to the Application Server, we configure the server to use our company's LDAP server for authentication, and map the application roles to the corresponding LDAP groups. So whenever a client calls one of our Web Services, it has to provide the credentials of the user who initiates the action via HTTP Basic Authentication. The J2EE container then retrieves the user's group from LDAP and matches it against the security constraints defined in the Web deployment descriptor of our application. If the constraints are fulfilled, the Web Service is processed, otherwise an authentication error is returned to the client.

If you are not familiar with the J2EE security model, you can find an introduction in Sun's *J2EE 1.4 Tutorial* at:

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security4.html

Important: The Application Server has to support the J2EE security model. WebSphere Application Server version 5 and 6 provide the required support.

15.1.4 Web Service implementation using Java wrappers

A different approach to expose the stored procedures of our accounting database as Web Services is to wrap the stored procedure calls into a Java object (a simple Java class, or a more complex Enterprise JavaBean) which is converted into a Web Service. The Web Service is deployed to an Application Server as part of a J2EE application.

We demonstrate the creation of a Java class which calls the stored procedure using JDBC, and the conversion of this Java class into a Web Service using the Rational Application Developer.

Create a dynamic Web project

We start with the creation of a Web project (and a corresponding Enterprise Application project) in RAD. Switch to the Web perspective (**Window** → **Open Perspective** → **Other** → **Web**). Select **File** → **New** → **Dynamic Web Project** and set the properties of the new project. In our example we use JavaAccountDB as the Web project name, JavaAccountDBApp as the EAR project name, and accountdb/java as the project's context root. Click **Finish** to complete this step. The RAD wizard adds the Web project to the Dynamic Web Projects folder in the Project Explorer view, and the new EAR project to the Enterprise Applications folder.

Define the skeleton of the Java class

Our task is to enable three stored procedures (LSACNBR, GETMRACT and CRTMRACT) for Web Services. So we create a stateless[2] Java class which contains one method for each of these stored procedures. All required parameters are defined as arguments of the methods. The skeleton of this Java class is shown in Example 15-4.

Example 15-4 Skeleton of AccountService Java class

```
package com.ibm.itso.sg247259.acctdb;

import java.sql.*;
import java.util.*;

/**
 * Java wrapper for accounting DB stored procedure calls.
 *
 * Calls stored procedures of ITSO Bank's accounting database.
 * This implementation is stateless, so each call of one of its
 * methods opens a separate connection to the accounting
 * database.
 */
public class AccountService {

  /**
```

[2] Web Services are stateless by nature, which means that a Web Service can't remember information from one invocation to another. The Web Services standards do not specify state management and implementations in general do not provide means to support state management (*cmp.* other standards like CORBA or Java Remote Method Invocation which include state management). You can implement stateful Web Services by using additional concepts like Java Servlet-based Web Service endpoints.

```
   * Call the ACCTDB.LSACNBR stored procedure with the given customer
   * number and return the results. Throws a DBException if the stored
   * procedure returns an error.
   *
   * @param userID         Database connection user.
   * @param password       Database connection password.
   * @param custNum        Customer number to search for.
   * @return AccountRow[]  All accounts owned by the given customer number.
   */
  public AccountRow[] getAccountNumbers(
      String userID, String password, String custNum)
    throws DBException {
  }

  /**
   * Call the ACCTDB.GETMRACT stored procedure with the given account
   * number and return the details of that account. Throws a DBException
   * if the account can't be found or if the stored procedure returns an
   * error.
   *
   * @param userID         Database connection user.
   * @param password       Database connection password.
   * @param acctNum        Account number to search for.
   * @return MortgageAccountRow Details of the mortgage account.
   */
  public MortgageAccountRow getMortgageAccount(
      String userID, String password, String acctNum)
    throws DBException {
  }

  /**
   * Call the ACCTDB.CRTMRACT stored procedure to create a new mortgage
   * account and return the account number of the new account. Throws a
   * DBException if the creation of the account fails or if the stored
   * procedure returns an error.
   *
   * @param userID         Database connection user.
   * @param password       Database connection password.
   * @param custNum        Customer number of requestor.
   * @param amount         Mortgage amount.
   * @param interestRate   Account's interest rate.
   * @param lifetime       Lifetime of mortgage (in months).
   * @param monthlyPayment Monthly payment for mortgage.
   * @return String        New account number.
   */
  public String createMortgageAccount(
      String userID, String password, String custNum, double amount,
      double interestRate, int lifetime, double monthlyPayment)
    throws DBException {
  }
}
```

Here are notes on the method definitions in Example 15-4 on page 417:

- Additionally to the stored procedure parameters, each of the wrapper methods has the arguments `userID` and `password`, which are the user credentials for the connection to the accounting database. The approach of specifying this data at the method is only one of the options to implement database connectivity (alternatively, you can use the Factory design pattern to retrieve a connection object, and store the authentication credentials in property files or in runtime configuration settings). We use this approach here to demonstrate the flexibility of the Java-based application as opposed to WORF-based application (which allows authentication only on J2EE level).
- The error handling of our stored procedures is performed by returning an SQL code, SQL state and additional message. The caller of the stored procedure has to check the SQL code to detect if the call was successful. Java and SOAP support exception management which fits this task better. So we mask all error conditions with exceptions instead, using the class `DBException`. If a client catches an exception after a Web Service call, the details of the error (SQL code, SQL state) can be accessed via attributes of the `DBException` object.
- The result sets of the stored procedures cannot be simply returned by the Java methods, since a SOAP data type equivalent does not exist. So we create JavaBeans (the beans `AccountRow` and `MortgageAccountRow` in our example) whose attributes hold the information returned in the result sets.

Important: All parameters and return values of wrapper methods have to be either basic data types, or JavaBeans. If you use a JavaBean, make sure it obeys the conventions of the JavaBeans specification:

- The class is serializable.
- It has a default constructor (a constructor with no arguments).
- All of its properties can be accessed using getter/setter methods that follow the standard naming convention.

Create a database connection class

it is good practise to encapsulate the JDBC connectivity code into a separate class. So we create a singleton class `DBConnectionProvider` that establishes and returns a connection to the accounting database at request. Example 15-5 contains the code of this class.

Example 15-5 DBConnectionProvider class handling JDBC connectivity

```
package com.ibm.itso.sg247259.acctdb;

import java.sql.*;

/**
 * Encapsulates the functionality to load the JDBC driver and
 * establish a connection to the accounting database.
 */
public class DBConnectionProvider {

  /** Contains the JDBC initialization status */
  private static boolean initialized = false;

  /** JDBC URL pointing to accounting database */
  private static final String JDBC_URL =
    "jdbc:db2://wtsc63.itso.ibm.com:12348/DB2ACCTD";
  static {
    try {
      // load the DB2 JDBC driver
      Class.forName("com.ibm.db2.jcc.DB2Driver");

      // set the JDBC status to initialized
      initialized = true;
    }
    catch (ClassNotFoundException cnfe) {
      System.out.println("Unable to load DB2 JDBC driver: " + cnfe.getMessage());
    }
  }

  /**
   * Establishes a JDBC connection to the accounting database.
   *
   * @param userID      Database connection user.
   * @param password    Database connection password.
   * @return Connection New accounting database connection.
   */
  public static Connection getConnection(String userID, String password)
    throws DBException {

    // check if JDBC is initialized
    if (!initialized)
      throw new DBException(-1, "", "JDBC driver is not loaded!");

    try {
      // create and return database connection
      return DriverManager.getConnection(JDBC_URL, userID, password);
    }
    catch (SQLException se) {
      throw new DBException(se.getErrorCode(), se.getSQLState(), se.getMessage());
    }
  }
}
```

Notes to the implementation of the `DBConnectionProvider` class in Example 15-5 on page 419:

- Our approach is to create a different user context for each Web Service call, so we have to specify the database user and password at each `getConnection(...)` call. As mentioned in the last section, this puts us in a position to extend the DB2 authorization context to the Web Service caller. On the other hand, it is not very efficient since connection pooling cannot be used.
- The JDBC URL is defined as a class constant. It points to the ITSO Bank accounting database which is running on a z/OS DB2 instance.

  To increase the maintainability of this application you can put it into a property file instead (which can be updated even if the application is already deployed).
- Instead of tying the implementation to a specific JDBC driver class (the DB2 Universal JDBC Driver in our case) you should consider to use the JNDI DataSource interface. Here a short example demonstrating how to acquire a JDBC data source connection:

  ```
  javax.naming.Context context = new javax.naming.InitialContext();
  javax.sql.DataSource ds =
  ```

```
    (javax.sql.DataSource) ctx.lookup("jdbc/DataSource");
java.sql.Connection con = ds.getConnection("userID", "password");
```

You define the attributes of the data source (JDBC driver classes, JDBC URL, optionally connection pooling parameters) in the J2EE Application Server, having the benefit that the actual configuration settings do not need to be known until the application is deployed.

You can find additional information about data sources and JDBC in the *Sun Advanced JDBC Tutorial* at:

http://java.sun.com/developer/Books/JDBCTutorial/index.html

Note: In our example we use the JDBC DB2 Universal Database Driver (`com.ibm.db2.jcc.DB2Driver`). This JDBC driver is included in the DB2 installation and has to be included in the class path of our application. Put the driver package (files `db2jcc.jar` and `db2jcc_license_cisuz.jar`) either into the WEB-INF/lib folder of your application or add it to the class path of your Application Server.

Call the stored procedures from the Java class

The last development-related step is to implement the Java wrapper code which calls the stored procedures via JDBC and converts the result sets to Java objects. We need to implement the three methods which headers are listed in Example 15-4 on page 417. Since the implementation is similar for these three methods, we show the actual code only for the `getAccountNumbers()` method (Example 15-6).

Example 15-6 Implementation of AccountService.getAccountNumbers() method

```
public AccountRow[] getAccountNumbers(
    String userID, String password, String custNum)
  throws DBException {

  // request database connection
  Connection conn = DBConnectionProvider.getConnection(userID, password);
  CallableStatement stmt = null;
  ResultSet rs = null;

  try {
    // prepare the LSTACNBR stored procedure
    stmt = conn.prepareCall("{ CALL ACCTDB.LSTACNBR (?, ?, ?, ?) }");

    // set parameters of stored procedure
    stmt.setString(1, custNum);
    stmt.registerOutParameter(2, Types.INTEGER);
    stmt.registerOutParameter(3, Types.CHAR);
    stmt.registerOutParameter(4, Types.VARCHAR);

    // execute the stored procedure
    rs = stmt.executeQuery();

    // check return code of stored procedure
    int sqlCode = stmt.getInt(2);
    if (sqlCode != 0)
      throw new DBException(sqlCode, stmt.getString(3), stmt.getString(4));

    // fetch the returned result set
    List list = new ArrayList();
    while (rs.next())
```

```
      list.add(new AccountRow(rs.getString(1), rs.getString(2)));

    // pass the result to the caller
    return (AccountRow[]) list.toArray(new AccountRow[list.size()]);
  }
  catch (SQLException se) {
    throw new DBException(se.getErrorCode(), se.getSQLState(), se.getMessage());
  }
  finally {
    try {
      // resource cleanup
      if (rs != null) rs.close();
      if (stmt != null) stmt.close();
      if (conn != null) conn.close();
    }
    catch (SQLException se) {
      // don't do anything
    }
  }
}
```

The following actions happen in this wrapper method:

1. The method acquires a new database connection, passing the current user credentials.
2. It prepares the stored procedure call and binds the input and output parameters of the procedure.
3. It executes the stored procedure call.
4. It checks the PSQLCODE output parameter to detect if the call was successful. If it encounters an error, it throws an exception.
5. It traverses the result set and creates JavaBean objects which are initialized with the values of the result set.
6. It returns the JavaBean objects.

This sequence is the same for all stored procedure wrapper methods. Although the amount of produced code is not very large, this step takes the most development effort, which is significant if a large number of stored procedures need to be wrapped.

Create WSDL from Java class

RAD supports the generation of the WSDL from the Java class with a wizard. To start this wizard select **File** → **New** → **Other** → **Web Services** → **Web Service**.

Note: If you have not previously enabled the Web Services development capability within Rational Application Developer, you will see a dialog with the message *This action requires the Web Services Development. Enable the required capability?* Click **OK**.

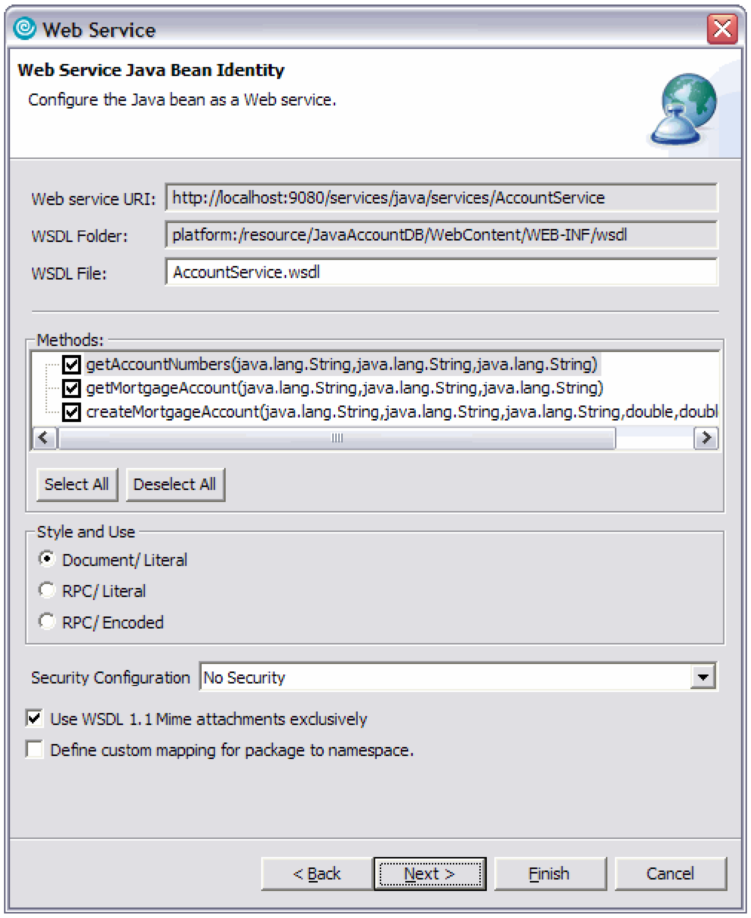
On the next page, select Java bean Web Service as the Web Service type. You can optionally create a client proxy, and a a JSP-based test framework, but we do not select these options for our example.

The next page allows you to enter the Java class. Browse for the AccountService class and select it. On the next page you can select the Web Service runtime. RAD currently supports IBM SOAP, Apache Axis 1.0 and the WebSphere runtime (use IBM WebSphere). Also, the

J2EE Application Server can be selected, but we keep the WebSphere Application Server v6.0 server setting. To complete this page, the Web Service project (it is JavaAccountDB in our example) and the EAR project (JavaAccountDBApp) need to be entered.

Do not select *Use an existing service endpoint interface* on the next page. We have the wizard create an interface (the service endpoint interface is a Java Interface which contains all methods that are contained in our Java wrapper class).

Figure 15-10 on page 423 shows the next page. On this page you can customize the generated WSDL. It lists all methods of the Java wrapper class, and all methods which need to be defined in the WSDL as operations have to be checked. The SOAP encoding style can be selected, and the wizard supports the use of Web Services security (XML Encryption and/or XML Signature of the Web Service SOAP body).


Web Service Java Bean Identity

Configure the Java bean as a Web service.

Web service URI: http://localhost:9080/services/java/services/AccountService

WSDL Folder: platform:/resource/JavaAccountDB/WebContent/WEB-INF/wsdl

WSDL File: AccountService.wsdl

Methods:
- [✓] getAccountNumbers(java.lang.String,java.lang.String,java.lang.String)
- [✓] getMortgageAccount(java.lang.String,java.lang.String,java.lang.String)
- [✓] createMortgageAccount(java.lang.String,java.lang.String,java.lang.String,double,doubl

Style and Use: [✓ Document/ Literal] RPC/ Literal RPC/ Encoded

Security Configuration: No Security

[✓] Use WSDL 1.1 Mime attachments exclusively

[ ] Define custom mapping for package to namespace.

Figure 15-10 RAD Web Service wizard: select operations

Keep the defaults on this page. All methods of our wrapper class are exposed as Web Service operations, the SOAP encoding style is document/literal and we do not use Web Service security at this time. Once you select **Finish**, the wizard not only generates the WSDL, but it also creates a couple of Java helper classes (performing the SOAP serialization of the data) and extends the Web deployment descriptor of the J2EE application with the mappings required to call the Web Service endpoint. Since the wizard performs all of these configuration tasks, our J2EE application is now ready for deployment.

Tip: The redbook *Rational Application Developer V6 Programming Guide*, SG24-6449, contains comprehensive information about the functionality and usage of the RAD Web Services wizard.

If you need to change your Java code after you created the WSDL bindings, you do not need to repeat this step, unless you change the Java interface (for example, modifying the parameter list of a method you expose as Web Service operation, or adding another method to the `AccountService` class.

Deploy the J2EE application into a WebSphere Application Server

We selected the IBM WebSphere SOAP runtime during the creation of the Web Service which means that our application can only be installed on an WebSphere Application Server (if you choose for example, Apache Axis in the Web Service wizard, you can install the application on any server which supports Axis). Our Application Server is a WebSphere Application Server v6.0 which runs in the intranet of the ITSO Bank company.

Similar to the installation of the WORF-based J2EE application we first export an EAR file. Select the JavaAccountDBApp project in the RAD Project Explorer and right-click to open the context menu. Select **Export** → **EAR file**, enter a local filename into the Destination field and select **Finish**. Make sure the specified file was created by the export wizard.

Go to the Web administration console of the WebSphere Application Server and select **Applications** → **Install New Application**. Enter the path to the exported .ear file under Local path; the other installation options do not need to be changed. Once the application is installed, save the configuration and start the application from the **Applications** → **Enterprise Applications** page.

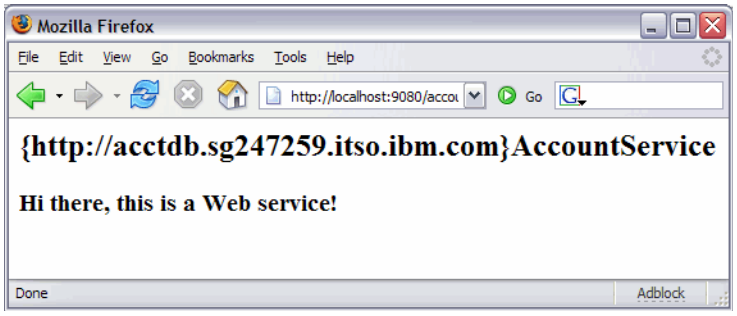
The WSDL of your new service is available at:

`http://`*`Your_WAS_server`*`/accountdb/java/AccountService?WSDL`

Open the WSDL to make sure the Web Service was installed correctly. It contains the location of the service endpoint, which should be

`http://`*`Your_WAS_server`*`/accountdb/java/services/AccountService`

The expected result when opening the service endpoint URL is shown in Figure 15-11.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/accou
Go
Done
Adblock

{http://acctdb.sg247259.itso.ibm.com}AccountService

Hi there, this is a Web service!

Figure 15-11 Java Web Service endpoint

Note: In order to test the Web Service, you can create a Web Service client proxy (in 15.4.4, "Creating Java client proxies for Web Service interfaces" on page 469 we demonstrate how to create client proxies). If you do not get the expected results, debug the Web Service in RAD and check the return value of the Web Service as well as the system console of the WebSphere Application Server.

15.1.5 Comparison of WORF-based and Java-based implementations

The previous chapters gave an insight into the activities required to perform a SOA enablement of the stored procedures of the ITSO Bank accounting database. This chapter once again points out the main differences between these approaches:

- **Development effort**

  When Java wrappers are used for stored procedures, a substantial amount of manual effort is required to implement the wrappers, since it is not supported by tools. Using WORF, no development tasks are required at all, and the generation of the DADX files is automated to a large extent (only the result sets of the stored procedures need to be defined manually).

- **Security considerations**

  The WORF-based approach has only limited support for security. Existing Web Services security standards like WS-Security or WS-Authentication cannot be used, so application developers need to resort to transport level security, which is SSL in most cases. A Java-based approach on the other hand can implement any security concept which is supported by the Web Services engine which is used.

- **Authentication and object authorization**

  As demonstrated in our example, a solution using Java wrappers is able to extend the authentication context of the DB2 database up to the Web Service caller by passing user credentials in the Web Service request. In addition, the J2EE security mechanisms or any custom, Java-based security frameworks can be used as well. A WORF application is restricted to the use of J2EE security mechanisms.

- **Exception handling**

  The WORF application simply uses the error handling mechanism of the stored procedures. If a stored procedure returns an SQL code, the code is included in the SOAP response; if an SQL exception is thrown, the exception is propagated to the Web Service caller. Whereas a Java wrapper class allows to implement any exception handling strategy, for example, masking SQL errors by exceptions as in our example, or do not propagate SQL error conditions at all.

- **Database connectivity**

  Both approaches are on equal terms in regard to available connectivity options. it is possible to define a JNDI data source or configure a JDBC driver class with connection properties in the WORF application as well as the Java-based application.

In summary, a Java-based approach is costly, but provides utmost flexibility for the design of the wrapper application and can meet a range of different requirements. WORF on the other hand is very simple to handle and has the capability for a fast go-to-market solution.

15.2 Scenario using DB2 as Web Service consumer

Two of the main building blocks of integrated SOA systems are Web Services providers and Web Services consumers. We've already shown the capabilities of DB2 acting as a Web Service provider. The focus of this chapter is to point out a scenario in which DB2 consumes a Web Service from a different service provider.

15.2.1 Overview

Our banking application provides a special feature for premium users: Their FICO score is shown on the online e-banking portal.

Note: The FICO score is a credit score, which is used by lenders to determine your credit risk. This score is one of the most important factors in obtaining credit in the United States. The FICO score is a number between 300 and 850, determined using a mathematical equation that evaluates the information in your credit file. For institutions that use scores as a factor in their lending decisions, scores below certain numbers (typically set by each lender's risk management department) may result in denial of credit, or credit being offered at a higher interest rate.

More information about this credit score can be found at:

http://en.wikipedia.org/wiki/Credit_score

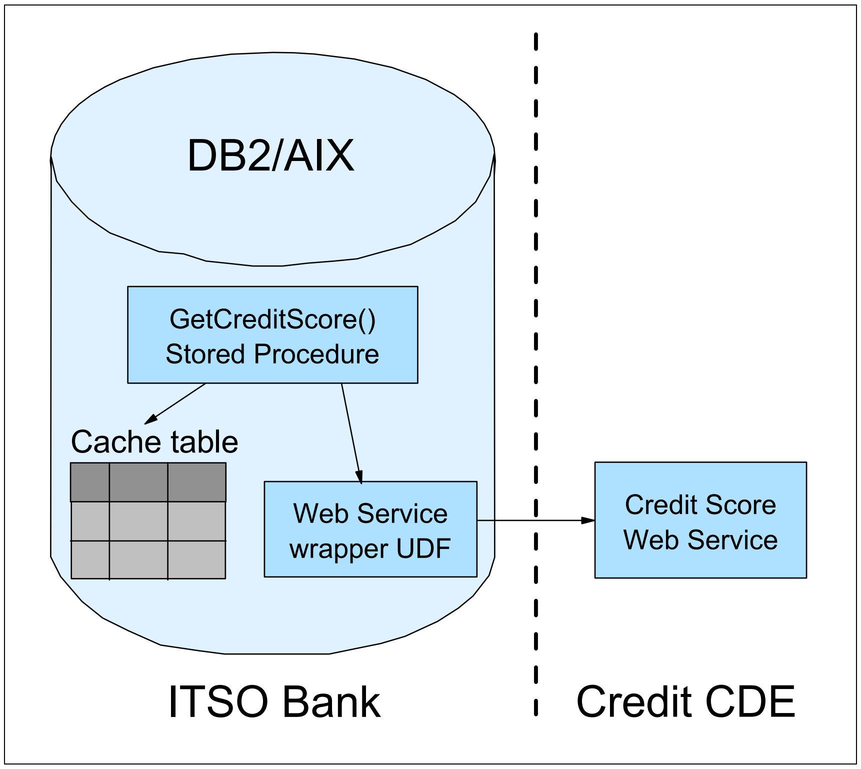


Figure 15-12 GetCreditScore() stored procedure

This credit score is not available within our system, so we retrieve the information from the credit institution company Credit CDE. The FICO score for a specific person is queried by calling a Web Service provided by this Credit CDE.

Since the usage of this service is not free (a service fee is charged by the credit institution for each call), and the credit score is not very volatile, we cache the score for our customers in the local DB2 database with an expiration period of one month (that is, if the local cached score of a customer is older than one month, the updated score is retrieved from the credit institution).

Figure 15-12 displays the components we use and develop in this scenario: The Web Service of the Credit CDE company is called by a wrapper UDF which is created using the Rational Application Developer. A stored procedure named `GetCreditScore()` performs the cache management: It checks if a credit score for a given customer is stored in the local table; if not, or if the local cache value got invalidated, the wrapper UDF is called. An application which wants to retrieve a customer's credit score calls the `GetCreditScore()` procedure.

The WSDL of the Credit CDE Web Service is publicly available. It is contained in Example 15-7.

Example 15-7 WSDL for credit score Web service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://services.creditcde.com"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://services.creditcde.com"
    xmlns:intf="http://services.creditcde.com"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types/>
  <wsdl:message name="getCreditScoreRequest">
    <wsdl:part name="reqName" type="xsd:string"/>
    <wsdl:part name="reqPassword" type="xsd:string"/>
    <wsdl:part name="ssn" type="xsd:string"/>
    <wsdl:part name="lastName" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getCreditScoreResponse">
    <wsdl:part name="getCreditScoreReturn" type="xsd:int"/>
  </wsdl:message>
  <wsdl:portType name="CreditQuery">
    <wsdl:operation
        name="getCreditScore" parameterOrder="reqName reqPassword ssn lastName">
      <wsdl:input
          message="impl:getCreditScoreRequest" name="getCreditScoreRequest"/>
      <wsdl:output
          message="impl:getCreditScoreResponse" name="getCreditScoreResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CreditQuerySoapBinding" type="impl:CreditQuery">
    <wsdlsoap:binding
        style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getCreditScore">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getCreditScoreRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://services.creditcde.com" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getCreditScoreResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://services.creditcde.com" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CreditQueryService">
    <wsdl:port binding="impl:CreditQuerySoapBinding" name="CreditQuery">
      <wsdlsoap:address
          location="http://services.creditcde.com/services/CreditQuery"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

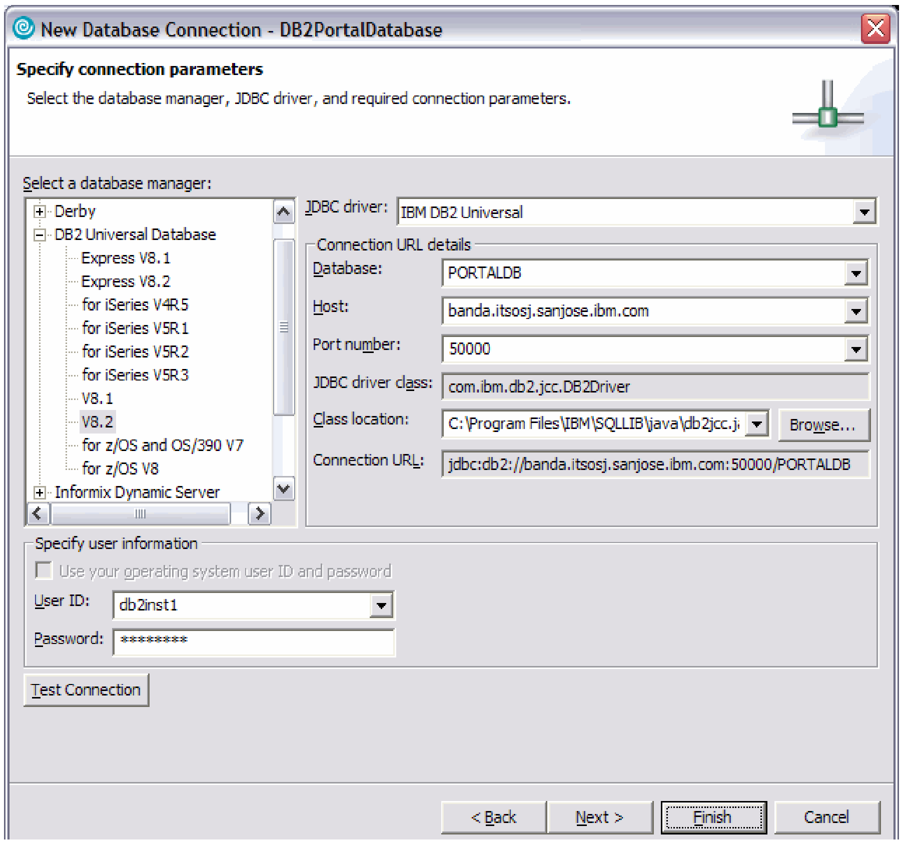
15.2.2 Implementation of the Credit Score function using RAD

This chapter describes a way to create the required objects and functions using the Rational Application Developer. They can also be created without tool support, but the RAD tool provides a number of wizards to accelerate the development process.

The Credit Score function is created in our PORTALDB database. This database is used subsequently in other scenarios as well. You can find instructions to create this database on your workstation in Appendix E, “Additional material” on page 685.

Create a database project for PORTALDB

When the PORTALDB database is created, open the Rational Application Developer and switch to the Data perspective (**Window** → **Open Perspective** → **Other** → **Data**). Open the context menu of the Database Explorer view and select **New connection** to open the New Database Connection wizard. At the first page, enter a name which identifies the connection later on and select **Choose a database manager and JDBC driver**. On the next page, you have to enter the connection attributes (Figure 15-13). Enter the connection details of your PORTALDB Database and click **Finish**.


New Database Connection - DB2PortalDatabase

Specify connection parameters

Select the database manager, JDBC driver, and required connection parameters.

Select a database manager:

- Derby
- DB2 Universal Database
  - Express V8.1
  - Express V8.2
  - for iSeries V4R5
  - for iSeries V5R1
  - for iSeries V5R2
  - for iSeries V5R3
  - V8.1
  - V8.2
  - for z/OS and OS/390 V7
  - for z/OS V8
- Informix Dynamic Server

JDBC driver: IBM DB2 Universal

Connection URL details

Database: PORTALDB

Host: banda.itsosj.sanjose.ibm.com

Port number: 50000

JDBC driver class: com.ibm.db2.jcc.DB2Driver

Class location: C:\Program Files\IBM\SQLLIB\java\db2jcc.j

Connection URL: jdbc:db2://banda.itsosj.sanjose.ibm.com:50000/PORTALDB

Specify user information

Use your operating system user ID and password

User ID: db2inst1

Password: *********

Figure 15-13   New database connection wizard in RAD

Next, RAD asks you to copy the database metadata to a project folder. Confirm this question and enter a project name. Once the wizard finishes, the Database Explorer view contains the new database connection and the Data Definition view contains details (database objects like schemas, tables, views, stored procedures and user-defined functions) about the database.

Preparations

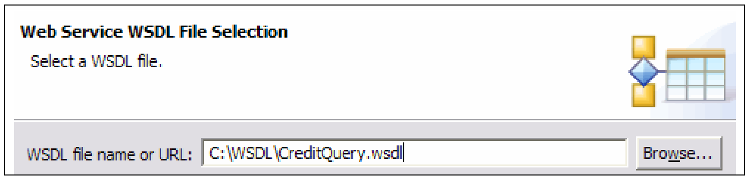
The user-defined functions and the cache table for the credit score component are created in our PORTALDB database. To separate the component from other components in the database, we put it into a separate schema, named FICO.

This schema can be created within the RAD application using the Schema Definition wizard. Select **File** → **New** → **Other** → **Data** → **Schema Definition** and click Next. On the next page, select the PORTALDB database and enter FICO as Schema name. Click **Finish** to create the schema.

Generate the Web Service wrapper UDF

Calling the DB2 SOAP-based Web Service consumer functions is not straightforward, since it requires to manually compose the SOAP body of the Web Service request. This composition step can be automated by deriving the SOAP body from the given Web Service definition (WSDL) file. RAD provides a wizard which performs exactly this task and creates a user-defined function acting as a wrapper for the Web Service call, requiring only minor input from the developer.

To open this wizard in RAD, select **File** → **New** → **Other** → **Data** → **Web Service User-Defined Function** and click Next. Enter the WSDL location (if the WSDL is stored in a local file, enter the full path of this file; otherwise, enter an URL pointing to the WSDL) as shown in Figure 15-14.

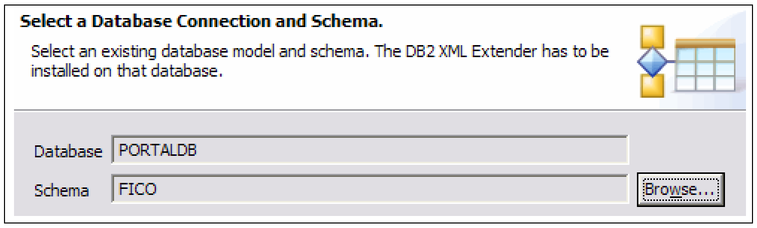

Web Service WSDL File Selection

Select a WSDL file.

WSDL file name or URL: C:\WSDL\CreditQuery.wsdl

Figure 15-14 Web Service UDF wizard: Select WSDL file

The next page allows you to select the database and schema in which the UDF will be created. Select the FICO schema of the PORTALDB database (Figure 15-15).


Select a Database Connection and Schema.

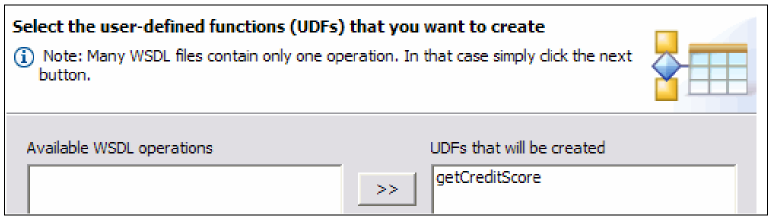
Select an existing database model and schema. The DB2 XML Extender has to be installed on that database.

Database PORTALDB

Schema FICO

Figure 15-15 Web Service UDF wizard: Select database schema

On the next page all available operations of the Web Service (as defined in the WSDL) are listed. The wizard creates a separate wrapper UDF for each operation. Move the operations you want to use to the list box on the right side (as shown in Figure 15-16).


Select the user-defined functions (UDFs) that you want to create

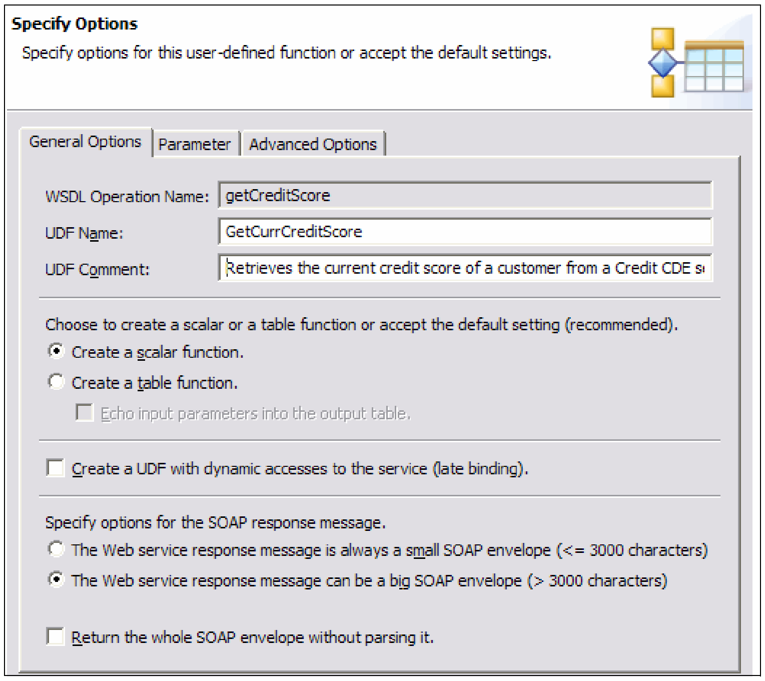
Note: Many WSDL files contain only one operation. In that case simply click the next button.

Available WSDL operations

UDFs that will be created: getCreditScore

Figure 15-16 Web Service UDF wizard: Select operations

The next page allows to set a number of options for the user-defined functions which will be created. This page is displayed in Figure 15-17 on page 431 (it shows the wizard page, in case only one Web Service operation was selected; if UDFs are created for multiple operations, a list box contains all operations and allows you to switch between them).


Specify Options

Specify options for this user-defined function or accept the default settings.

WSDL Operation Name: getCreditScore

UDF Name: GetCurrCreditScore

UDF Comment: Retrieves the current credit score of a customer from a Credit CDE s

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 15-17 Web Service UDF wizard: General options page

The following options can be set:

- A name (and an optional comment) of the new wrapper UDF can be specified. Our example uses `GetCurrCreditScore` as the UDF name.
- The result of the wrapper UDF can either be a scalar value or a table. Depending on the result type of the Web Service operation one of these values has to be selected; you find more information about choosing the correct option in "Processing of Web Service results with different complexities" on page 434. Since our Web Service returns a numeric value, we select a scalar return value.
- If the result value is a table, the input parameters of the UDF can be added as columns to this table optionally.
- Optionally, the UDF can be created in a way to allow dynamic access to the service, by adding an input parameter for the service endpoint URL (for details read "Late binding of Web Service end points" on page 434).
- If the size of the SOAP response message (including the SOAP envelope) is less than 3,000 characters, the overloaded `DB2XML.SOAPHTTP()` methods which return a `VARCHAR` value can be used to call the Web Service. Otherwise, the methods returning a `CLOB` value are used. If you're not sure about the maximum size of the SOAP response, choose the second option.
- The last check box allows you to select if the SOAP response message shall be parsed by the UDF, or if the complete SOAP envelope has to be returned. Usually you will want to have the SOAP response parsed.

Moving to the next page you see the body of the wrapper UDF to review your selections. Select **Finish** to complete the action.

The new UDF can be found in the Data Definition view. To see this view, switch to the Data perspective (**Window** → **Open Perspective** → **Other** → **Data**). Do create the UDF in the PORTALDB database, you have to use the context menu and select **Build**. You can test the Web Service call using the **Run** command found in the context menu.

Note: The Web Service wizard tries to parse the SOAP result of the Web Service and convert the return values to appropriate SQL types. This action fails sometimes (either the structure of the response is not recognized, or the wrong SQL types are mapped), so a manual modification of the UDF may be required. You can find additional information about this matter in 15.2.3, “Best practices” on page 434.

Create the local cache table

We use a table in the PORTALDB database to cache credit score ratings of customers. This table needs to contain the customer’s social security number, the credit score which was retrieved last time as well as the time of the last update. Table 15-2 contains the layout of this new table.

Table 15-2 FICO.CREDIT_SCORE_CACHE database table layout

Column	Type	Notes	Description
SSN	CHAR(11)	NOT NULL, PRIMARY KEY	Customer’s Social Security Number.
SCORE	INTEGER	NOT NULL	Last retrieved credit score.
UPDATE_TMSP	TIMESTAMP	NOT NULL	Timestamp of last score update.

You can either use the DB2 Control Center or the RAD Table Definition wizard to create the table in the PORTALDB database wizard-based, or execute the SQL statements shown in Example 15-8 against the PORTALDB database in the DB2 Command Center.

Example 15-8 FICO.CREDIT_SCORE_CACHE database creation statements

```
CREATE TABLE FICO.CREDIT_SCORE_CACHE(
    SSN CHARACTER (11)    NOT NULL PRIMARY KEY,
    SCORE INTEGER         NOT NULL,
    UPDATE_TMSP TIMESTAMP NOT NULL WITH DEFAULT CURRENT TIMESTAMP);

COMMENT ON TABLE FICO.CREDIT_SCORE_CACHE IS
    'Caches the credit scores of customers.';

COMMENT ON FICO.CREDIT_SCORE_CACHE(
    SSN IS 'Customer''s Social Security Number',
    SCORE IS 'Last retrieved credit score.',
    UPDATE_TMSP IS 'Timestamp of last score update.');

GRANT SELECT, UPDATE, INSERT, DELETE
    ON TABLE FICO.CREDIT_SCORE_CACHE TO USER PORTALUSER;
```

Create the GetCreditScore() stored procedure

The `GetCreditScore()` procedure has to perform the following actions:

- Check if the credit score of a given customer does exist in the local cache.
- If not, call the Web Service wrapper UDF to retrieve the most recent credit score for the customer.
- Update the local cache with the new credit score value.

To create the stored procedure using RAD, select **File** → **New** → **Other**, click **Show all Wizards.** and select the SQL Stored Procedure wizard from the Data category. The wizard lets you specify the database schema and procedure name, the procedure parameters and result type, and you can also compose SQL statements which are executed when the procedure is called. Your procedure contains SQL statements as well as a control flow, so the code is added manually later on.

The wizard finishes with creating the procedure skeleton which can be modified. Example 15-9 contains the complete stored procedure, so overwrite the code skeleton with this code. To actually create the stored procedure in the PORTALDB database, select the stored procedure in the Data Definition view and select **Build** from the context menu.

Example 15-9 Code of GetCreditScore() stored procedure

```
CREATE PROCEDURE FICO.GetCreditScore(
        IN custSSN CHARACTER(11),
        IN custLastName VARCHAR(50),
        OUT newScore INTEGER)
P1: BEGIN ATOMIC

  -- Requestor credentials for Credit CDE Web Service are defined as
  -- constants; optionally store them in a database table
  DECLARE reqUser     VARCHAR(100) DEFAULT 'itsousr';
  DECLARE reqPassword VARCHAR(100) DEFAULT 'password';

  DECLARE tmp INTEGER;
  DECLARE noRecord INTEGER DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET noRecord = 1;

  -- check if a valid value of the customer’s credit score does exist in the
  -- local cache (credit scores expire after one month)
  SELECT SCORE INTO newScore
    FROM FICO.CREDIT_SCORE_CACHE
   WHERE SSN = custSSN
    AND UPDATE_TMSP > CURRENT TIMESTAMP - 1 MONTH;

  IF noRecord = 1 THEN
    -- retrieve credit score from Web Service if no cached value does exist
    SELECT FICO.GetCurrCreditScore(reqUser, reqPassword, custSSN, custLastName)
      INTO newScore
      FROM SYSIBM.SYSDUMMY1;

    SET noRecord = 0;

    SELECT 1 INTO tmp
      FROM FICO.CREDIT_SCORE_CACHE
     WHERE SSN = custSSN;

    -- update the local cache with the new score value
    IF noRecord = 1 THEN
      INSERT INTO FICO.CREDIT_SCORE_CACHE(SSN, SCORE) VALUES (custSSN, newScore);
    ELSE
      UPDATE FICO.CREDIT_SCORE_CACHE
         SET SCORE = newScore, UPDATE_TMSP = CURRENT TIMESTAMP
       WHERE SSN = custSSN;
    END IF;
  END IF;
END P1
```

15.2.3 Best practices

Late binding of Web Service end points

The `DB2XML.SOAPHTTPx()` functions require the following arguments:

- The service endpoint URL
- The body of the SOAP request
- A SOAP action URI reference (may be empty in most cases)

Whereas the action URI and the structure of the SOAP body will not change unless the called Web Service is replaced by a new version, the service endpoint URL may change (at example, if the server providing the Web Service moves, or, more likely, if the Web Service has to be called in different development, test, and production environments). In such cases it is preferable if changing the DB2 wrapper UDF can be avoided. An approach to keep the service endpoint URL more flexible is to provide it as a parameter of the wrapper UDF. This approach is called *late binding*, because the service endpoint URL is not defined until the wrapper UDF is actually called. Service endpoint URLs can be part of the application configuration, and stored either in configuration files, application deployment descriptors, or in the database.

The Web Service User-Defined Function wizard in RAD supports late binding when creating a DB2 wrapper UDF. The wizard option *Create a UDF with dynamic accesses to the service (late binding)* must be checked.

Treatment of complex input parameters (arrays, structured objects)

The input parameters of Web Services can have any complexity, from no parameters at all to a single parameter with a simple data type, to complex object trees or even large XML documents.

The RAD Web Service User-Defined Function wizard supports simple attributes (including bean-based parameter types; for example, if a Java-based Web Service interface specifies a Java Bean with a number of attributes as input parameter, the RAD wizard regards each bean attribute as separate bean parameter). Web Services with more complex input parameters can also be called, but in such cases the body of the SOAP request needs to be created manually within the wrapper UDF body. In order to do so, you need to be familiar with the structure of WSDL files.

Processing of Web Service results with different complexities

The result of a Web service is either a scalar value (for example, a numeric result like our credit score, or a character string), a simple object (*cmp.* a Java bean with a number of attributes), or a hierarchical structure (an object tree, or an complex XML document). The DB2 SOAP UDFs do not perform any special processing or conversion of the SOAP body returned by the Web Service, so it is your responsibility to treat the result.

You learned above that the most convenient way of calling Web Services within DB2 is to enclose the calls into wrapper user-defined functions. It makes sense to convert the result of the Web service call to a data structure which can be handled by DB2 applications.

When dealing with different types of results, the following conversion strategies should be followed:

- In case the Web Service result is a scalar value, convert it to the appropriate scalar SQL type and return it in the wrapper UDF.

- If a simple object (or an array of simple objects), or an array of objects is returned by the Web Service, convert it to a table.
- For a more complex result, a conversion may not be appropriate. Return the XML fragment instead.

Returning a scalar value

Our credit score Web Service is an example for a service with a scalar value. The SOAP result returned by the `DB2XML.SOAPHTTPV(...)` function looks similar to the text shown in Example 15-10 (the text in **bold** shows the actual result value).

Example 15-10 SOAP result with scalar value

```
<ns1:getCreditScoreResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="http://services.creditcde.com"
    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">
  <getCreditScoreReturn xsi:type="xsd:int">743</getCreditScoreReturn>
</ns1:getCreditScoreResponse>
```

In this case, the most appropriate return value of the wrapper UDF has an `INTEGER` type, and the conversion happens by calling DB2 XML Extender functions:

```
RETURN DB2XML.EXTRACTINTEGER(DB2XML.XMLCLOB(DB2XML.SOAPHTTPV(...)), '//*')
```

Returning a table

If a Web Service returns a number of attributes (pertaining to a specific object instance), or an array of attributes, the return value of the wrapper UDF is most likely a table. Each returned attributes is put into a separate column of the table, and if an array is returned, each element of the array is put into a separate table row. The caller of the wrapper UDF can access the result, for example, using an SQL `SELECT statement` and a cursor.

This approach works fine if number and types of the returned attributes are static. (In case the number of returned attributes is dynamic, and no array is returned, then each attribute can be put into a separate table row.)

Example 15-11 shows the result of a Web Service which returns the employee record for a given employee. The lines in **bold** contain the attribute values (only a few attributes are actually shown).

Example 15-11 SOAP result with some attributes

```
<ns1:getEmployeeRecordResponse
    xmlns:ns1="http://www.itso.com"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <getEmployeeRecordReturn>
    <employeeId>156</employeeId>
    <firstname>Michael</firstname>
    <lastname>Liberman</lastname>
    [...]
  </getEmployeeRecordReturn>
</ns1:getEmployeeRecordResponse>
```

In this case the skeleton of the wrapper UDF (the parameters of the SOAP call and the additionally returned attributes are not listed to make the point clear) looks like Example 15-12.

Example 15-12 Wrapper UDF

```
CREATE FUNCTION GET_EMPLOYEE_RECORD([...])
    RETURNS TABLE (
        EMPLOYEE_ID INTEGER,
        FIRSTNAME   VARCHAR(50),
        LASTNAME    VARCHAR(50))
    LANGUAGE SQL CONTAINS SQL
    EXTERNAL ACTION NOT DETERMINISTIC
    RETURN
        WITH SOAP_OUTPUT(OUT) AS
            (VALUES DB2XML.XMLCLOB(DB2XML.SOAPHTTPV([...])))
        SELECT
            DB2XML.EXTRACTINTEGER(OUTPUT.OUT, '//employeeId'),
            DB2XML.EXTRACTVARCHAR(OUTPUT.OUT, '//firstname'),
            DB2XML.EXTRACTVARCHAR(OUTPUT.OUT, '//lastname')
        FROM
            SOAP_OUTPUT OUTPUT;
```

Note: When performing a conversion of the result values, take care about the limitations of the converted attributes. At example, the WSDL does not specify the maximum length of strings (like the employee's first name and last name), so this additional information needs to be requested from the Web Service provider. Otherwise actual results may exceed limits of the SQL types defined in the wrapper UDF, leading to runtime errors.

Returning an XML document

There are several cases when it is adequate to return the generated XML result instead of converting it to a table or to a scalar SQL type. The result may contain an object tree which is hard to impossible to fit into a relational structure, or the number of returned objects and attributes is so high that it is too tedious to define a conversion function for each of them, or the further processing of the result is done using XML technologies like XSLT.

Returning an XML document means to strip the SOAP body text from the result and return the remaining XML document as a scalar text type (a `VARCHAR`, `LONG VARCHAR`, or `CLOB`, depending on the expected size of the result).

Example 15-13 shows the result of a Web Service call which receives the aggregated marketing reports of departments of a company. The results are returned as an XML document and will be further transformed into a printable HTML format.

Example 15-13 SOAP result with complex hierarchy

```
<ns1:getMarketingReportResponse
    xmlns:ns1="http://www.itso.com"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <getMarketingReportReturn>
    <department name="deptA">
      <account id="">
        <results quarter="01-2006">
```

```
          [...]
        </results>
        <results quarter="02-2006">
          [...]
        </results>
        [...]
      </account>
      <account id="acct2">
        [...]
      </account>
    </department>
    <department name="deptB">
      [...]
    </department>
    [...]
  </getMarketingReportReturn>
</ns1:getMarketingReportResponse>
```

The wrapper UDF extracts the XML document from the SOAP as a scalar CLOB value which is returned:

```
   RETURN DB2XML.EXTRACTCLOB(
       DB2XML.XMLCLOB(DB2XML.SOAPHTTPC(...)), '/getMarketingReportResponse')
```

Implications of RPC/encoded results

Most existing SOAP-based Web Services use either the RPC/encoded or document/literal style to exchange SOAP messages. Those styles differ only in some details, and application developers usually do not need to care about the style used by a Web Service since the SOAP layer provides means to encode and decode the SOAP messages.

For Web Services which are consumed using the DB2 SOAP functions, this aspect is important though. Whereas the conversion of SOAP responses which use the document/literal style is straightforward, since their XML structure is hierarchical and follows a given XML Schema document, RPC/encoded SOAP responses may use a special feature called *multiRef* elements. In this case, the message is split into several sections, with *href* attributes linking the different sections.

Example 15-14 displays the RPC/encoded result of a Web Service returning an employee record (the same example was already presented in Example 15-11 on page 435, that time using document/literal style). Since the result is split into several sections, it is not feasible to extract the attributes using the functions provided by the DB2 XML Extender, especially since the actual usage of *multiRef* elements is arbitrary and does not follow specific rules.

Example 15-14 SOAP result in RPC/encoded style, using <multiRef> elements

```
<ns1:getEmployeeRecordResponse
    xmlns:ns1="http://www.itso.com"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <getEmployeeRecordReturn href="#id0"></getEmployeeRecordReturn>
</ns1:getEmployeeRecordResponse>
<multiRef id="id0" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xsi:type="ns2:ResultBean"
    xmlns:ns2="http://www.itso.com"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <employeeId xsi:type="xsd:int">156</employeeId>
  <firstname href="#id1"></firstname>
  <lastname xsi:type="xsd:string">Liberman</lastname>
</multiRef>
<multiRef id="id1" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xsi:type="xsd:string"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  Michael
</multiRef>
```

As a rule of thumb, with Web Services returning a single scalar value, do not use *multiRef* elements; but since this is dependent on the SOAP engine used by the provider of the Web Service, no guarantee can be given.

Tip: If a consumed Web Service uses RPC/encode style, test the SOAP response for the occurrence of `<multiRef>` elements. Avoid processing the Web Service with the DB2 SOAP functions in that case.

Handling of exceptional conditions

Calls of the DB2 Web Services consumers UDFs can terminate unexpectedly because of one of the following reasons:

- The called Web Service throws an exception while processing the incoming request. This behavior can be caused, for example, by input parameters which are outside of the allowed range, or by an internal error state in the Web Service application.
- A problem in the SOAP layer of the DB2 user-defined functions is identified. Examples are invalid SOAP responses, wrong service endpoints URLs, connection problems, and so on.

In case the functions of the `DB2XML.SOAPHTTPx()` UDF family encounter such a problem, they return an SQL exception, setting the SQL code to SQL0443 and an SQL state which helps to further identify the actual failure. A list of possible SQL state values is given at 6.8.2, “SOAPHTTPC and SOAPHTTPV” on page 164. Here are some common examples:

Web Service application unavailable

```
SQL0443N  Routine "DB2XML.SOAPHTTPCL" (specific name "SOAPHTTPVICLO") has
returned an error SQLSTATE with diagnostic text "Error during socket connect".
SQLSTATE=38309
```

The SOAP layer cannot open an HTTP connection to the Web Service provider. Make sure the Web Service application is running.

Problem establishing SSL connections

```
SQL0443N  Routine "DB2XML.SOAPHTTPCL" (specific name "SOAPHTTPVICLO") has
```

```
returned an error SQLSTATE with diagnostic text "SSL API error".
SQLSTATE=38328
```

Web Services which transmit confidential data are usually secured using the SSL transport protocol. SSL requires that the server running the service provides an electronic certificate which was requested by an official certificate authority. In some cases servers only have test certificates (or so called self-certified certificates) installed, especially if these servers are only used for development or testing purposes. If that's the case, the SOAP UDFs raise the error shown above. Corrective actions for such cases are to install a valid certificate on the server, or avoid to use SSL (not recommended for privacy and security reasons).

Application throws exception

```
SQL0443N  Routine "DB2XML.SOAPHTTPCL" (specific name "SOAPHTTPVICLO") has
returned an error SQLSTATE with diagnostic text
"org.itso.LanguageNotSupportedException: Please specify a different language
than 'en_XX'".  SQLSTATE=38327
```

Application-specific exceptions are generally mapped to the SQL state 38327. More detailed information about the cause of the exception can be found in the diagnostic text.

In general it is recommended that applications and systems using the DB2 Web Service consumer UDFs implement a strategy to handle exceptions thrown by these UDFs. The Web Service concept is based on the principle of loose coupling, with the effect that service endpoints may not be available all the time. Appropriate handlers can be put into the wrapper UDFs, or in application code calling these UDFs.

15.2.4 Considerations using DB2 Version 9.1

The differences between DB2 Version 8 and Version 9.1 regarding the provided Web Service consumer UDFs are minor. The main difference is that Version 9.1 of the DB2 database replaces the DB2 XML Extender component with a native XML engine. The XML Extender functions are still supported, but its functionality should not be used anymore for new development.

The `DB2XML.SOAPHTTP()` UDFs which are logically bundled with the XML Extender (indicated by the schema name `DB2XML` they share with other XML Extender functions) are not replaced by another component. When these UDFs are installed in a DB2 version 8 database, XML Extender is a prerequisite; in a DB2 Version 9.1 database they can be created without having the XML Extender component created.

Our scenario shows only one dependency on the XML Extender: The SOAP body of the Web Service response is parsed using XML Extender functions within the Web Service wrapper UDFs. These functions need to be replaced by the XML functions which are available in DB2 Version 9.1. Find a list of these new XML functions in Appendix A.7, "SQL/XML" on page 571.

At example, here's a fragment of a wrapper UDF which was used before to show the parsing of SOAP responses with a number of attribute results, using XML Extender functions (marked in **bold**) to parse the result:

```
WITH SOAP_OUTPUT(OUT) AS
    (VALUES DB2XML.XMLCLOB(DB2XML.SOAPHTTPV([...])))
SELECT
    DB2XML.EXTRACTINTEGER(OUTPUT.OUT, '//employeeId'),
    DB2XML.EXTRACTVARCHAR(OUTPUT.OUT, '//firstname'),
    DB2XML.EXTRACTVARCHAR(OUTPUT.OUT, '//lastname')
FROM
```

```
        SOAP_OUTPUT OUTPUT;
```

The equivalent expression, using native DB2 XML functions, is:

```
WITH SOAP_OUTPUT(OUT) AS
    (VALUES XMLPARSE(DOCUMENT B2XML.SOAPHTTPV([...])))
SELECT
    INTEGER(CAST(XML2CLOB(XMLQUERY(
      '$x//employeeId/text()' PASSING OUTPUT.OUT AS "x")) AS CHAR(20))),
    CAST(XML2CLOB(XMLQUERY(
      '$x//firstname/text()' PASSING OUTPUT.OUT AS "x")) AS VARCHAR(50)),
    CAST(XML2CLOB(XMLQUERY(
      '$x//lastname/text()' PASSING OUTPUT.OUT AS "x")) AS VARCHAR(50))
FROM
    SOAP_OUTPUT OUTPUT;
```

Note: Some of the XML functions of DB2 Version 9.1 do already exist in DB2 version 8.2 or earlier (for example, the XMLQUERY function). The syntax of the functions differs slightly which needs to be considered when writing code which has to run under different DB2 versions.

15.2.5 DB2 SOAP functions and Web Service interoperability

Apart from simple SOAP messaging, Web Service implementation may use additional standards and techniques which enhance the functionality of those services, but on the other hand restrict interoperability between different systems.

The DB2 SOAP consumer UDFs have a number of characteristics which potentially cause problems, restricting their usage when consuming Web Services of different service providers:

- The SOAP engine used by the UDFs supports only basic SOAP messaging (for example, the features of the WS-I Attachments profile cannot be leveraged).
- The SOAP UDF interface leaves much of the implementation effort to the caller. When you use these functions, you need to compose the SOAP request body and parse the SOAP response. One of the disadvantages caused by this fact is, that an already mentioned RPC/encoded response cannot be parsed in a straightforward way (although the SOAP engine might provide means to parse the response).
- On the other hand, the SOAP functions do not allow to customize the SOAP interaction to include additional information which is required by other Web Services standards or by the used HTTP protocol. At example, if a Web Service is secured by HTTP Basic Authentication, the user credentials cannot be included in the HTTP header of the SOAP request.

These restrictions have a considerate impact on the usage of the SOAP UDFs in an enterprise environment, especially when the following standards have to be met:

- WS-Security
- WS-AtomicTransaction
- WS-ReliableMessaging
- HTTP Basic Authentication
- SSL/HTTPS (certificate issues, client certificates not supported)
- Web Service DIME Attachments
- RPC/encoded SOAP messages (parsing issues)

This list of Web Service standards is not complete, and some of these standards are not of widespread use yet (especially standards covering reliable or transactional messaging). The technologies SSL, HTTP Basic Authentication, or RPC/encoded style though are often used in an enterprise Web Services environment.

The missing support of these standards needs to be considered when assessing the use of the DB2 SOAP UDFs in a specific application scenario. If that's not possible, alternatives like the aggregation of Web Services outside the DB2 environment have to be used.

15.2.6 Consuming Web Services using Information Integrator

The WebSphere Information Integrator product (which was recently renamed from DB2 Information Integrator) also supports the consumption of Web Services within DB2 via Web Service wrappers.

Example 15-15 shows the statement to create a Web Service based data source using the Information Integrator. The Web Service returns the current temperature for a given ZIP code which is described by the ZIPCODE column (it's a required input column because of the nickname TEMPLATE syntax). The output value is described by the RETURN column. You create an input column for each value in the input message of a Web service operation and an output column for each return value of a Web service operation. You control the input and output column definitions with the nickname column option definitions.

Example 15-15 Information Integrator Web Service data source

```
CREATE NICKNAME GETTEMP(
    ZIPCODE VARCHAR (10) OPTIONS(TEMPLATE '&column'),
    RETURN  VARCHAR (10) OPTIONS(XPATH './return/text()'))
  FOR SERVER "XMETHWEB"
  OPTIONS(
    URL 'http://services.xmethods.net/soap/servlet/rpcrouter',
    SOAPACTION ' ' ,
    TEMPLATE '<soapenv:Envelope>
                <soapenv:Body>
                  <ns2:getTemp>
                    <zipcode> &zipcode[1,1] </zipcode>
                  </ns2:getTemp>
                </soapenv:Body>
              </soapenv:Envelope>',
    XPATH '/soapenv:Envelope/soapenv:Body/*' ,
    NAMESPACES ' ns1="http://www.xmethods.net/sd/TemperatureService.wsdl",
                 ns2="urn:xmethods-Temperature" ,
                 soapenv="http://schemas.xmlsoap.org/soap/envelope/"');
```

The Web services wrapper nickname options URL and SOAPACTION provide the ability to override the endpoint, or the address that you specified when you created the nickname (which is a more flexible approach for late binding than the one shown for the DB2 Web Service wrapper UDFs). The value for the SOAPACTION nickname option becomes an attribute in the HTTP header. The value for the URL nickname option is the HTTP URL to which the request is sent.

You can retrieve the current temperature for ZIP code 95119 with the following SQL SELECT statement:

```
SELECT * FROM GETTEMP
 WHERE ZIPCODE = '95119';
```

The product page of the WebSphere Information Integrator can be found at:

`http://www.ibm.com/software/data/integration/db2ii/`

Additional information concerning the usage of the Information Integrator Web Service wrapper and examples of data sources are shown at:

`http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp`

15.3 Scenario exposing I4GL business logic as Web services

This scenario shows how an existing Informix 4GL application can be easily integrated into a Web services framework through conversion into EGL.

15.3.1 Overview

During 2005 the ITSO bank was trying to diversify its portfolio and bought a smaller bank that was focussed on personal lines of credits, issuing credit cards. This company (the Informix Bank, another fictional name) uses Informix as a database and Informix 4GL applications to manage credit card accounts.

There are existing 4GL applications to list accounts (credit card numbers) for a customer number and get detailed account info for a credit card number. The customer record that existed in Informix was merged into the IMS-based customer record.

Before we go ahead with the actual conversion and integration steps, let us look at the existing Informix 4GL application at the time the Informix bank got acquired.

15.3.2 The Informix Bank 4GL credit card application

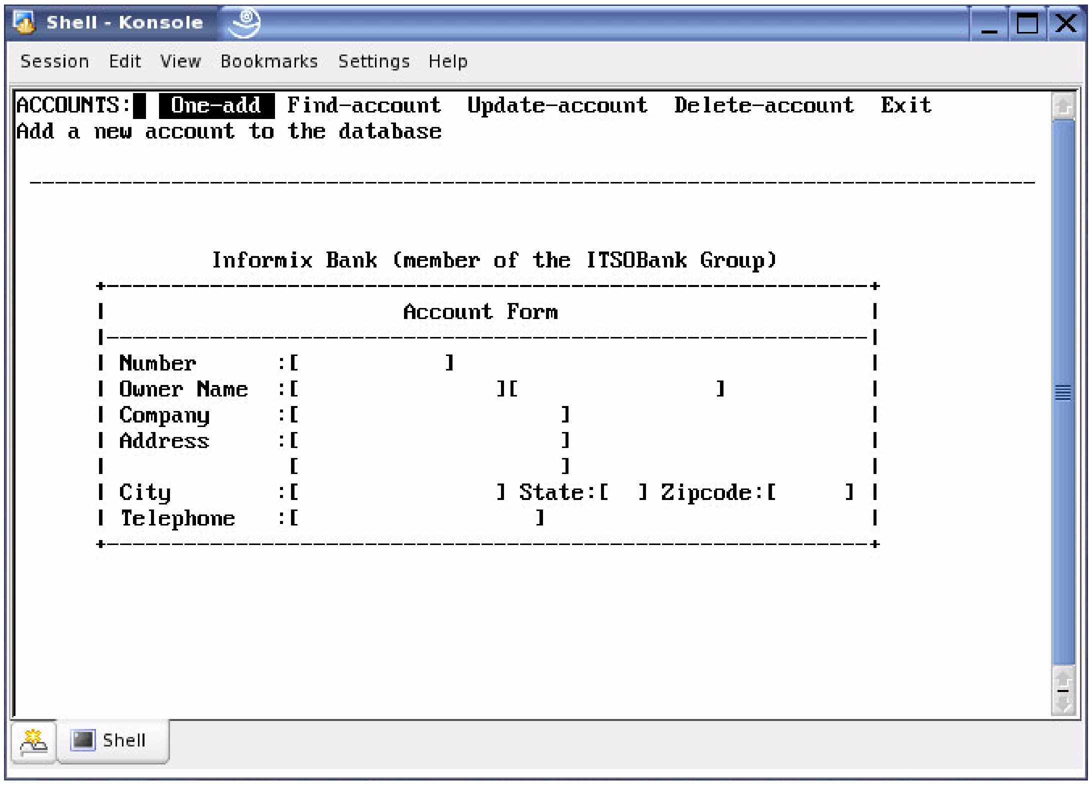
The original 4GL credit card application allows a simple management of customer account details and related credit card entries per customer account.

The application itself consists out of the following I4GL modules:

- ifxbank_main.4gl
- ifxbank_accounts.4gl
- ifxbank_cards.4gl
- ifxbank_lib.4gl
- ifxbank_globals.4gl
- accountform.per
- cardform.per
- state_list.per

So it basically has one 4GL library (ifxbank_lib.4gl) which contains some shared functions, the actual application modules plus two I4GL forms description files.

Let us briefly look at the application’s functionality through images from some of the applications menus and form screens.



```
ACCOUNTS:  One-add  Find-account  Update-account  Delete-account  Exit
Add a new account to the database

 ------------------------------------------------------------------------------


               Informix Bank (member of the ITSOBank Group)
      +------------------------------------------------------------+
      |                        Account Form                        |
      |------------------------------------------------------------|
      | Number      :[           ]                                 |
      | Owner Name  :[               ][                ]           |
      | Company     :[                     ]                       |
      | Address     :[                     ]                       |
      |              [                     ]                       |
      | City        :[               ] State:[  ] Zipcode:[      ] |
      | Telephone   :[                   ]                         |
      +------------------------------------------------------------+
```

Figure 15-18   Informix Bank: The Account maintenance form and menu

The application has two major functions: account maintenance and credit card handling. In the account section, one can create new accounts, update accounts, lookup accounts and delete accounts.

The credit card section allows the addition of new credit cards to an existing account, to list all existing credit cards for a given account and to do credit card maintenance (in our example code, not all features are completely implemented.

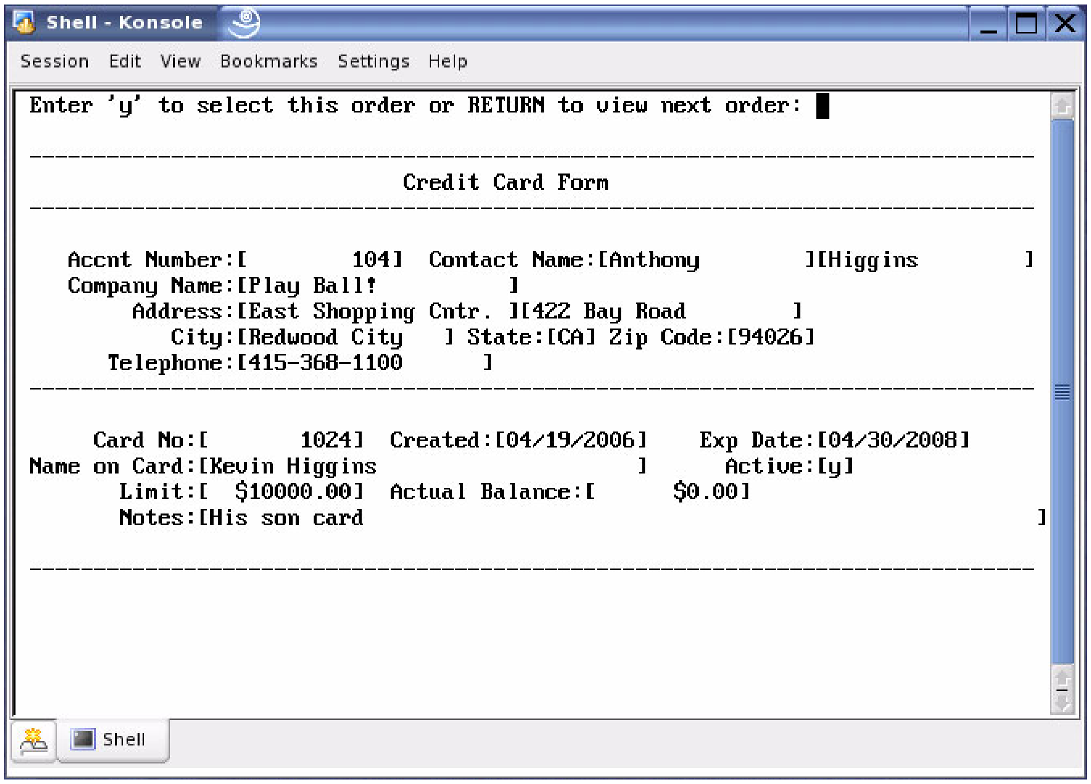
After the acquisition of Informix Bank through the ITSO Bank, the Informix Bank had to keep the existing system up and running in order to be able to still maintain its existing customer base. At the same the ITSO Bank already had the requirement to be able to access the credit card information in the Informix Bank system for it own account validations and credit ratings.

For this purpose the Informix Bank had to provide a Web service which lists all the existing credit cards for an account number.

Fortunately Informix Bank already implemented a dedicated I4GL function in one of their I4GL libraries which allows the retrieval of existing credit card information for an existing account.

To easily make this I4GL function accessible as a Web service to ITSO Bank, Informix Bank decided to convert its existing I4GL application to EGL and then utilize the EGL Web services support to expose the required function to ITSO Bank.

See the credit card form in Figure 15-19 on page 444.



```
Enter 'y' to select this order or RETURN to view next order: 

------------------------------------------------------------------------------
                              Credit Card Form
------------------------------------------------------------------------------

   Accnt Number:[        104]  Contact Name:[Anthony        ][Higgins        ]
   Company Name:[Play Ball!          ]
        Address:[East Shopping Cntr. ][422 Bay Road         ]
           City:[Redwood City     ] State:[CA] Zip Code:[94026]
      Telephone:[415-368-1100      ]
------------------------------------------------------------------------------

     Card No:[       1024]  Created:[04/19/2006]    Exp Date:[04/30/2008]
Name on Card:[Kevin Higgins                    ]      Active:[y]
       Limit:[  $10000.00]  Actual Balance:[      $0.00]
       Notes:[His son card                                                  ]

------------------------------------------------------------------------------
```

Figure 15-19 Informix Bank: The credit card form while browsing an accounts credit cards

Before we continue with the next steps towards the Web services integration, let us look a the I4GL function(s) which should be eventually exposed as a Web service in Example 15-16.

Example 15-16 ifxbank_lib.4gl with the get_all_cards() and the get_one_card() functions

```
GLOBALS
   "ifxbank_globals.4gl"

DEFINE
        a_cards ARRAY[20] OF RECORD
           card_num LIKE cards.card_num,
           create_date LIKE cards.create_date,
           name_on_card LIKE cards.name_on_card,
           is_active LIKE cards.is_active,
           exp_date LIKE cards.exp_date,
           balance LIKE cards.balance,
           limit LIKE cards.limit,
           notes LIKE cards.notes
      END RECORD

FUNCTION get_all_cards(a_num)
        DEFINE a_num LIKE accounts.account_num,
                idx INTEGER

        DECLARE all_cards_list CURSOR FOR
        SELECT
                card_num, create_date, name_on_card, is_active,
                exp_date, balance, limit, notes
```

```
            FROM cards
            WHERE account_num = a_num
        LET idx = 1
        FOREACH all_cards_list INTO
                        a_cards[idx].card_num, a_cards[idx].create_date,
                        a_cards[idx].name_on_card, a_cards[idx].is_active,
                        a_cards[idx].exp_date, a_cards[idx].balance,
                        a_cards[idx].limit, a_cards[idx].notes
                LET idx = idx + 1
                IF idx > 20 THEN
                        EXIT FOREACH
                END IF
        END FOREACH
        LET idx = idx - 1
        RETURN idx
END FUNCTION

FUNCTION get_one_card(idx)
        DEFINE idx INTEGER,
        t_card RECORD
      card_num LIKE cards.card_num,
      create_date LIKE cards.create_date,
      name_on_card LIKE cards.name_on_card,
      is_active LIKE cards.is_active,
      exp_date LIKE cards.exp_date,
      balance LIKE cards.balance,
      limit LIKE cards.limit,
      notes LIKE cards.notes
   END RECORD

   LET t_card.card_num = a_cards[idx].card_num
   LET t_card.name_on_card = a_cards[idx].name_on_card
   LET t_card.notes = a_cards[idx].notes
   LET t_card.limit = a_cards[idx].limit
   LET t_card.balance = a_cards[idx].balance
   LET t_card.is_active = a_cards[idx].is_active
   LET t_card.create_date = a_cards[idx].create_date
   LET t_card.exp_date = a_cards[idx].exp_date

   RETURN t_card.*

END FUNCTION
```

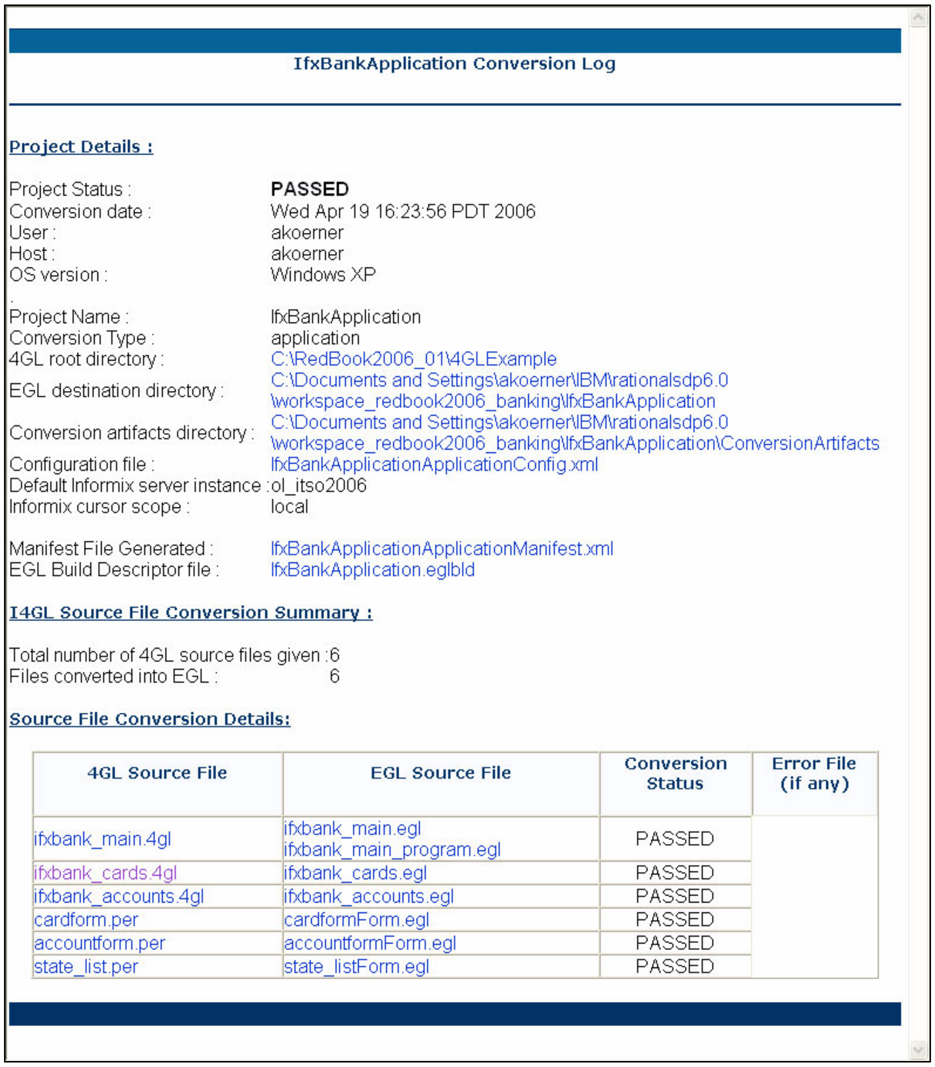
15.3.3 4GL to EGL conversion of the Informix Bank application

To successfully convert the I4GL banking application to EGL we just need to follow part a) of the steps outlined in 9.6.2, “What the required steps are” on page 346.

After starting the Rational Development environment (in our example Rational Application Developer or RAD), we are creating a new workspace and enable that workspace for EGL.

In the next step we need to create an EGL schema project which contains the correct mappings between the EGL record structures and the underlying database tables.

As soon we have the schema project created, we can convert the ifxbank_lib.4gl file and then finally the actual application modules. If the conversion process is successful, you should see a conversion log file like the one in Figure 15-20.



IfxBankApplication Conversion Log

Project Details :

Project Status :	**PASSED**
Conversion date :	Wed Apr 19 16:23:56 PDT 2006
User :	akoerner
Host :	akoerner
OS version :	Windows XP
Project Name :	IfxBankApplication
Conversion Type :	application
4GL root directory :	C:\RedBook2006_01\4GLExample
EGL destination directory :	C:\Documents and Settings\akoerner\IBM\rationalsdp6.0 \workspace_redbook2006_banking\IfxBankApplication
Conversion artifacts directory :	C:\Documents and Settings\akoerner\IBM\rationalsdp6.0 \workspace_redbook2006_banking\IfxBankApplication\ConversionArtifacts
Configuration file :	IfxBankApplicationApplicationConfig.xml
Default Informix server instance :	ol_itso2006
Informix cursor scope :	local
Manifest File Generated :	IfxBankApplicationApplicationManifest.xml
EGL Build Descriptor file :	IfxBankApplication.eglbld

I4GL Source File Conversion Summary :

Total number of 4GL source files given :	6
Files converted into EGL :	6

Source File Conversion Details:

4GL Source File	EGL Source File	Conversion Status	Error File (if any)
ifxbank_main.4gl	ifxbank_main.egl ifxbank_main_program.egl	PASSED	
ifxbank_cards.4gl	ifxbank_cards.egl	PASSED	
ifxbank_accounts.4gl	ifxbank_accounts.egl	PASSED	
cardform.per	cardformForm.egl	PASSED	
accountform.per	accountformForm.egl	PASSED	
state_list.per	state_listForm.egl	PASSED	

Figure 15-20 The 4GL to EGL conversion log file after converting the Informix Bank application

Let us look at the EGL equivalent to the ifxbank_lib.4gl file, now named the ifxbank_lib.egl file as part of the IfxBankLib project, shown in Example 15-17.

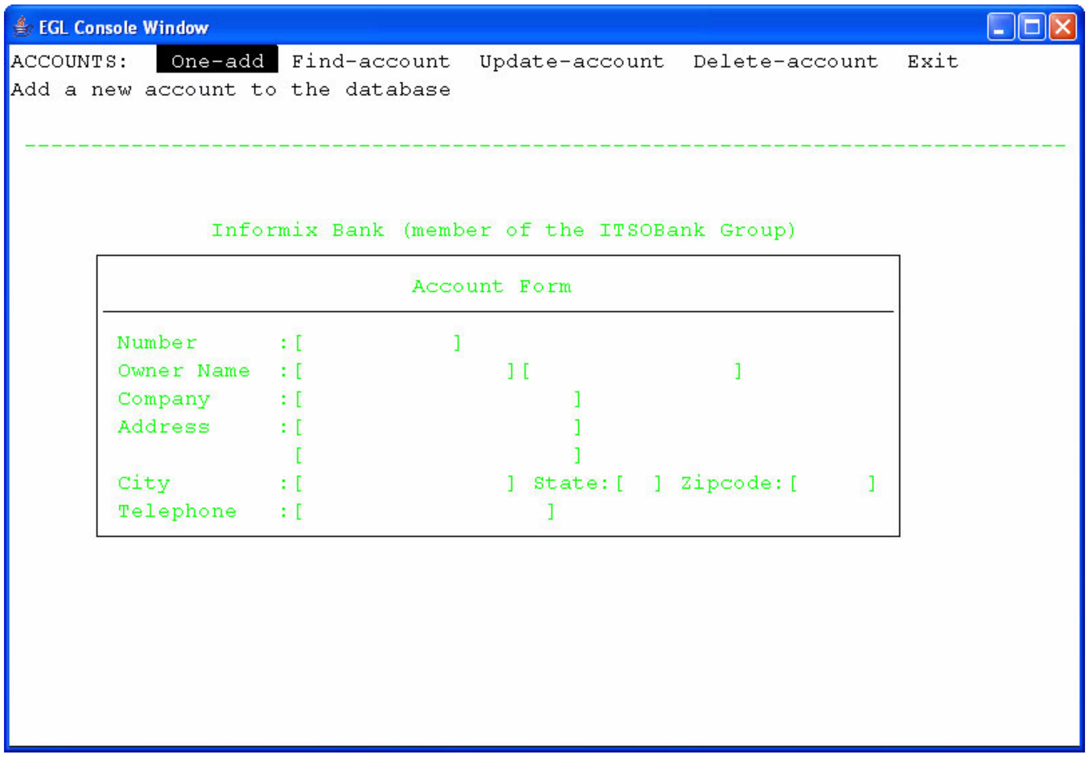
Example 15-17 The generated ifxbank_lib.egl file

```
/*
 -------------------------------------------------------------
 -  Informix 4GL to EGL Conversion Tool converted I4GL file
 -  Generated on : Wed Apr 19 16:15:38 PDT 2006
 -------------------------------------------------------------
*/

package IfxBankLib;

import ifxbanklib.*;
import ifxbankschema.ol_itso2006.ifxbank.*;
```

```
LIBRARY ifxbank_lib{localSQLScope=YES}

use IfxBankLib.ifxbank_globals;
use IfxBankLib.IfxBankLibConversionGlobals;

// globals "ifxbank_globals.4gl"

use IfxBankLib.ifxbank_libLibraryVariables;

FUNCTION get_all_cards(a_num dataitem_like_accounts_account_num IN)
returns (INT)
                idx  INT;

        PREPARE /*DECLARE all_cards_list*/ $_STMT_all_cards_list FROM
        "SELECT" +
   "                card_num, create_date, name_on_card, is_active," +
   "                exp_date, balance, limit, notes" +
   "            FROM cards" +
   "            WHERE account_num = ?";
        idx =  1;

        open all_cards_list with $_STMT_all_cards_list using a_num;
        Foreach (From  all_cards_list INTO
                        a_cards[idx].card_num, a_cards[idx].create_date,
                        a_cards[idx].name_on_card, a_cards[idx].is_active,
                        a_cards[idx].exp_date, a_cards[idx].balance,
                        a_cards[idx].limit, a_cards[idx].notes)
                idx = ( idx +  1);
                IF ( idx >  20)
                        EXIT FOREACH;
                END
        END
        idx = ( idx -  1);
        return(  idx );
END

FUNCTION get_one_card(idx INT IN,
   /*returning*/ $_retvar_1 int OUT, $_retvar_2 date OUT, $_retvar_3 unicode(60)
OUT, $_retvar_4 unicode(1) OUT, $_retvar_5 date OUT, $_retvar_6 money(8,2) OUT,
$_retvar_7 money(8,2) OUT, $_retvar_8 unicode(60) OUT)

        t_card  recordtype_ifxbank_lib_a_cards;

   t_card.card_num =  a_cards[idx].card_num;
   t_card.name_on_card =  a_cards[idx].name_on_card;
   t_card.notes =  a_cards[idx].notes;
   t_card.limit =  a_cards[idx].limit;
   t_card.balance =  a_cards[idx].balance;
   t_card.is_active =  a_cards[idx].is_active;
   t_card.create_date =  a_cards[idx].create_date;
   t_card.exp_date =  a_cards[idx].exp_date;

   $_retvar_1 = t_card.card_num;
   $_retvar_2 = t_card.create_date;
   $_retvar_3 = t_card.name_on_card;
   $_retvar_4 = t_card.is_active;
   $_retvar_5 = t_card.exp_date;
   $_retvar_6 = t_card.balance;
   $_retvar_7 = t_card.limit;
   $_retvar_8 = t_card.notes;
   return ;

END
END // LIBRARY

record recordtype_ifxbank_lib_a_cards type SqlRecord
      card_num
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_card_num{isNullable=yes};
      create_date
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_create_date{isNullable=yes};
      name_on_card
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_name_on_card{isNullable=yes}
;
      is_active
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_is_active{isNullable=yes};
      exp_date
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_exp_date{isNullable=yes};
      balance
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_balance{isNullable=yes};
      limit
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_limit{isNullable=yes};
      notes
IfxBankSchema.ol_itso2006.ifxbank.dataitem_like_cards_notes{isNullable=yes};
   END
```

To verify that the conversion has been successful, we do a test run of the converted application. You will notice that the application looks and behaves like the original I4GL application, but is now completely EGL based. In Figure 15-21 on page 449, the screen capture has been inverted for printing purposes.

After also validating the correctness of the converted library functions, we can now go ahead and expose the library functions as Web services.



```
ACCOUNTS:  One-add  Find-account  Update-account  Delete-account  Exit
Add a new account to the database
-----------------------------------------------------------------------------

              Informix Bank (member of the ITSOBank Group)

                         Account Form

       Number       :[           ]
       Owner Name   :[               ][               ]
       Company      :[                     ]
       Address      :[                     ]
                     [                     ]
       City         :[               ] State:[  ] Zipcode:[     ]
       Telephone    :[                   ]
```

Figure 15-21 The Informix Bank Account screen, after the conversion to EGL

15.3.4 Expose the converted I4GL library functions as an EGL Web service

In this final step we only need to create a new EGL Web project, include the converted IfxBankLib into the Web projects build path, write an EGL Web service wrapper and publish it to a Web application server.

Since we primarily re-use the converted I4GL code, we only need to create one new EGL part to define the EGL Web service. Let us call this file IfxBankService.egl. See Example 15-18.

Example 15-18 IfxBankService.egl

```
import IfxBankLib.ifxbank_lib;
import IfxBankLib.recordtype_ifxbank_lib_a_cards;

Service IfxBankService

   function get_all_creditcards(account_num int, all_cards
recordtype_ifxbank_lib_a_cards[]) returns(int)

      max, i int;
      one_card recordtype_ifxbank_lib_a_cards;

      max = ifxbank_lib.get_all_cards(account_num);

      if (max > 0)
         i = 1;
         while (i <= max)
            ifxbank_lib.get_one_card(i, one_card.card_num, one_card.create_date,
                        one_card.name_on_card, one_card.is_active,
                        one_card.exp_date, one_card.balance,
                        one_card.limit, one_card.notes);

            all_cards.appendElement(one_card);
            i = i + 1;
         end
      end
      return (max);
   end
end
```

In the IfxBankService.egl we are calling the two functions get_all_cards() and get_one_card() to obtain the requested credit card entries for a given account number.

As soon as we have saved the EGL services part above, RAD recognizes that we defined a new Web service and automatically generates the associated WSDL file for that Web service. This WSDL file (see also Example 15-19) can be now used to either develop client applications to access that new Web service or to simply test the Web service with the built-in Web Services Explorer.

Example 15-19 The IfxBankService.wsdl file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://services" xmlns:tns="http://services"
xmlns:tns1="http://ifxbanklib" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types>
     <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://services" xmlns:tns1="http://ifxbanklib"
xmlns:tns="http://services" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
         <import namespace="http://ifxbanklib"/>
         <complexType name="ArrayOfRecordtype_ifxbank_lib_a_cards">
           <sequence>
             <element name="ArrayOfRecordtype_ifxbank_lib_a_cards" minOccurs="0"
maxOccurs="unbounded" type="tns1:Recordtype_ifxbank_lib_a_cards"/>
           </sequence>
         </complexType>
         <element name="get_all_creditcards">
           <complexType>
             <sequence>
            <element name="account_num" nillable="false" type="xsd:int"/>
            <element name="all_cards" nillable="false"
type="tns:ArrayOfRecordtype_ifxbank_lib_a_cards"/>
             </sequence>
           </complexType>
         </element>
         <element name="get_all_creditcardsResponse">
           <complexType>
             <sequence>
            <element name="account_num" nillable="false" type="xsd:int"/>
            <element name="all_cards" nillable="false"
type="tns:ArrayOfRecordtype_ifxbank_lib_a_cards"/>
            <element name="return" nillable="false" type="xsd:int"/>
```

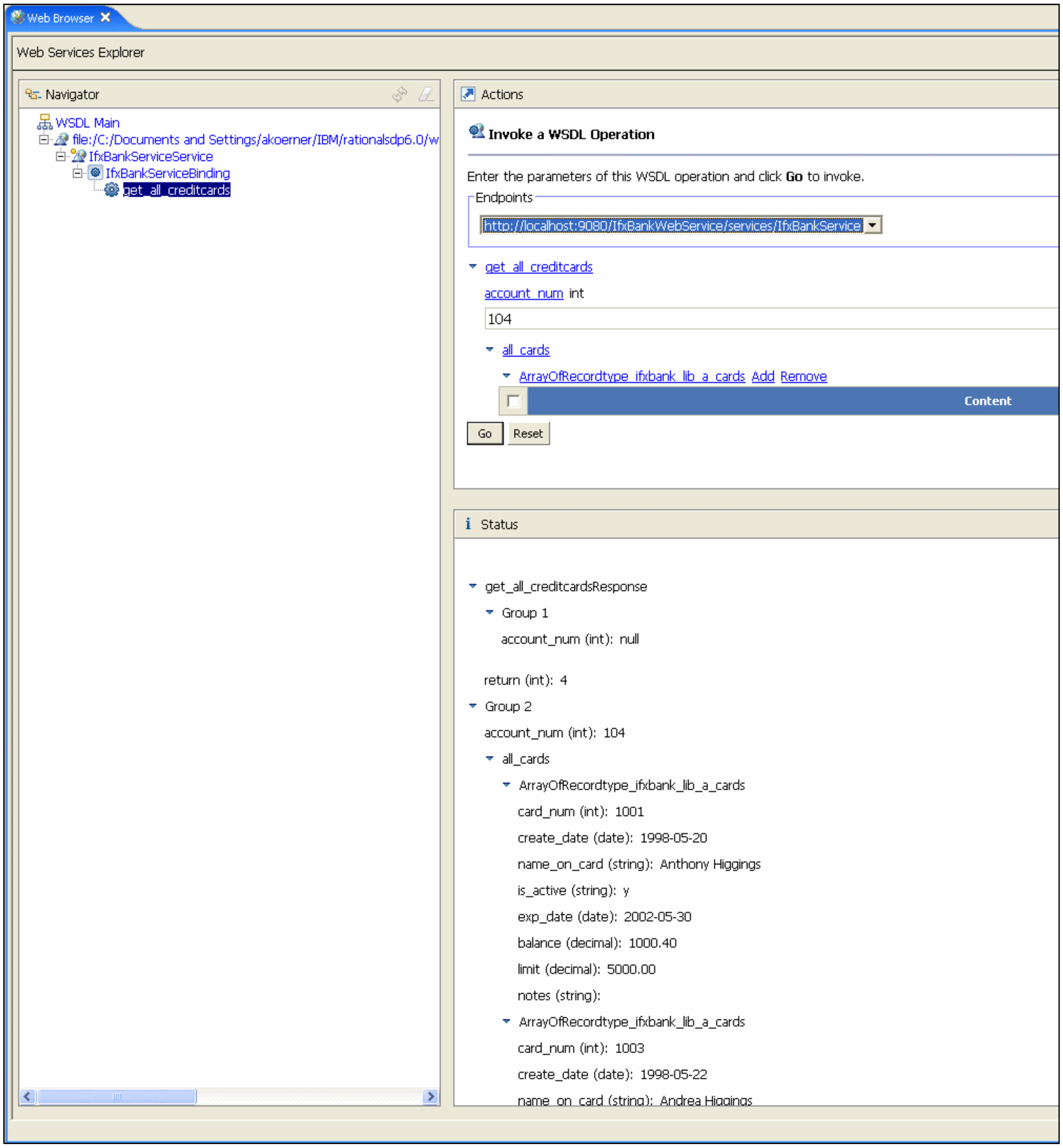
```
              </sequence>
            </complexType>
          </element>
       </schema>
       <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://ifxbanklib" xmlns:tns="http://services"
xmlns:tns1="http://ifxbanklib" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
          <import namespace="http://services"/>
          <simpleType name="ezeunicodeL60">
            <xsd:restriction base="xsd:string">
             <xsd:maxLength value="60"/>
            </xsd:restriction>
          </simpleType>
          <complexType name="Recordtype_ifxbank_lib_a_cards">
            <sequence>
             <element name="card_num" nillable="true" type="xsd:int"/>
             <element name="create_date" nillable="true" type="xsd:date"/>
             <element name="name_on_card" nillable="true"
type="tns1:ezeunicodeL60"/>
             <element name="is_active" nillable="true" type="tns1:ezeunicodeL1"/>
             <element name="exp_date" nillable="true" type="xsd:date"/>
             <element name="balance" nillable="true" type="tns1:ezemoneyL9D2"/>
             <element name="limit" nillable="true" type="tns1:ezemoneyL9D2"/>
             <element name="notes" nillable="true" type="tns1:ezeunicodeL60"/>
            </sequence>
          </complexType>
          <simpleType name="ezemoneyL9D2">
            <xsd:restriction base="xsd:decimal">
             <xsd:totalDigits value="9"/>
             <xsd:fractionDigits value="2"/>
            </xsd:restriction>
          </simpleType>
          <simpleType name="ezeunicodeL1">
            <xsd:restriction base="xsd:string">
             <xsd:maxLength value="1"/>
            </xsd:restriction>
          </simpleType>
       </schema>
    </wsdl:types>
    <wsdl:message name="get_all_creditcardsRequest">
       <wsdl:part name="parameters" element="tns:get_all_creditcards"/>
    </wsdl:message>
    <wsdl:message name="get_all_creditcardsResponse">
       <wsdl:part name="parameters" element="tns:get_all_creditcardsResponse"/>
    </wsdl:message>
    <wsdl:portType name="IfxBankService">
       <wsdl:operation name="get_all_creditcards">
         <wsdl:input name="get_all_creditcardsRequest"
message="tns:get_all_creditcardsRequest"/>
         <wsdl:output name="get_all_creditcardsResponse"
message="tns:get_all_creditcardsResponse"/>
       </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="IfxBankServiceBinding" type="tns:IfxBankService">
```

```
     <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="get_all_creditcards">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="get_all_creditcardsRequest">
          <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="get_all_creditcardsResponse">
          <wsdlsoap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="IfxBankServiceService">
     <wsdl:port name="IfxBankService" binding="tns:IfxBankServiceBinding">
       <wsdlsoap:address
location="http://localhost:9080/IfxBankWebService/services/IfxBankService"/>
     </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Before the new EGL based Web service can be utilized, it needs to be published to a WebSphere Application Server (WAS), since the current EGL Web service implementation requires some certain WebSphere Application Server specific runtime libraries. This restriction will be relaxed with future versions of EGL.

15.3.5 Test the new EGL based Web service

Finally we want to test the newly created and deployed EGL Web service, based on a former Informix 4GL library, containing I4GL business logic.



Enter the parameters of this WSDL operation and click **Go** to invoke.

http://localhost:9080/IfxBankWebService/services/IfxBankService

get_all_creditcards
account_num int
104
all_cards
ArrayOfRecordtype_ifxbank_lib_a_cards Add Remove

```
get_all_creditcardsResponse
  Group 1
    account_num (int): null

  return (int): 4
Group 2
  account_num (int): 104
  all_cards
    ArrayOfRecordtype_ifxbank_lib_a_cards
      card_num (int): 1001
      create_date (date): 1998-05-20
      name_on_card (string): Anthony Higgings
      is_active (string): y
      exp_date (date): 2002-05-30
      balance (decimal): 1000.40
      limit (decimal): 5000.00
      notes (string):
    ArrayOfRecordtype_ifxbank_lib_a_cards
      card_num (int): 1003
      create_date (date): 1998-05-22
      name_on_card (string): Andrea Higgings
```

Figure 15-22 Usage of the Web Services Explorer to test the IfxBankService

The easiest way of testing the new Web service is to use the built-in Web Services Explorer of RAD. To do that just locate the generated WSDL file either in the **Web Services** → **Services** folder or in the **Dynamic Web Projects** → **<EGL Web project name>** → WebContent → **WEB-INF** → **wsdl** folder. Right-click that file and then choose **Web Services** → **Test with Web Services Explorer**.

As soon as the browser window of the Web Services Explorer appears, select the **get_all_creditcards** operation in the Navigator window and provide a correct value (for example, 104) for the **account_num** value in the Actions window.

Then click Go and after short while you should see the results of the Web service request in the Status window (see also Figure 15-22 on page 453).

Example 15-20 shows the actual SOAP message which is being sent to the EGL based Web service provider.

Example 15-20 SOAP message being sent to the EGL Web service

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://services" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <q0:get_all_creditcards>
      <account_num>104</account_num>
      <all_cards/>
    </q0:get_all_creditcards>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 15-21 shows the results being returned (also SOAP formatted).

Example 15-21 Results of the Web service call in SOAP format

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
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:Header/>
  <soapenv:Body>
    <p632:get_all_creditcardsResponse xmlns:p632="http://services">
      <return>4</return>
      <account_num>104</account_num>
      <all_cards>
        <ArrayOfRecordtype_ifxbank_lib_a_cards>
          <card_num>1001</card_num>
          <create_date>1998-05-20</create_date>
          <name_on_card>Anthony Higgings</name_on_card>
          <is_active>y</is_active>
          <exp_date>2002-05-30</exp_date>
          <balance>1000.40</balance>
          <limit>5000.00</limit>
          <notes> </notes>
        </ArrayOfRecordtype_ifxbank_lib_a_cards>
        <ArrayOfRecordtype_ifxbank_lib_a_cards>
          <card_num>1003</card_num>
          <create_date>1998-05-22</create_date>
          <name_on_card>Andrea Higgings</name_on_card>
          <is_active>y</is_active>
          <exp_date>2001-05-30</exp_date>
          <balance>235.60</balance>
          <limit>2000.00</limit>
          <notes>His wife's card</notes>
        </ArrayOfRecordtype_ifxbank_lib_a_cards>
        <ArrayOfRecordtype_ifxbank_lib_a_cards>
          <card_num>1011</card_num>
```

```
            <create_date>1998-06-18</create_date>
            <name_on_card>Anthony Higgins</name_on_card>
            <is_active>y</is_active>
            <exp_date>2003-09-30</exp_date>
            <balance>650.40</balance>
            <limit>4000.00</limit>
            <notes>Business Card</notes>
          </ArrayOfRecordtype_ifxbank_lib_a_cards>
          <ArrayOfRecordtype_ifxbank_lib_a_cards>
            <card_num>1013</card_num>
            <create_date>1998-06-22</create_date>
            <name_on_card>Anthony Higgings</name_on_card>
            <is_active>n</is_active>
            <exp_date>2000-07-31</exp_date>
            <balance>0.00</balance>
            <limit>2000.00</limit>
            <notes>Stolen Card</notes>
          </ArrayOfRecordtype_ifxbank_lib_a_cards>
        </all_cards>
      </p632:get_all_creditcardsResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

15.3.6 Summary

This section has clearly shown that Informix 4GL customers and/or developers can easily integrate their existing valuable I4GL business logic with an SOA infrastructure by converting the existing I4GL code first into EGL and then leverage the built-in EGL Web services functionality to achieve the desired Web services support.

Looking back at our usage case scenario, ITSO Bank and Informix Bank could very efficiently integrate their heterogeneous systems due to the availability of the 4GL to EGL conversion path.

15.4 Scenario aggregating services as portlets

This chapter leads you through a scenario which demonstrates how to aggregate the services which are created in the previous chapters into a portal application, implementing a number of different portlets. The chapter contains the following topics:

- Overview
- Setting up the portlet project
- Creating Data Access Objects to retrieve database information
- Creating Java client proxies for Web Service interfaces
- Creating the credit score portlet
- Creating the foreign exchange calculator portlet
- Creating the mortgage accounts portlet
- The whole picture

If you just want to learn more about **Data Access Objects** (DAO) and a framework to use this technology in your Java applications, read 15.4.3, “Creating Data Access Objects to retrieve database information” on page 462.

In 15.4.4, “Creating Java client proxies for Web Service interfaces” on page 469 we explain how arbitrary SOAP-based Web Services can be easily accessed from J2EE applications using features of the IBM Rational Application Developer.

If you want to understand which steps are required to develop and test a simple Java portlet in RAD, read 15.4.5, “Creating the credit score portlet” on page 475. The chapters following this one demonstrate additional features of portlet applications.

15.4.1 Overview

The purpose of this scenario is to demonstrate how existing Web services and data stores can be easily aggregated into a single user experience using the Web portal technology. Our showcase is an Internet portal for ITSOBank customers. Customers shall get the ability to perform their business transactions, and get access to additional services provided by ITSOBank using a single Web portal site.

Note: Find more information about portal technology and the benefits of using this technology in a SOA environment in Chapter 4, “SOA and user interfaces with portals” on page 69.

Figure 15-23 shows the services which are made available to the customer via portlets. We focus on a few representative services whose creation is demonstrated in other chapters of this book, instead of including all services which may be offered by a banking Web portal. When the customer logs in to the ITSOBank customer Web portal, she or he can manage mortgage accounts (see details of existing mortgage accounts, and request new mortgages), perform currency conversions using current exchange rates, and see the personal FICO credit score.

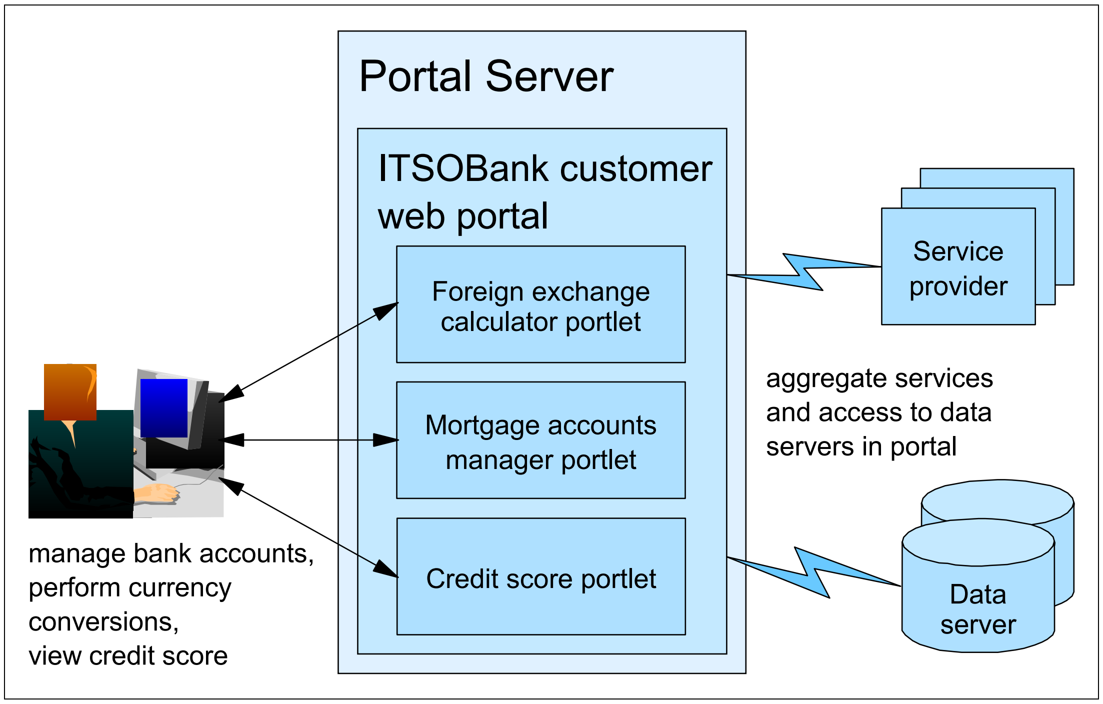

Portal Server
ITSOBank customer web portal
Foreign exchange calculator portlet
Mortgage accounts manager portlet
Credit score portlet
Service provider
aggregate services and access to data servers in portal
Data server
manage bank accounts, perform currency conversions, view credit score

Figure 15-23 Overview of ITSOBank customer Web portal

The portal application is comprised of a number of self-contained portlets, which run in a portal server environment (a portal server is an application server which provides additional

APIs and services). The local data server/data store is accessed by the portlets using the Data Access Object design principle, which abstracts the data access by adding a component layer between the business logic and the data repository. Web Services are integrated via Java Web Service client proxies. The portlets use these components and add a small layer of workflow or business logic, and user presentation (HTML, Web portlet pages).

We use the IBM Rational Application Developer 6.0 to implement the different components of the ITSOBank customer Web portal. The portlets are tested in the WebSphere Portal V5.1 test environment which is included in RAD. The complete portal application can be installed in a production environment using the WebSphere Portal Server 5.1, which provides, together with the other servers and tools of the WebSphere application suite, a robust and scalable runtime environment.

The portlets of our ITSOBank customer Web portal application, whose creation is demonstrated in the following chapters, have different degrees of complexity:

- **Credit score portlet**: This portlet reads the FICO score from the stored procedure developed in 15.2.2, "Implementation of the Credit Score function using RAD" on page 428 and displays the score to the user. This portlet is the most simple one, since no data processing is required.

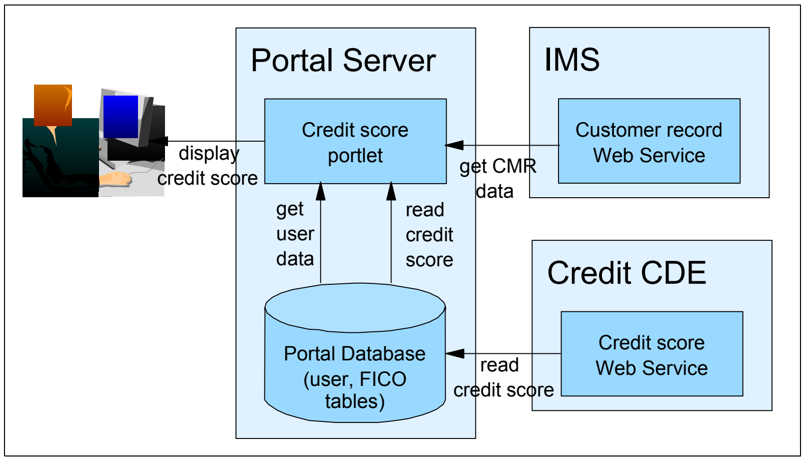


Figure 15-24 System context diagram of credit score portlet

- **Foreign exchange calculator**: The foreign exchange calculator portlet allows the user to convert amounts of money between different currencies. The user enters an amount, select source currency and target currencies. The portlet takes this input, calls the PHP currency conversion Web Service which is developed in 16.5, "Access an enterprise application using PHP" on page 528 to retrieve the currency exchange rate, and displays the result.

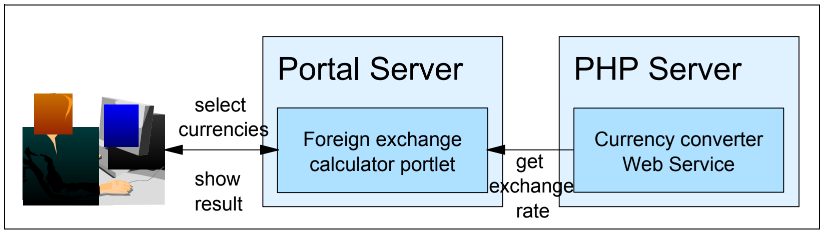
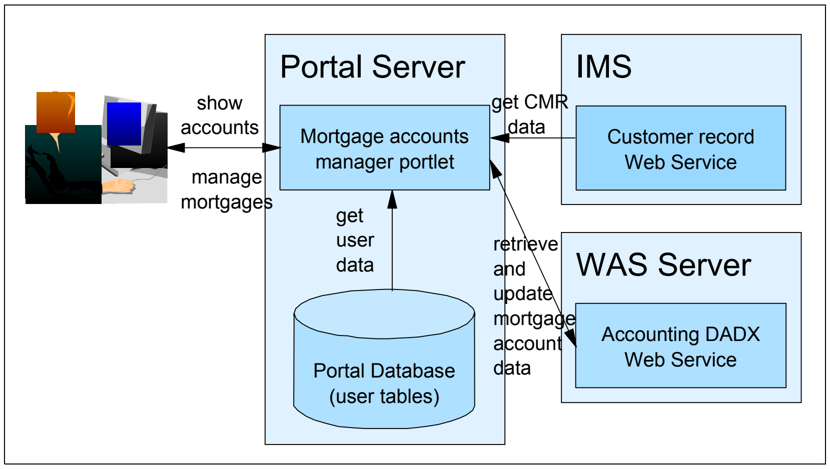


Figure 15-25 System context diagram of foreign exchange calculator portlet

- **Mortgage accounts manager**: This portlet is the most complex one in this scenario because it includes forms processing and a stripped-down workflow. The portlet reads the customer details from the CMR system to display them, and allows the user to manage mortgage accounts. A list with all existing accounts is shown, and the user can request a new mortgage account. This portlet interfaces with the ITSOBank accounting database using the DADX Web Services developed in 15.1.2, “Implementation of the Web Services using WORF” on page 404.



Portal Server
IMS
show accounts
manage mortgages
Mortgage accounts manager portlet
get CMR data
Customer record Web Service
get user data
retrieve and update mortgage account data
WAS Server
Accounting DADX Web Service
Portal Database (user tables)

Figure 15-26 System context diagram of mortgage accounts manager portlet

In the following chapters, we first show the steps required to setup the RAD environment to create the different components of this portal application.

Then, we introduce the techniques we use to access data and Web Services from within a Java portal application (though the same techniques apply to any J2EE application). We use Data Access Objects (DAO) to access data which resides in a DB2 database local to the portal application. Also, we demonstrate how to use RAD to generate Java-based Web Services proxies.

Finally, we show how to create the portlets that comprise our sample ITSOBank customer portal and use the portlet technology to aggregate the different data sources into user-viewable content.

15.4.2 Setting up the portlet project

This chapter shows how to setup a portlet project in RAD which is used in the subsequent chapters to contain the portlets developed in our scenario. The chapter is organized into the following tasks:

- Configure the WebSphere portal test environment
- Create a new portlet project
- Set up the portal database

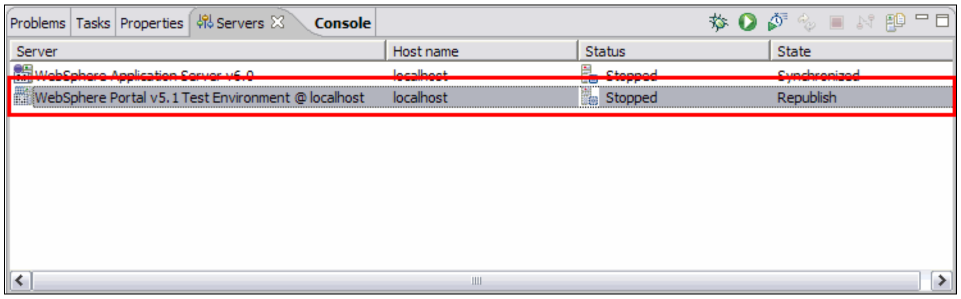
Tip: This chapter gives a brief description of the steps required to setup a project to and prepare the development environment to create portlet applications. You can find a comprehensive introduction into portlet development in the redbook *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*, SG24-6681.

Configure the WebSphere portal test environment

To run this project in the WebSphere Portal V5.1 Test Environment, you will need to create a new portal test server in RAD. To do this, follow the following instructions:

1. Switch to the Web perspective in RAD (select **Window** → **Open Perspective** → **Web**).

2. Open the **Servers** view at the bottom of the workbench, right-click and select **New** → **Server**.
3. In the Define a New Server dialog select the server type **IBM** → **WebSphere Portal V5.1 Test Environment**.
4. On the next page, WebSphere Server Configuration Settings, keep the default port number 9081.
5. The Portal Server Settings page allows you to define an administrative portal user (this user is not related to an operating system user). Keep the defaults which are wpsadmin for both, username and password.
6. Click **Finish** to complete the creation of the test environment. You will see a new entry for the test environment as shown in Figure 15-27.



Server	Host name	Status	State
~~WebSphere Application Server v6.0~~	~~localhost~~	~~Stopped~~	~~Synchronized~~
WebSphere Portal v5.1 Test Environment @ localhost	localhost	Stopped	Republish

Figure 15-27 WebSphere Portal Server showing up in RAD Servers view

Create a new portlet project

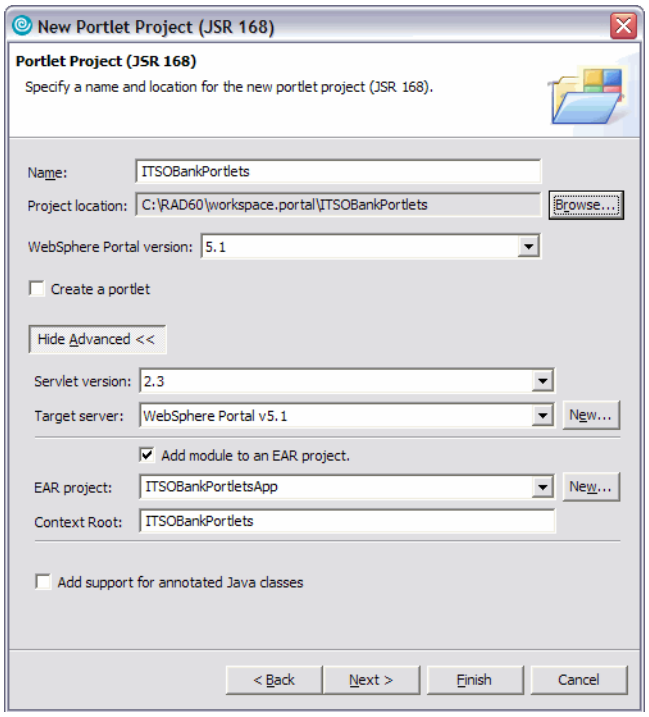
When you develop portlets with RAD, you create a portlet project which contains the portlet code, descriptors and resources. A portlet project has the following attributes:

- From a deployment perspective, it is an equivalent to a portlet application. A portlet application is actually a dynamic Web application with additional portlet descriptors.
- A portlet project can contain one or multiple portlets. All portlets within a portlet project share the same context which contains all resources such as images, property files and class files.
- You cannot deploy portlets to a WebSphere Portal Server. Instead, you deploy the complete portlet application (the deployment format of a portlet application is a WAR file).

Depending on the structure and needs of your portlets, you will group them into one or multiple portlet projects.

Our ITSOBank portal application consists of multiple portlets. Since we want to install all of the portlets into one portal, we choose to add them to a single portlet project. To create this portlet project in RAD, perform the following steps:

1. In RAD, select **File** → **New** → **Project** → **Portlet Project (JSR 168)**. The application asks you if the Portlet Development capability shall be enabled. Confirm that the capability can be enabled.
2. In the Portlet Project wizard, enter project name, target server and the EAR project name as shown in Figure 15-28 on page 460. Uncheck the **Create a portlet** option; we are adding our portlets later on.


New Portlet Project (JSR 168)

Portlet Project (JSR 168)

Specify a name and location for the new portlet project (JSR 168).

Name:	ITSOBankPortlets
Project location:	C:\RAD60\workspace.portal\ITSOBankPortlets
WebSphere Portal version:	5.1
☐ Create a portlet	
Servlet version:	2.3
Target server:	WebSphere Portal v5.1
☑ Add module to an EAR project.	
EAR project:	ITSOBankPortletsApp
Context Root:	ITSOBankPortlets
☐ Add support for annotated Java classes	

Figure 15-28 RAD Portlet Project (JSR 168) wizard

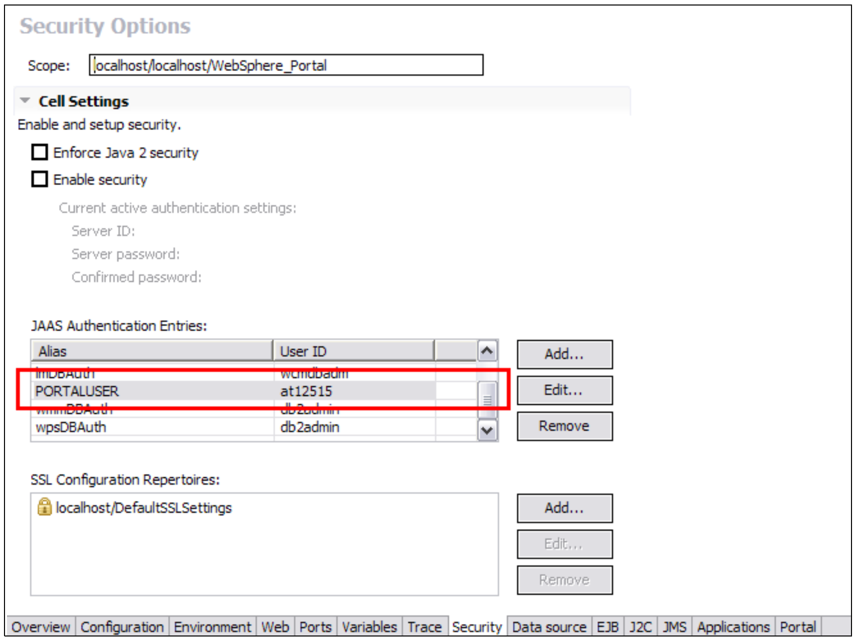
3. On the next page, uncheck all **Web Project features** and click **Finish** to create the portlet project in the workspace.

The wizard creates the Web deployment descriptor and portlet deployment descriptor for you. You did not select to create a portlet yet, so these descriptors are empty right now.

Set up the portal database

Some of the portlets access data from a local DB2 database (compare diagrams in Figure 15-24 on page 457 and Figure 15-26 on page 458). In order to go through the portlet scenarios, create this test database, named PORTALDB, on a DB2 instance on your workstation. You can find instructions to create and populate that database in Appendix E, “Additional material” on page 685.

The portlet code accesses the PORTALDB database using a JNDI data source. So we add a data source pointing to the PORTALDB in the WebSphere Portal V5.1 test environment. First, we open the portal server configuration in RAD by clicking on the server entry in the Servers view (see Figure 15-27 on page 459). Then we select the Security tab and add a JAAS authentication entry, entering the database user and password, as shown in Figure 15-29 on page 461.


Security Options

Scope: localhost/localhost/WebSphere_Portal

Cell Settings

Enable and setup security.

- [ ] Enforce Java 2 security
- [ ] Enable security

Current active authentication settings:
Server ID:
Server password:
Confirmed password:

JAAS Authentication Entries:

Alias	User ID
wmDBAuth	wcmdbadm
PORTALUSER	at12515
wmmDBAuth	db2admin
wpsDBAuth	db2admin

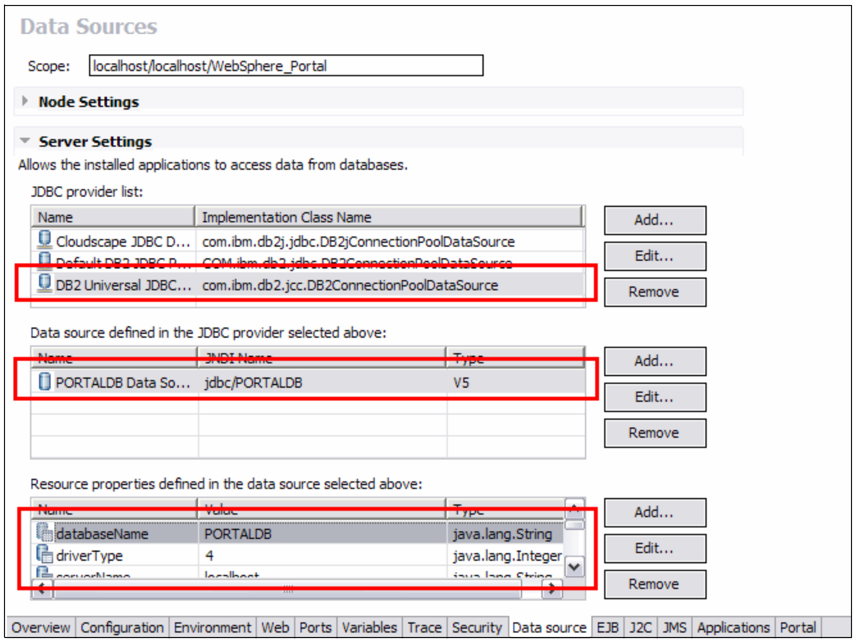
SSL Configuration Repertoires:

localhost/DefaultSSLSettings

Add...
Edit...
Remove
Add...
Edit...
Remove
Overview | Configuration | Environment | Web | Ports | Variables | Trace | Security | Data source | EJB | J2C | JMS | Applications | Portal

Figure 15-29 Add JAAS entry for PORTALDB access in portal server configuration

Then, we create a new DB2 data source in the Data source tab, as shown in Figure 15-30. Select the DB2 Universal JDBC Driver Provider from the JDBC provider list, add a data source with the JNDI name `jdbc/PORTALDB` (this name is used in the portlet code to refer to the data source) and select the JAAS alias which was created in the previous step in the Component-managed authentication alias field. Finally, add the database-specific information (database name, server name, port number) to the resource properties.


Data Sources

Scope: localhost/localhost/WebSphere_Portal

Node Settings

Server Settings

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Cloudscape JDBC D...	com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource
Default DB2 JDBC P...	COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource
DB2 Universal JDBC...	com.ibm.db2.jcc.DB2ConnectionPoolDataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
PORTALDB Data So...	jdbc/PORTALDB	V5

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	PORTALDB	java.lang.String
driverType	4	java.lang.Integer
serverName	localhost	java.lang.String

Add...
Edit...
Remove
Add...
Edit...
Remove
Add...
Edit...
Remove
Overview | Configuration | Environment | Web | Ports | Variables | Trace | Security | Data source | EJB | J2C | JMS | Applications | Portal

Figure 15-30 Add data source for PORTALDB in portal server configuration

15.4.3 Creating Data Access Objects to retrieve database information

This chapter introduces the Data Access Objects (DAO) design pattern and demonstrates the usage of DAOs in our portlet applications to access data from the local DB2 database.

Data Access Objects design pattern

The Data Access Objects design pattern is an essential part of good application architecture. A business layer which contains application or operation specific details and needs to access persistent data should not include functionality pertaining to the actual technology used for accessing the persistent data (the most common persistence technology being JDBC, but the pattern is extensible to any types of data sources, like object-relational access, XML-based access or content access). Using DAO means the underlying technology can be swapped without requiring to change the business objects of the application.

The business layer usually is application-specific, including Web Services, Java applications, or servlet-based Web applications. In our scenario we use portlets as business objects, and we create specific DAO classes to encapsulate the code accessing the DB2 database via JDBC. Figure 15-31 shows a class diagram with the basic roles of the DAO pattern:

- The *DataSource* represents the data source implementation. Data sources include relational database, object-oriented databases, XML repositories, flat filesystems, or any other existing or earlier systems or services.
- The *DataAccessObject* provides an abstraction of the underlying data source to enable transparent access to the data source. In our case the DAO classes contain JDBC calls to access our DB2 database.
- The *TransferObject* contains the data which is exchanged between the business objects and the data source. The data access object passes data retrieved from the data source via a transfer object, and receives updated data from the business layer also in a transfer object.
- The *BusinessObject* contains the application logic. In our scenario it is the respective Java portlet class containing the code to display data in the Web portal and request data updates from the client.

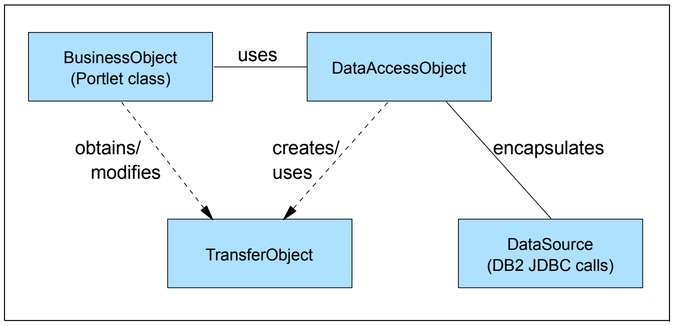


Figure 15-31 DAO class relationships

> **Tip:** A complete DAO design pattern definition, including additional code samples, is available at:
>
> http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

Data Access Objects for portal-specific data

We need to store some user-specific data of our ITSOBank portal in a local DB2 database. In order to keep flexible (maybe we can replace this database with a Cloudscape database, or an Informix data server) we use the DAO pattern to implement the data access.

Our ITSOBank customers are identified in our accounting and billing systems with their unique customer number. When a user logs on to the ITSOBank portal, he or she uses a portal-specific user name, which does not match with the customer number. So we keep a lookup table in the portal DB2 database that allows us to get the customer number of an authenticated portal user.

Figure 15-32 shows the main classes of our DAO implementation (this scenario is simplified to a high degree, so the number of classes required to implement the DAO layer seems an overkill, but if a large number of data objects is used, the DAO abstraction pays off):

- The abstract class `DAOFactory` is the entry point to our DAO layer. It provides an abstract getter method for each data access object, and contains a static method `getDAOFactory()` to retrieve the actual factory implementation based on a parameter value.

  This approach is very flexible. Imagine you have two implementations of the DAO layer, one for DB2 and another one for Informix. Then you can decide which implementation to use at runtime by providing the respective parameter value when retrieving the factory object. Our scenario only contains a DB2-specific implementation whose DAOs can be accessed through the class `DB2DAOFactory`.

- The interface `PortalUserDAO`, along with the `PortalUser` transfer object class, defines the abstract interface of the portal user mapping. The interface defines a method to create a portal user/customer number mapping, and another method to retrieve the customer number for a given portal user. The `PortalUser` transfer object contains the attributes of a virtual portal user, being the username and the customer number.

- The DAO class `DB2PortalUserDAO` finally contains the implementation of the JDBC access methods to retrieve or update portal user data in the local DB2 database.

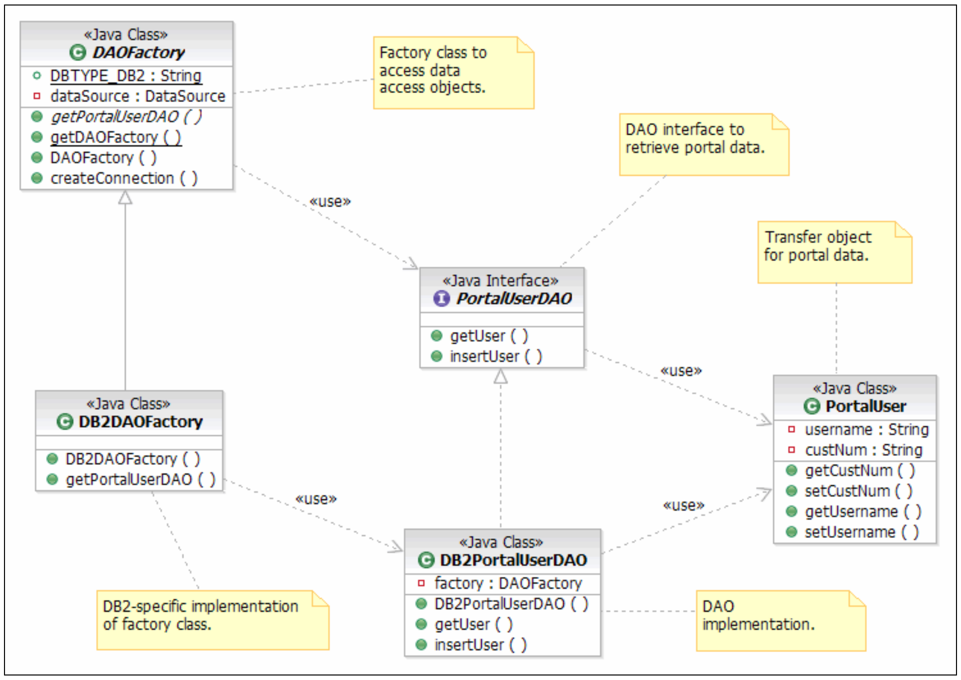


Figure 15-32 Class diagram showing portal-related DAO implementation

Note: This chapter does not contain step-by-step instructions to show how to create and extend the Java classes which comprise the DAO layer. Instead, the most important features of these classes are demonstrated. You can find instructions to download the complete code in Appendix E, “Additional material” on page 685.

The implementation of the `DAOFactory` class is shown in Example 15-22.

Example 15-22 DAOFactory class interface

```
package com.itso.sg247259.dao;

[...]

// Factory pattern used to allow the retrieval of Data Access Objects.
// This class needs to be inherited by an implementation-specific
// subclass which returns actual DAO implementations.
public abstract class DAOFactory {

  // Identifiers for DAO implementations (currently only DB2).
  public final static String DBTYPE_DB2 = "DB2";

  // Returns an actual implementation of the DAO factory, based on the given
  // database type.
  public static DAOFactory getDAOFactory(String dbType, String jndiURL)
    throws DAOException {
    // return the actual DAO factory implementation
    if (DBTYPE_DB2.equals(dbType))
      return new DB2DAOFactory(jndiURL);
    else
      throw new DAOException("Database type " + dbType + " not supported!");
  }

  // JDBC-based interface to data source.
  private DataSource dataSource = null;

  // Initializes the factory, creating a new data source object.
  // @param jndiURL JNDI URL of data source.
  public DAOFactory(String jndiURL)
    throws DAOException {
    try {
      dataSource = (DataSource) (new InitialContext()).lookup(jndiURL);
    } catch (NamingException ne) {
      throw new DAOException("Unable to retrieve data source!", ne);
    }
  }

  // Returns a JDBC connection to the data source.
  public Connection createConnection() throws DAOException {
    try {
      return dataSource.getConnection();
    } catch (Exception e) {
      throw new DAOException(e.getMessage(), e);
    }
  }
  /**
   * Returns the DAO for the portal user information.
   */
  public abstract PortalUserDAO getPortalUserDAO();
}
```

Notes to the implementation in Example 15-22 on page 464:

- We assume that all implementations of this DAO interface are running in a Java J2EE container and using the JDBC DataSource class to establish a connection to the data source. This assumption allows us to implement the connection-related parts of the code in this abstract base class. The method `DAOFactory.getDAOFactory()` which returns a DAOFactory object requires to specify a JNDI URL which defines the data source endpoint.
- DAO implementations use the method `createConnection()` to request a JDBC connection to the data source.
- The method `getPortalUserDAO()` is overloaded by an implementation of this factory class to return an implementation-specific data access object.
- Since the DAO pattern is an abstraction of the data source access layer, any exceptions originating in this layer need to be masked by generic exceptions. Examples for this type of exceptions are the (temporary) loss of the data source connection, or an object authorization issue. For this purpose we introduce a `DAOException` class which is thrown whenever the DAO layer encounters an exceptional state, that is, both for data source-specific problems and for application specific exceptions.

The implementation of the class `DB2DAOFactory` is very simple. it is shown in Example 15-23 for demonstration purposes. It returns the DB2 implementation of the portal user DAO. If additional DAOs are used, this class extended by getter method returning the implementation of those DAOs.

Example 15-23 DB2DAOFactory class implementation

```
package com.itso.sg247259.dao.db2;

import com.itso.sg247259.dao.*;

// DB2-specific DAO factory for the portal application.
public class DB2DAOFactory extends DAOFactory {

  // Constructor of DB2 DAO factory.
  public DB2DAOFactory(String jndiURL)
    throws DAOException {
    super(jndiURL);
  }

  // Returns a DAO for the access to the DB2 portal database.
  public PortalUserDAO getPortalUserDAO() {
    return new DB2PortalUserDAO(this);
  }
}
```

The `PortalUserDAO` interface and the implementation of the `PortalUser` transfer object class are not included here; the code does not contain special program logic and you can easily deduct it from the class diagram.

The implementation of the DB2PortalUserDAO class is contained in Example 15-24. The most notable detail is the code to access the portal DB2 database in the methods getUser() and insertUser() (the code of the second method is not shown due to its similarity to the code for the getUser() method). First, the methods request a JDBC connection object from the factory object, then they prepare and execute the SQL statement; finally, the JDBC resources are released and the data (that is, the transfer object) is returned.

Example 15-24 DB2PortalUserDAO class implementation

```
package com.itso.sg247259.dao.db2;

import java.sql.*;
import com.itso.sg247259.dao.*;
import com.itso.sg247259.dao.transfer.PortalUser;

// DB2 implementation of the portal user data access object. Accesses the
// PORTAL.USER table of a DB2 database to retrieve and update the portal user/
// customer number mapping.
public class DB2PortalUserDAO implements PortalUserDAO {

  // Reference to the DAO factory (to retrieve a JDBC connection).
  private DAOFactory factory;

  // Object constructor.
  public DB2PortalUserDAO(DAOFactory factory) {
    this.factory = factory;
  }

  // Reads the data of a user in the PORTAL.USER table and returns the data
  // in a PortalUser transfer object. Throws an exception if the user can't
  // be found.
  public PortalUser getUser(String username) throws DAOException {

    Connection con = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try {
      // get connection and prepare statement
      con = factory.createConnection();
      ps = con.prepareStatement(
        "SELECT USERNAME, CUSTNUM FROM PORTAL.USER WHERE USERNAME = ?");
      ps.setString(1, username);

      // Process results
      rs = ps.executeQuery();
      if (rs.next()) {
        PortalUser user = new PortalUser();
        user.setUsername(rs.getString(1));
        user.setCustNum(rs.getString(2));
        return user;
      }
      else
        throw new DAOException("Customer not available!");
    } catch (SQLException sqle) {
      throw new DAOException(sqle.getMessage(), sqle);
    } finally {
      DAOUtil.closeResources(rs, ps, con);
    }
  }

  // Inserts the given user record into the PORTAL.USER table.
  public void insertUser(PortalUser user) throws DAOException {
    [...] // use SQL INSERT statement to insert record into PORTAL.USER table
  }
}
```

An application which wants to access this DAO interface just needs to perform the following actions (for example, the code below demonstrates how to use the portal user DAO to insert a new user record).

```
// retrieve DB2-based DAO and specify datasource URL
PortalUserDAO userDAO =
  DAOFactory.getDAOFactory("DB2", "jdbc/PORTALDB").getPortalUserDAO();

// create new portal user transfer object
PortalUser user = new PortalUser();
user.setUsername("annprise");
user.setCustNum("19034954");

// insert user into portal database
userDAO.insertUser(user);
```

Use a DAO to call the FICO score stored procedure

The example above demonstrates the usage of Data Access Objects for the access of database tables and views. The representation of the relational data is reflected in the attribute structure of the corresponding transfer objects (for example, the `PortalUser` transfer object contains the attributes found in the `PORTAL.USER` database table), and the DAOs contain methods to manipulate the data.

DAOs can also be used to call stored procedures. In this case each stored procedure call maps to a method of a corresponding Data Access Object, and the parameters and results of the stored procedures are exchanged via transfer objects. To avoid writing a separate DAO for each stored procedure call, multiple stored procedures calls can be grouped within one DAO. it is good practice to group related procedures.

We demonstrate the DAO stored procedure concept with the credit score retrieval procedure we created in 15.2.2, "Implementation of the Credit Score function using RAD" on page 428. The interface of this stored procedure is repeated here:

```
CREATE PROCEDURE FICO.GetCreditScore(
        IN custSSN CHARACTER(11),
        IN custLastName VARCHAR(50),
        OUT newScore INTEGER)
```

It takes a user's social security number and lastname, and returns the user's credit score as a numeric value. Since the interface is very simple, we do not use separate transfer objects and pass the parameters directly instead.

The first step is to create the interface for a Data Access Object which is shown in Example 15-25 on page 468. It defines a method which takes the input parameters of the stored procedure and returns the credit score.

Example 15-25 FICOScoreDAO interface

```
package com.itso.sg247259.dao;

// DAO interface for credit score stored procedure.
public interface FICOScoreDAO {

  // Returns the credit score of a person, given the social security number
  // and the person's lastname.
  public int getFICOScore(String ssn, String lastname)
    throws DAOException;
}
```

The DB2-specific implementation of this interface calls the DB2 stored procedure and returns the credit score value. If a problem occurs, a `DAOException` is thrown. Example 15-26 shows the code of the `getFICOScore()` method. Apart from this method the class is identical to the `DB2PortalUserDAO` class (it also contains a DAO factory reference).

Example 15-26 DB2FICOScoreDAO class implementation

```
package com.itso.sg247259.dao.db2;

public class DB2FICOScoreDAO implements FICOScoreDAO {
  ...
  public int getFICOScore(String ssn, String lastname) throws DAOException {
    Connection con = null;
    CallableStatement cs = null;

    try {
      // get connection and prepare stored procedure call
      con = factory.createConnection();

      cs = con.prepareCall("CALL FICO.GETCREDITSCORE(?, ?, ?)");
      cs.setString(1, ssn);
      cs.setString(2, lastname);
      cs.registerOutParameter(3, Types.INTEGER);

      // call stored procedure and return result
      cs.executeUpdate();
      return cs.getInt(3);

    } catch (SQLException sqle) {
      throw new DAOException(sqle.getMessage(), sqle);
    } finally {
      DAOUtil.closeResources(null, cs, con);
    }
  }
  ...
}
```

The last step is to extend both factory classes `DAOFactory` and `DB2DAOFactory` with a method to return the data access object, as shown here for `DB2DAOFactory` (the `DAOFactory` method is only abstract, that is, without implementation):

```
...
// Returns a DAO for the access to the credit score SP.
```

```
public FICOScoreDAO getFICOScoreDAO() {
  return new DB2FICOScoreDAO(this);
}
...
```

Conclusion

In this chapter we demonstrated the usage of the Data Access Object design pattern to create a layer between the data source implementation and the business layer containing the application logic. The DAO layer is generic, so it can be used in a variety of environments (for example, in a service-oriented system, a Web application, or a portal application); the only restriction we introduced with our implementation is to use a JNDI data source, so the DAO component can only be part of a J2EE-based application (on the other hand only little effort is required to annul this restriction).

We showed that it makes sense to put implementation-specific code into a separate layer; here once more some of the advantages:

- **Increases data transparency**: The introduction of a DAO layer enables an implementation-independent point of view of the application data.
- **Data migration is easier**: Since the business layer does not contain details of the underlying data implementation, a migration of the data to a new data source does only involve changes in the DAO layer. The factory approach shown here has the additional advantage that no existing code needs to be changed. it is only required to implement DAO objects for the new data source and add a new factory method.
- **Reduces complexity of the application logic**: By separating the data access code from the application logic, the code in the business objects shrinks, increasing the maintainability of the code.

15.4.4 Creating Java client proxies for Web Service interfaces

Our portal application consumes a number of different Web Services:

- The PHP-based currency exchange rate Web Service (see 16.5, “Access an enterprise application using PHP” on page 528) is used to provide a foreign exchange calculator for portal users.
- The portal calls the DB2 DADX Web Services of the ITSOBank accounting database (see “Implementation of the Web Services using WORF” on page 404) to handle mortgage accounts.
- The customer records stored in the ITSOBank IMS system are made available via Web Services by the IMS SOAP Gateway (see 8.3, “Web Services with SOAP Gateway” on page 205).

Since we access the Web Services through Java code, we use the Web Services Client wizard of RAD to create proxy Java classes from the Web Services’s WSDL definitions. We can use these Java classes to call the Web Services in the same way we’d call simple Java objects. Thus we do not need to care about the SOAP layer and can concentrate on developing the application code.

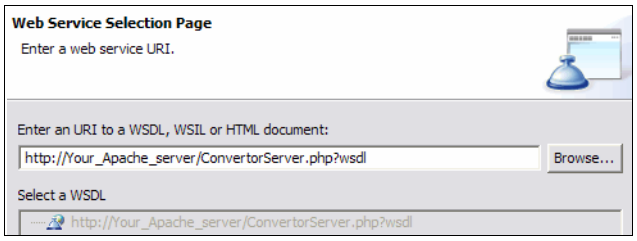
Create Web Service client proxy classes for the exchange rate service

The creation of the PHP exchange rate service is demonstrated in 16.5, “Access an enterprise application using PHP” on page 528. We assume the PHP service has been installed on an Apache HTTP Server, and the WSDL is available via the URL:

`http://`*`Your_Apache_server`*`/ConvertorService.php?wsdl`

To create the proxy Java classes, do the following:

1. Start the WebSphere Portal V5.1 Test Environment server. The RAD wizard requires the server to create the Web Service client bindings.
2. Select the ITSOPortlets project in the RAD project explorer.
3. Select **File** → **New** → **Other** → **Web Services** → **Web Service Client** and click **Next**.
4. When the Web Services Client dialog appears, select Java proxy for **Client proxy type**. Do not check the **Test the Web Service** and **Monitor the Web Service**, because these options will trigger the creation of additional classes and JSP files, which we do not need currently. You can use these options if you want to test and debug the Web Service.
5. On the next page, you are asked to enter the location of a WSDL, WSIL or HTML document (you can create Java proxy classes for all of these Web service definition types). Enter the PHP service WSDL as shown in Figure 15-33 and select **Next**.



Web Service Selection Page

Enter a web service URI.

Enter an URI to a WSDL, WSIL or HTML document:

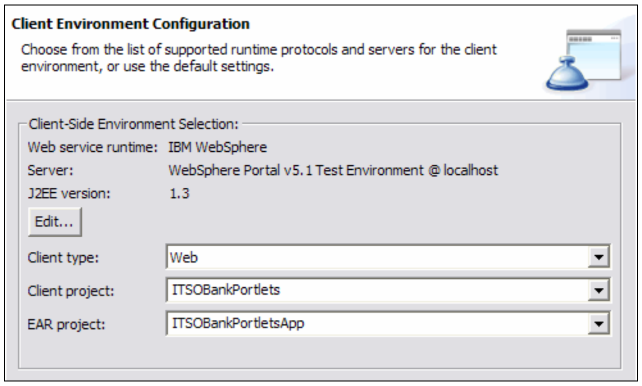
http://Your_Apache_server/ConvertorServer.php?wsdl

Select a WSDL

http://Your_Apache_server/ConvertorServer.php?wsdl

Figure 15-33 Enter WSDL location for Web Service client proxies

6. The next page allows you to select the client environment. Make sure your settings match those in Figure 15-34. The **Web service runtime** has to be set to IBM WebSphere, the **Server** to WebSphere Portal V5.1 Test Environment, and the proxy classes have to be added to your portlet project.



Client Environment Configuration

Choose from the list of supported runtime protocols and servers for the client environment, or use the default settings.

Client-Side Environment Selection:

Web service runtime: IBM WebSphere

Server: WebSphere Portal v5.1 Test Environment @ localhost

J2EE version: 1.3

Client type: Web

Client project: ITSOBankPortlets

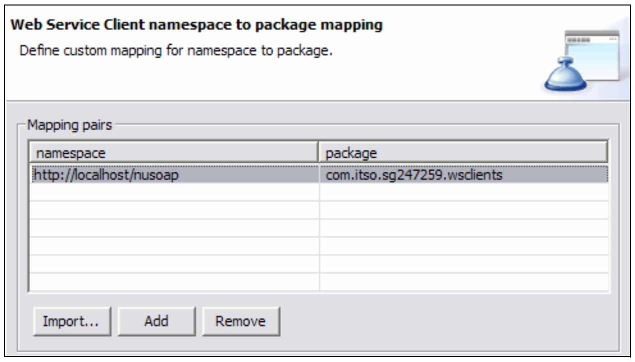
EAR project: ITSOBankPortletsApp

Figure 15-34 Environment configuration for Web Service clients

7. On the next page you define the Web Service security level (you can select XML Encryption or XML Signature which are both options of the WS-Security standard). Our Web Services do not support this type of security, so select No Security. Also, check **Define custom mapping for namespace to package**.

Tip: The generated Java proxy classes are put into a Java package whose name derives from the target namespace specified in the WSDL. If you want to put the Java classes into a different package which follows the standards of your application, define a mapping between the WSDL's target namespace and the Java package. Note that some WSDL files contain multiple target namespaces; the RAD wizard allows you to define a separate mapping for each namespace.

8. The next page allows you to add a namespace/package mapping. We put all proxy classes into the `com.itso.sg247259.wsclients` package, so the mapping we use is shown in Figure 15-35.



Web Service Client namespace to package mapping

Define custom mapping for namespace to package.

Mapping pairs

namespace	package
http://localhost/nusoap	com.itso.sg247259.wsclients

Figure 15-35 Custom packages for WSDL namespaces

9. Click **Finish** to complete the creation of the Web Service client proxies.

Table 15-3 Java classes generated by RAD Web Service Client wizard

Java class name	**Purpose**
your.package.ServiceName	Factory interface for the target service endpoint proxy.
*your.package.ServiceName*BindingStub	WebSphere-specific Web Service client stub.
*your.package.ServiceName*Information	Defines Java bindings for Web Service operations.
*your.package.ServiceName*Locator	Allows to retrieve a Web Service client instance.
*your.package.ServiceName*PortType	Java interface defining the Java method signatures of the Web Service operations.
*your.package.ServiceName*PortTypeProxy	Proxy class implementing the PortType interface.
*your.package.Beans**	Zero, one, or more JavaBean classes (and serializers) containing complex parameter values and results.

The Java classes created by the wizard are listed in Table 15-3. Additionally, the wizard creates Web Service client bindings in the WEB-INF/Webservicesclient.xml file and adds WSDL mapping XML files to the project.

We actually use only a few of the Java classes listed in Table 15-3 when calling a Web Service, namely the Locator class to retrieve a Web Service proxy object, the PortType interface to call the Web Service, and optionally the JavaBeans classes to handle the Web Service parameters and results.

The PHP exchange rate WSDL defines a single operation which takes source currency code and target currency code as input parameters, and returns the exchange rate. The PortType interface contains

Example 15-27 Java interface of the PHP exchange rate Web Service client

```
public interface ConvertorServerPortType extends java.rmi.Remote {
  // retrieve the exchange rate between two currencies
  public String getExchangeRate(String country1, String country2)
    throws java.rmi.RemoteException;
}
```

Example 15-28 demonstrates the usage of the proxy classes for our PHP Web Service, assuming we want to retrieve the current exchange rate between US dollars and the euro.

Example 15-28 Calling the PHP Currency Converter Web service

```
// retrieve a Currency Converter locator object
ConvertorServerLocator wsLocator = new ConvertorServerLocator();

// request a Currency Converter proxy object
ConvertorServerPortType wsClient = wsLocator.getConvertorServerPort();

// call the Currency Convertor Web Service
String rate = wsClient.getExchangeRate("USD", "EUR");
```

When you develop applications using Web Service clients, you may encounter the situation that the Web Services you use are installed in different locations (for example, on different servers for test environments versus production environments). In this case it is not required to regenerate the proxies whenever the service endpoint URL changes. Instead, you use a different method of the Locator class to request a proxy object, specifying the actual service endpoint URL. You can store the URL as a configuration setting of your application, which can be modified when you deploy your application. For example, you would request a proxy object for our PHP Web Service using the following statement:

```
// request a Currency Converter proxy object (specify endpoint URL)
ConvertorServerPortType wsClient = wsLocator.getConvertorServerPort(
    new URL("http://Your_server/ConvertorServer.php"));
```

Important: Web Service client proxies cannot easily moved between RAD projects because of the additional binding information. If you already generated the Web Service client proxies in a project and need the proxies in a different project, repeat the proxy generation step for the new project instead.

Create Web Service client proxy classes for the DADX Web Service

To create the client proxy classes for the DADX Web Service, you follow the same instructions as above when creating the proxy classes for the PHP Web Service. Only the location of the WSDL and the values for the custom namespace mapping differ.

If you setup the DADX Web Service as demonstrated in 15.1.2, "Implementation of the Web Services using WORF" on page 404, the WSDL of the Web Service is available using the following URL:

`http://`*`Your_DADX_WAS_server`*`/accountdb/worf/AccountGroup/AccountOps.dadx/WSDL`

When you open this URL in a Web browser, you see the `targetNamespace` definitions of the WSDL, which you can use to define a custom namespace mapping for your proxy classes.

Calling DADX Web Services by using the generated Java proxy classes requires more effort then calling the PHP Web Service as demonstrated above. The generated PortType class defines output parameters as well as result sets of the encapsulated stored procedures as parameters of the corresponding Java methods as shown in Example 15-29. The data type of each output parameter is a so-called *holder* type which allows to retrieve the value of the output parameter after the Web Service is called (holder types are, for example, the `StringHolder` type or the `CreateAccountReturnHolder` type).

Example 15-29 Java interface of the DADX accounting Web Service client

```
public interface TheSoapPortType extends java.rmi.Remote {

  // create a mortgage account
  public void createMortgageAccount(
      String CUSTNUM, BigDecimal AMOUNT, BigDecimal INTRATE, BigDecimal LIFETIME,
      BigDecimal PAYMENT, javax.xml.rpc.holders.IntHolder PSQLCODE,
      javax.xml.rpc.holders.StringHolder PSQLSTATE,
      javax.xml.rpc.holders.StringHolder PSQLERRMC,
      com.itso.sg247259.wsclients.holders.CreateAccountReturnHolder
          createAccountReturn) throws java.rmi.RemoteException;

  // return attributes of an existing mortgage account
  public void getMortgageAccount(
      String ACCTNUM, javax.xml.rpc.holders.IntHolder PSQLCODE,
      javax.xml.rpc.holders.StringHolder PSQLSTATE,
      javax.xml.rpc.holders.StringHolder PSQLERRMC,
      com.itso.sg247259.wsclients.holders.MortgageAccountDetailsHolder
          mortgageAccountDetails) throws java.rmi.RemoteException;

  // list all accounts of a customer
  public void listAccountNumbers(
      String CUSTNUM, javax.xml.rpc.holders.IntHolder PSQLCODE,
      javax.xml.rpc.holders.StringHolder PSQLSTATE,
      javax.xml.rpc.holders.StringHolder PSQLERRMC,
      com.itso.sg247259.wsclients.holders.AccountNumbersHolder accountNumbers)
          throws java.rmi.RemoteException;
}
```

Example 15-30 demonstrates how to call a DADX Web Service operation using the Java proxy classes. You request a locator object and a service object in the same way as you do for the PHP Web Service. But before actually calling the Web Service operation, you need to create objects for each holder parameter and pass these objects when you call the proxy method. Afterwards you can retrieve the returned values by accessing the `*holder*.value` attribute.

Example 15-30 Calling the DADX Web Service

```
// retrieve a DADX service locator object
TheServiceLocator wsLocator = new TheServiceLocator();

// request a DADX service proxy object
TheSoapPortType wsClient = (new TheServiceLocator()).getTheSoapPort();

// define holder objects for output parameters
IntHolder sqlCodeHolder = new IntHolder();
StringHolder sqlStateHolder = new StringHolder();
StringHolder sqlMessageHolder = new StringHolder();
AccountNumbersHolder accountNumbersHolder = new AccountNumbersHolder();

// call DADX service
wsClient.listAccountNumbers("134095", sqlCodeHolder, sqlStateHolder,
    sqlMessageHolder, accountNumbersHolder);

// check SQL code and print the number of returned accounts
if (sqlCodeHolder.value == 0) {
  System.out.println("# of returned accounts = " + accountNumbersHolder
      .value.getAccountNumbersResult().getAccountNumber().length);
}
```

Create client proxy classes for the IMS SOAP Gateway Web Service

The creation of the client proxy classes for the IMS SOAP Gateway service is performed in the same way as already described in the two chapters above. You create the classes using the RAD Web Service Client wizard.

The WSDL of the IMS service is either deployed on the IMS SOAP Gateway and can be retrieved through an URL, or the service developer provides the WSDL file. You can provide either the WSDL URL, or an WSDL file in the RAD Web Service Client wizard. If you want to place the created Java classes into specific packages, you need to use the custom namespace mapping as demonstrated in Figure 15-35 on page 471.

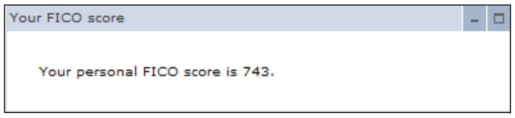
In our example we use the WSDL file CQUERY.wsdl which was created from the COBOL copybook in 8.3.3, “Generating a WDSL file using WebSphere Developer for zSeries” on page 208. The input and output records of the IMS service are visible in the Web Service signature and they are not easy to handle, so we create a special wrapper for the Web Service call which is shown in Example 15-31 - this wrapper can be used whenever the IMS service needs to be called. The wrapper method `CMRServiceWrapper.getCustomer(...)` expects the URL of the service endpoint as well as a valid customer number. It calls the service method and returns a `Customer` object containing the attributes of the customer record.

Example 15-31 Custom wrapper for IMS Web service call

```
// A wrapper for the IMS SOAP Gateway service providing customer details.
public class CMRServiceWrapper {

  // Returns the customer record for a given customer number, calling the
  // IMS SOAP Gateway CMR web service. Throws a DAOException if the service
  // is not available or reports a problem.
  public static Customer getCustomer(URL serviceEndPoint, String custNum)
    throws DAOException {

    // call IMS SOAP Gateway web service to retrieve the customer record
    CQUERYServiceLocator wsLocator = new CQUERYServiceLocator();

    // set call parameters
    INDATA in = new INDATA();

    in.setCustomernr(new Customernr(custNum));

    // these parameters are required for the COBOL copybook
    in.setIn_ll(new In_ll((short)32));
    in.setIn_zz(new In_zz((short)0));
    in.setIn_trcd(new In_trcd("CQUERY"));
    in.setTracex(new Tracex("N"));

    try {
      // call IMS SOAP service
      OUTDATA out = wsLocator.getCQUERYPort(serviceEndPoint).CQUERYOperation(in);

      // check result and throw exception in case of error
      if (out.getMessagex().getValue().indexOf("ERROR") > -1)
        throw new DAOException(
            "IMS SOAP service returned: " + out.getMessagex().getValue());

      // return customer record
      Customer cmr = new Customer();

      cmr.setAddress1(out.getAddress1().getValue());
      cmr.setAddress2(out.getAddress2().getValue());
      cmr.setSSN(out.getSsn().getValue());
      cmr.setCustNum(out.getCustomernr().getValue());
      cmr.setFirstname(out.getFirstnme().getValue());
      cmr.setMiddleInitial(out.getMi().getValue());
      cmr.setLastName(out.getLastname().getValue());
      cmr.setCity(out.getCity().getValue());
      cmr.setState(out.getState().getValue());
      cmr.setZipCode(out.getZipcd().getValue());
      cmr.setPhoneNumber(out.getPhone().getValue());
      cmr.setFaxNumber(out.getFax().getValue());
      cmr.setEmailAddress(out.getEmladdr().getValue());
      cmr.setSalutation(out.getSalutat().getValue());

      return cmr;
    }
    catch (Exception e) {
      throw new DAOException("Unable to call IMS SOAP service", e);
    }
  }
}
```

15.4.5 Creating the credit score portlet

This chapter demonstrates creating a credit score portlet which displays the user's credit score. Figure 15-36 shows the user interface of this portlet. The portlet displays only a single sentence containing the user's FICO score.



Your personal FICO score is 743.

Figure 15-36 View of the credit score portlet

Still, a few steps are required to return this information to the user when a request to display the portlet is received. These steps include:

1. The portlet retrieves the username of the logged in portal user.

2. The portlet reads the user's customer number from the DB2 PORTALDB database.
3. The portlet retrieves the CMR customer record from the IMS CMR system. This record contains the customer name and social security number.
4. The portlet calls the Credit Score stored procedure in the DB2 PORTALDB database to retrieve the credit score.
5. The credit score is written to the portlet response.

Prerequisites: The credit score portlet uses services of other system components. Before running this portlet in the test environment, make sure that the systems which are accessed by the portlet are available:

- DB2 PORTALDB database
- IMS SOAP Gateway of the CMR system
- Application Server providing the Credit CDE credit score Web Service

The IMS SOAP Gateway example is demonstrated in 8.3, "Web Services with SOAP Gateway" on page 205. The Credit CDE credit score Web Service is fictitious, so you have to install it. You can find an example Credit CDE application which can be deployed in a WebSphere Application Server and used for this scenario in Appendix E, "Additional material" on page 685.

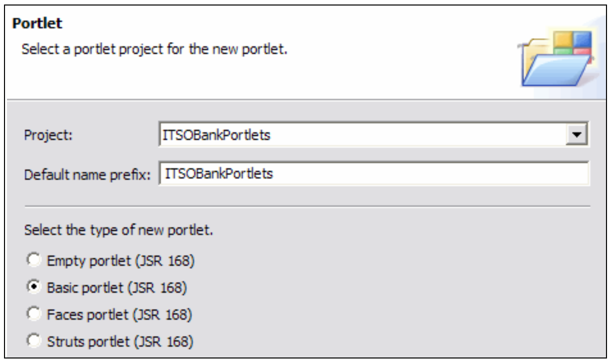
This chapter continues with a description of the following tasks which need to be performed to create the portlet:

- Create a new portlet
- Create a method to retrieve the name of the portal user
- Create a DB connector factory
- Register endpoint of IMS CMR Web Service
- Extend the portlet class with the application logic
- Modify the portlet JSP to show the credit score
- Run the portlet in the portal test environment

Create new portlet

To add the new credit score portlet to the ITSOBankPortlets project perform the following steps:

1. Open the New Portlet wizard in RAD by selecting **File** → **New** → **Other** → **Portal** → **Portlet**. On the first page of the wizard, select your portlet project and specific Basic portlet as **portlet type** as shown in Figure 15-37. The Basic portlet type option creates skeletons for the portlet Java class and the JSP pages.

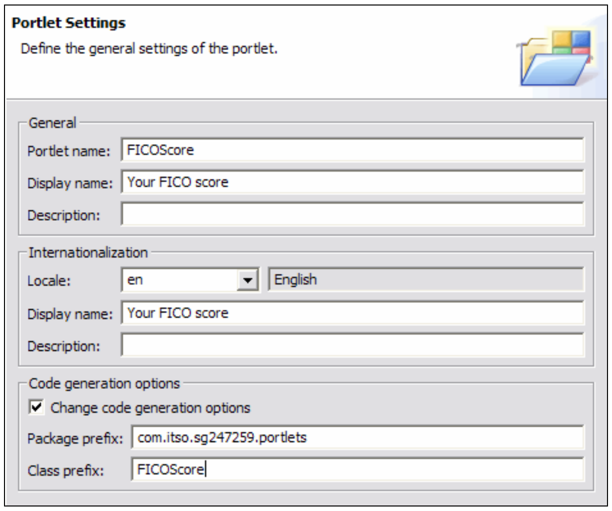

Portlet
Select a portlet project for the new portlet.

Project: ITSOBankPortlets
Default name prefix: ITSOBankPortlets

Select the type of new portlet.
- Empty portlet (JSR 168)
- [✓ Basic portlet (JSR 168)]
- Faces portlet (JSR 168)
- Struts portlet (JSR 168)

Figure 15-37 Select portlet type in New portlet wizard

2. On the next page, enter the portlet settings as demonstrated in Figure 15-38. The **Display name** attribute appears in the portlet header when it is displayed in a portal page in the user's Web browser. You can enter the display name in different languages; the user's locale defines in which language it is displayed in the Web browser. We also change the **Package prefix** attribute to make sure the generated Java classes fit into our package structure.



Portlet Settings
Define the general settings of the portlet.

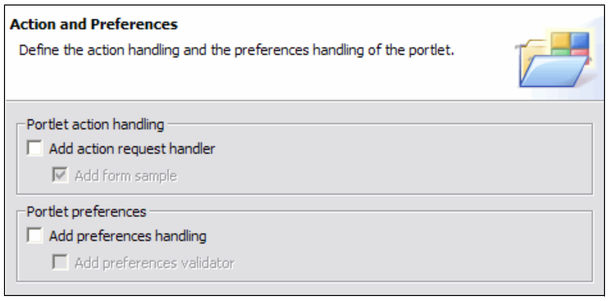
General
Portlet name: FICOScore
Display name: Your FICO score
Description:

Internationalization
Locale: en English
Display name: Your FICO score
Description:

Code generation options
☑ Change code generation options
Package prefix: com.itso.sg247259.portlets
Class prefix: FICOScore

Figure 15-38 Enter portlet settings in New portlet wizard

3. The next page allows you to add an optional action handler and handling of portlet preferences (which can be changed by the user when the portlet is in *edit* mode). Our credit score portlet does not process user input, so we do not need any of these features. Uncheck the **Add action request handler** option as shown in Figure 15-39. Click **Finish** to close the wizard.



Action and Preferences
Define the action handling and the preferences handling of the portlet.

Portlet action handling
☐ Add action request handler
☑ Add form sample

Portlet preferences
☐ Add preferences handling
☐ Add preferences validator

Figure 15-39 Define action and preferences options in New portlet wizard

Let us look at the generated resources which show up in the RAD Project explorer (see the marked resources in Figure 15-40 on page 478). The wizard creates an entry for the new portlet in the portlet deployment descriptor, a Java class `FICOScore` which handles the portlet processing, localized resource property files `FICOScoreResource*`, and a JSP file `FICOScoreView.jsp` which defines the user interface of the portlet. We only need to extend the Java class and the JSP file to have the portlet meet our requirements.

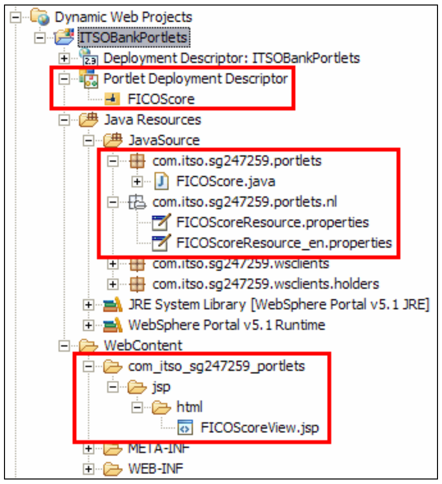

Dynamic Web Projects
ITSOBankPortlets
Deployment Descriptor: ITSOBankPortlets
Portlet Deployment Descriptor
FICOScore
Java Resources
JavaSource
com.itso.sg247259.portlets
FICOScore.java
com.itso.sg247259.portlets.nl
FICOScoreResource.properties
FICOScoreResource_en.properties
com.itso.sg247259.wsclients
com.itso.sg247259.wsclients.holders
JRE System Library [WebSphere Portal v5.1 JRE]
WebSphere Portal v5.1 Runtime
WebContent
com_itso_sg247259_portlets
jsp
html
FICOScoreView.jsp
META-INF
WEB-INF

Figure 15-40 Resources created by the New portlet wizard

Note: The credit score portlet is very simple, so splitting its functionality between a Java class and a JSP page seems to be an overkill. But in the design of this portlet we follow the Model-View-Controller (MVC) principle which separates application logic from the user interface and the data model. In our scenario, the Java portlet class acts as controller, the JSP page is the view, and the credit score information exchanged between these components is the model. In the following scenarios we continue to use this principle for portlets with higher complexity.

Request username of currently logged in portal user

The credit score portlet shows the personalized FICO score of a user who accesses the Web portal. To read the FICO score from our database we first need to identify this user. The WebSphere Portal Server provides a system programming interface for its user management implementation called Portal User Management Architecture (PUMA) which is used by our portlet to request this information.

Note: The JSR standard 168 does not define an interface to retrieve the portal user information, so the PUMA service provided by WebSphere Portal Server is a proprietary technology. Additional information about this service is available at

http://publib.boulder.ibm.com/infocenter/wpdoc/v510/index.jsp?topic=/com.ibm.wp.ent.doc/wps/wpspuma.html

The user information is also required by other portlets, so we create a separate class `PortalUserInfo` which contains a static method to request the user information from the PUMA service, as shown in Example 15-32.

Example 15-32 Implementation of PortalUserInfo.getCurrentUsername() method

```
package com.itso.sg247259.portlets.helpers;

import javax.naming.InitialContext;
import javax.portlet.PortletRequest;

import com.ibm.portal.portlet.service.PortletServiceHome;
import com.ibm.portal.um.PumaProfile;
import com.ibm.portal.um.portletservice.PumaHome;
```

```
public class PortalUserInfo {

  // Returns the portal username of the user which is currently logged in. If this
  // information can't be retrieved, an empty string is returned.
  public static String getCurrentUsername(PortletRequest request) {
    try {
      PortletServiceHome psh = (PortletServiceHome) (new InitialContext())
        .lookup("portletservice/com.ibm.portal.um.portletservice.PumaHome");

      if (psh != null) {
        PumaHome service = (PumaHome) psh.getPortletService(PumaHome.class);

        PumaProfile pp = service.getProfile(request);

        String id = pp.getIdentifier(pp.getCurrentUser());

        // the user identifier is returned in the format "uid=AAAA,o=BBB"; we
        // only need the uid, so we parse this information from the string
        return id.substring(
            id.indexOf("uid=") + 4, id.indexOf(',', id.indexOf("uid=")));
      }
    } catch (Exception e) {}

    return "";
  }
}
```

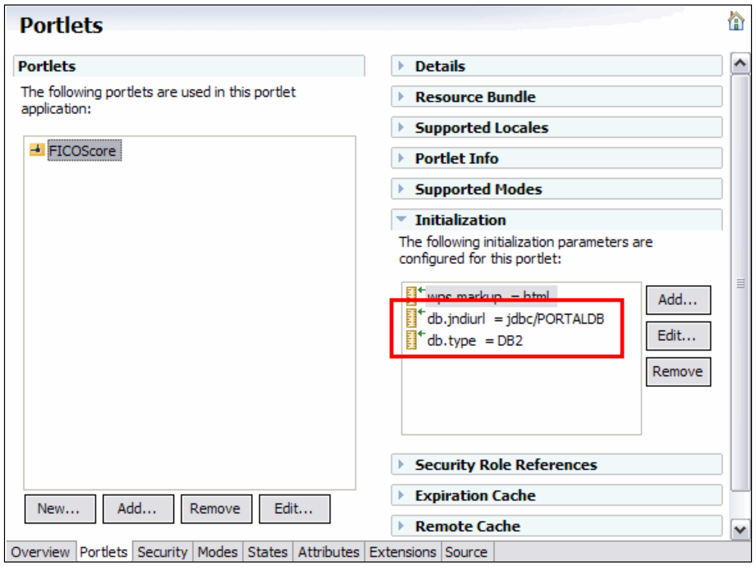
Create a DB connector factory

The credit score portlet system context diagram (see Figure 15-15 on page 430) shows that the portlet access the DB2 PORTALDB database to retrieve user data and the user's credit score rating. To access this data, we use the DAO component of 15.4.3, "Creating Data Access Objects to retrieve database information" on page 462.

The DAO requires the following parameters to establish the connection to the database:

- A JNDI URL pointing to the DB2 PORTALDB database
- The DAO-specific database type (in our case it is *DB2*)

We want to avoid to hard-code this information, so that the actual values can be changed without requiring to updating the portlet code. So we add these attributes as portlet initialization parameters which can be accessed in the portlet code using the `PortletContext` interface. To add portlet initialization parameters, open the portlet deployment descriptor of the ITSOBankPorlets project (you find the deployment descriptor in the Project explorer view) and select the FICOScore portlet. Then add the parameters `db.jndiurl` and `db.type` as demonstrated in Figure 15-41 on page 480.


Portlets

Portlets

The following portlets are used in this portlet application:

FICOScore

New... Add... Remove Edit...

Details

Resource Bundle

Supported Locales

Portlet Info

Supported Modes

Initialization

The following initialization parameters are configured for this portlet:

wps.markup = html
db.jndiurl = jdbc/PORTALDB
db.type = DB2

Add... Edit... Remove

Security Role References

Expiration Cache

Remote Cache

Overview | Portlets | Security | Modes | States | Attributes | Extensions | Source

Figure 15-41 Add initialization parameters for credit score portlet

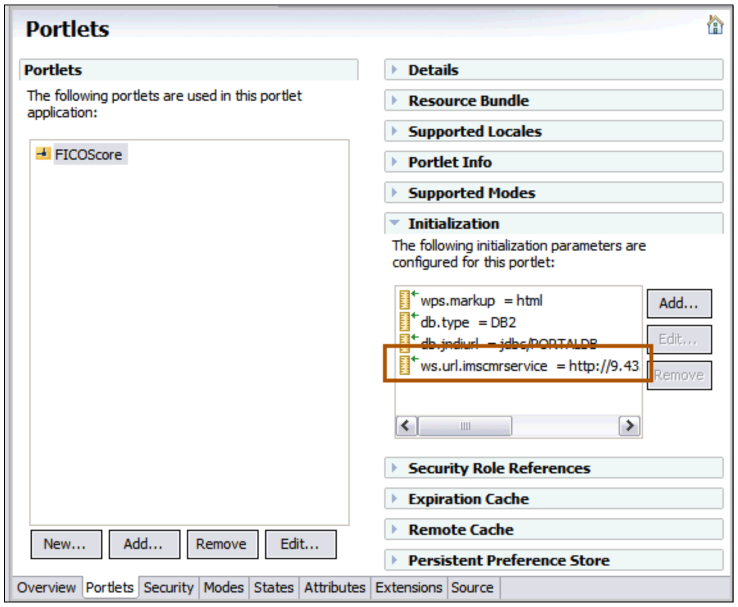
The code to access the DB2 PORTALDB database needs to be shared between our portlets, so we put it into a separate Java class called `PortalDBConnector`. The source code of this class is shown in Example 15-33. The implementation of this class is quite simple. it is only required to read the portlet parameters and return a `DAOFactory` object with these parameters.

Example 15-33 Implementation of PortalDBConnector class

```
package com.itso.sg247259.portlets.helpers;

import javax.portlet.PortletContext;

import com.itso.sg247259.dao.*;

// Provides access to the portal's Data Access Object factory.
public class PortalDBConnector {

  // Returns a DAO factory object. The portlet context has to contain the
  // JNDI URL of the data source and the database type which is used.
  public static DAOFactory getDAOFactory(PortletContext context)
    throws DAOException {
    return
        DAOFactory.getDAOFactory(getDatabaseType(context), getJNDIURL(context));
  }

  // Returns the data source JNDI URL defined in the portlet context.
  private static String getJNDIURL(PortletContext context) {
    return context.getInitParameter("db.jndiurl");
  }

  // Returns the database type defined in the portlet context.
  private static String getDatabaseType(PortletContext context) {
    return context.getInitParameter("db.type");
  }
}
```

Register endpoint of IMS CMR Web Service

The FICO portlet uses the CMR Web Service provided by the ITSOBank IMS SOAP Gateway to retrieve customer information. For the same reasons as for the DB connector facility we make the endpoint of this service configurable via the portlet's initialization parameter facility. So we add another portlet parameter which contains the URL address of the IMS service as demonstrated in Figure 15-42.



```
wps.markup = html
db.type = DB2
db.jndiurl = jdbc/PORTALDB
ws.url.imscmrservice = http://9.43
```

Figure 15-42 Add service endpoint initialization parameter for credit score portlet

Extend the portlet class the application logic

By now we have all prerequisites met to complete the application logic; we can retrieve the portal user information and get Data Access Objects for the PORTALDB database. So the next step in our scenario is to complete the `FICOScore` class with the application logic of this portlet.

The application logic is pretty simple: When the portlet retrieves the first render request from the portal container for a specific portlet session, the user's credit score value is retrieved and stored in the portlet session. If a problem occurs (for example, the interface to the portlet database is unavailable, or if a connection to the Credit CDE credit score Web Service cannot be established), an error message is stored in the portlet session.

The generated `FICOScore` portlet class does only need to be modified slightly for this purpose. It contains a method `doView()` inherited from the `GenericPortlet` class which is called when a render request for the portlet view mode is received. We extend this method as shown in Example 15-34 (the additional code is marked bold) to update the portlet session with the credit score. Additional changes in Example 15-34 are the definition of constants for the session attribute names.

Example 15-34 Extend FICOScore portlet class with application logic

```
...
// portlet session attribute name for credit score
public static final String FICO_SCORE = "FICOScore";

// portlet session attribute name for error message
public static final String FICO_MESSAGE = "FICOMessage";
```

```
// process render request for the portlet 'view' mode
public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  // retrieve the credit score
  updateFICOScore(request);

  // Invoke the JSP to render
  PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(
      getJspFilePath(request, VIEW_JSP));
  rd.include(request, response);
}
...
```

Finally, we create a new method `FICOScore.updateFICOScore()` which sets the information in the portlet session, as demonstrated in Example 15-35. The method first resets the session's error message attribute (so that in case of a temporary error accessing the external systems the portlet retries to retrieve credit score data). Then it checks if the credit score data is already stored in the portlet session, and if not, creates DAO objects for the information we need to retrieve, and uses the DAOs and Web service wrappers to access portal user, customer, and credit score data.

Example 15-35 Method updateFICOScore() sets the credit score portlet session attribute

```
// Sets the portlet sessions FICO_SCORE attribute with the user's credit
// score; in case of a problem it sets the FICO_MESSAGE session attribute.
private void updateFICOScore(PortletRequest request) {

  // retrieve the current portlet session
  PortletSession session = request.getPortletSession();

  // reset the error message attribute
  session.setAttribute(FICO_MESSAGE, null);

  try {
    // we cache the FICO score in the portlet session; its value is
    // not volatile, so we need to read it only once per session
    if (session.getAttribute(FICO_SCORE) == null) {

      // retrieve DAOs for portal user and customer data
      DAOFactory daoFactory =
          PortalDBConnector.getDAOFactory(getPortletContext());
      PortalUserDAO userDAO = daoFactory.getPortalUserDAO();
      FICOScoreDAO ficoScoreDAO = daoFactory.getFICOScoreDAO();

      // call IMS SOAP Gateway web service to retrieve the customer record
      String wsURL = request.getPortletSession().getPortletContext()
          .getInitParameter("ws.url.imscmrservice");
      Customer customerRecord = CMRServiceWrapper.getCustomer(new URL(wsURL),
        userDAO.getUser(PortalUserInfo.getCurrentUsername(request)).getCustNum());

      // read customer record
      Customer customerRecord = customerDAO.getCustomer(
        userDAO.getUser(PortalUserInfo.getCurrentUsername(request)).getCustNum());

      // retrieve FICO credit score and store it in portlet session
      session.setAttribute(
          FICO_SCORE, String.valueOf(
              ficoScoreDAO.getFICOScore(
                  customerRecord.getSSN(), customerRecord.getLastName())));
    }
  }
  catch (DAOException daoe) {
    System.err.println("FICOScore: Caught DAOException " + daoe);

    // in case of a problem set the error message
    session.setAttribute(
        FICO_MESSAGE, "The FICO score service is temporarily unavailable!");
  }
}
```

Modify the portlet JSP to show the credit score

When the `FICOScore` portlet class receives a render request, it executes the `doView()` method whose last action is to pass control to the `FICOScoreView.jsp` JSP. This JSP is responsible to display the information in the portlet output.

The JSP reads the data which is passed in the portlet session object. If an error message is set, this message is displayed. Otherwise the credit score of the user is shown. The implementation of the JSP is demonstrated in Example 15-36.

Example 15-36 Credit score portlet view defined in FICOScore.jsp

```
<%@ page session="false" contentType="text/html"
    import="com.itso.sg247259.portlets.*" %>
<%@taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>

<DIV style="margin: 20px;">
<P>
<%
   // retrieve portlet session attributes
   javax.portlet.PortletSession portletSession =
       renderRequest.getPortletSession();

   String ficoScore = (String) portletSession.getAttribute(FICOScore.FICO_SCORE);
   String message = (String) portletSession.getAttribute(FICOScore.FICO_MESSAGE);

   if (message == null) {
%>
     Your personal FICO score is <%= ficoScore %>.
<% } else { %>
     <%= message %>
<% } %>
<P>
</DIV>
```

Run the portlet in the portal test environment

The development of the portlet is complete, so you can test it now using the WebSphere portal test environment which you created in "Configure the WebSphere portal test environment" on page 458.

Go to the Servers view in the RAD workbench, right-click the WebSphere Portal Server V5.1 Test Environment and select **Add and remove projects.** Select the ITSOBankPortletsApp project and add it to the server as shown in Figure 15-43.

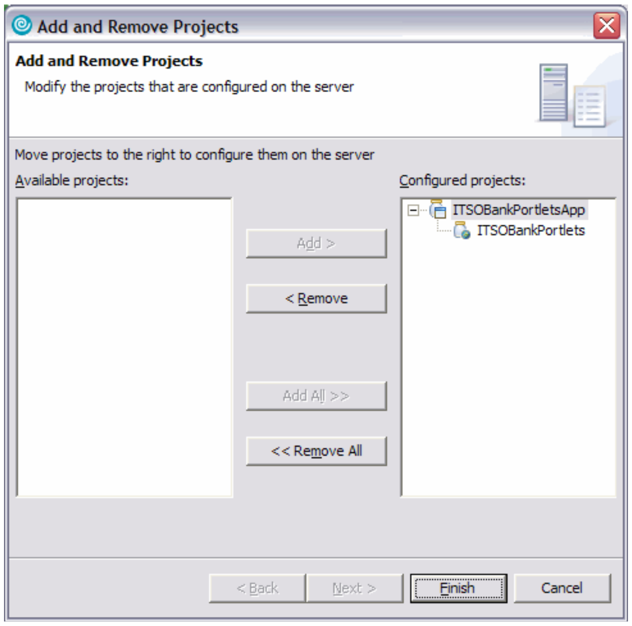


Figure 15-43 Add portlet project to portal test environment

Note: The portlet application needs to be added to the portal test environment only once. When you change the portlet project by adding new portlets, you only need to republish the project to the test environment.

Now start the WebSphere Portal V5.1 Test Environment via the Servers view. Right-click the test environment and select `Start`. The Console view shows the progress of the start procedure.

When the server is started, open a Web browser and enter the following URL to go to the portal page:

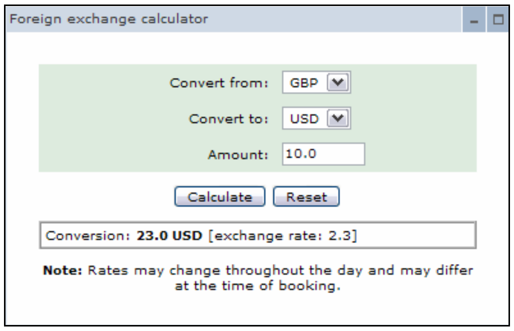
```
http://localhost:9081/wps/portal
```

You are asked for a username to login to the portal. Enter *wpsadmin* for username and password. A portal page containing the credit score portlet appears. In case all systems which are accessed by this portlet are running, you should see the credit score displayed in this portlet as shown in Figure 15-36 on page 475.

15.4.6 Creating the foreign exchange calculator portlet

The foreign exchange calculator is a form-based portlet which allows the user to convert amounts of money between different currencies. The portlet uses a Web Service to retrieve up-to-date currency exchange rates from a separate system, as shown in the system context diagram in Figure 15-25 on page 457.

The portlet provides minimum user interaction (the user can enter the amount and select currencies), so the implementation needs to go beyond the view-only credit score portlet presented in the previous chapter. Figure 15-44 shows the user interface of the portlet. It contains a Web form to allow the user to select currencies and enter an amount, and shows the result below of the form.



Conversion: **23.0 USD** [exchange rate: 2.3]

Note: Rates may change throughout the day and may differ at the time of booking.

Figure 15-44 View of the foreign exchange calculator portlet

The implementation of the portlet follows the Model-View-Controller (MVC) principle, as shown in Figure 15-45:

- A `CurrencyConverterSessionBean` object representing the **Model** encapsulates the attributes of the portlet session (the selected currencies, exchange rate, amounts).
- The `CurrencyConverter` portlet class which is the portlet **Controller** processes an incoming conversion request from the client (it reads the HTML form values and calculates the final amount based on the exchange rate received from the PHP Web Service) and sets the attributes of the session bean with the results.
- The application **View** `CurrencyConverterView.jsp` reads the values from the session bean and displays the HTML form as well as the conversion result.

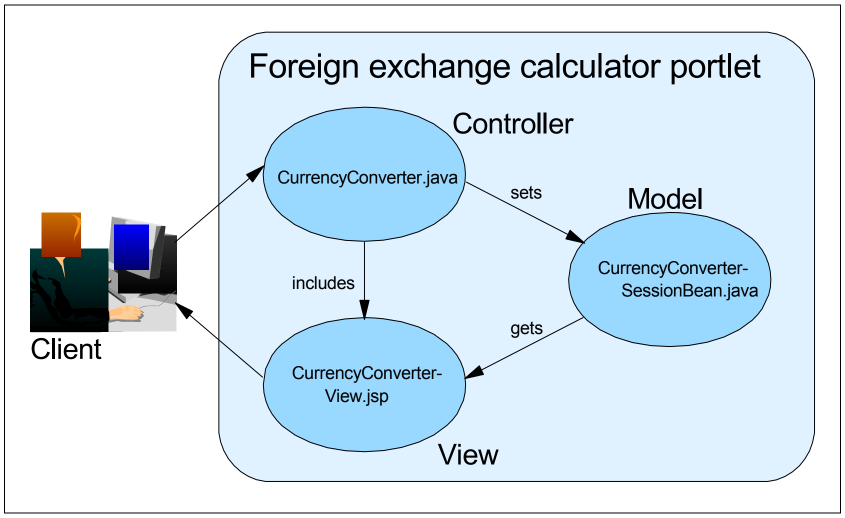

Foreign exchange calculator portlet
Controller
CurrencyConverter.java
sets
Model
CurrencyConverter-SessionBean.java
includes
gets
CurrencyConverter-View.jsp
View
Client

Figure 15-45 Foreign exchange calculator portlet application

The chapter demonstrates how to create this portlet in RAD. Some of the steps are already explained in 15.4.5, “Creating the credit score portlet” on page 475, so we do not get into the same level of detail. This chapter continues describing the following tasks:

- Create a new portlet.
- Add session attributes to the session bean class.
- Extend the portlet class with the application logic.
- Modify the portlet JSP to show the HTML form and result.
- Run the portlet in the test environment.

Create a new portlet

To add the foreign exchange calculator portlet to the existing ITSOBankPortlets portlet project in RAD, open the New portlet wizard by selecting **File** → **New** → **Other** → **Portal** → **Portlet** and follow the steps as shown in 15.4.5, "Creating the credit score portlet" on page 475. When you enter the **Portlet settings** on the third page, use the following values:

- The portlet name should be *CurrencyConverterPortlet*.
- Enter *Foreign exchange calculator* as display name (this text will appear in the portlet title).
- The package prefix is *com.itso.sg247259.portlets*.
- The class prefix is *CurrencyConverter*.

On the **Actions and preferences** page check the options Add action request handler and Add form sample.

Tip: When you select the option *Add action request handler*, the New portlet wizard extends the portlet class with an action handler stub. When you additionally select option *Add form sample*, the wizard adds an HTML form sample to the generated JSP file, adds a session bean class, and includes sample code to the action handler to process the form.

The wizard creates the files shown in Figure 15-45 on page 485. Our task is to modify and extend these files to implement the foreign exchange calculator functionality, as demonstrated below.

Add session attributes to the session bean class

An object of the session bean class of our Foreign exchange calculator portlet holds all attributes of the portlet session, in our case all attributes which are displayed in the portlet window. The following session attributes are defined:

- Amount of money to be converted
- Source currency
- Target currency
- Converted amount of money
- Currency exchange rate used for the conversion
- An (optional) error message

We modify the session bean class `CurrencyConverterSessionBean` and add these attributes. Example 15-37 lists the getter/setter methods which are implemented by this class.[3]

Example 15-37 Interface of the CurrencyConverterSessionBean class

```
package com.itso.sg247259.portlets;

// Session bean of foreign exchange calculator portlet
public class CurrencyConverterSessionBean {

  // Returns the amount of money to be converted
  public double getAmount();
```

[3] The implementation of these method is trivial: You need to store the attributes in instance variables and access these variables through the getters and setters. The additional materials include the complete implementation of this class.

```
  // Sets the amount of money to be converted
  public void setAmount(double amount);

  // Returns the source currency
  public String getSourceCurrency();

  // Sets the source currency
  public void setSourceCurrency(String sourceCurrency);

  // Returns the target currency
  public String getTargetCurrency();

  // Sets the target currency
  public void setTargetCurrency(String targetCurrency);

  // Returns the converted amount of money
  public double getConvertedAmount();

  // Sets the converted amount of money
  public void setConvertedAmount(double convertedAmount);

  // Returns the currency exchange rate
  public double getRate();

  // Sets the currency exchange rate
  public void setRate(double rate);

  // Returns the error message
  public String getMessage();

  // Sets the error message
  public void setMessage(String message);

  // Returns an array with supported currencies
  public String[] getSupportedCurrencies();
}
```

Extend the portlet class with the application logic

The `CurrencyConverter` portlet class includes two methods which are of interest for us:

- The `doView()` method is called when the portlet receives a render request. It only forwards to the JSP page which renders the portlet content. The wizard already created the code for us, so we do not need to change it.
- The `processAction()` method is called by the portlet container when an action is received. Our portlet receives the submit actions of the HTML form, so this method has to process them. The HTML form includes two submit actions: The *Calculate* action indicates that a currency conversion has to happen, and the *Reset* action resets all HTML form fields.

The implementation of the portlet's `processAction()` method is listed in Example 15-38.

Example 15-38 Implementation of the CurrencyConverter.processAction() method

```
// process a portlet action request
public void processAction(ActionRequest request, ActionResponse response)
```

```
  throws PortletException, java.io.IOException {

if (request.getParameter(FORM_CALCULATE) != null) {

  // retrieve the portlet's session bean
  CurrencyConverterSessionBean sessionBean = getSessionBean(request);

  if (sessionBean != null) {

    // reset the (error) message attribute
    sessionBean.setMessage(null);

    // read currency selections and amount from HTML form and add them
    // to session bean
    sessionBean.setAmount(
        Double.parseDouble(request.getParameter(FORM_AMOUNT)));
    sessionBean.setSourceCurrency(request.getParameter(FORM_SRC_CURR));
    sessionBean.setTargetCurrency(request.getParameter(FORM_TGT_CURR));

    try {
      // call currency conversion web service to retrieve the current
      // exchange rate
      ConvertorServerLocator wsLocator = new ConvertorServerLocator();
      String wsURL = request.getPortletSession().getPortletContext()
          .getInitParameter("ws.url.currencyconverter");

      wsLocator.getConvertorServerPort(new URL(wsURL))
          .getExchangeRate(
              sessionBean.getSourceCurrency(),
                sessionBean.getTargetCurrency());
      double exchangeRate = Double.parseDouble(
        wsLocator.getConvertorServerPort(new URL(wsURL))
            .getExchangeRate(
                sessionBean.getSourceCurrency(),
                sessionBean.getTargetCurrency()));

      // set calculated session bean attributes
      sessionBean.setRate(exchangeRate);
      sessionBean.setConvertedAmount(
          sessionBean.getAmount()  * exchangeRate);
    } catch (Exception e) {
      System.err.println("CurrencyConverter exception: " + e);

      // set the error message if the converter web service is unavailable
      sessionBean.setMessage(
  "The conversion service is temporarily unavailable! Please try again later.");
    }
  }
}
else if (request.getParameter(FORM_RESET) != null) {
  CurrencyConverterSessionBean sessionBean = getSessionBean(request);

  // reset all attributes in the portlet session
  sessionBean.setSourceCurrency("");
  sessionBean.setTargetCurrency("");
```

```
    sessionBean.setAmount(0);
    sessionBean.setRate(0);
    sessionBean.setConvertedAmount(0);

    sessionBean.setMessage(null);
  }
}
```

Notes to the implementation of the `processAction()` method:

- The input from the HTML form is accessed through parameters of the passed `PortletRequest` object. The request parameters include the form action which was selected by the user, and the field values entered by the user. The method identifies the form action (either FORM_CALCULATE or FORM_SUBMIT) and takes the appropriate steps.
- The method sets all attributes of the session bean, the submitted form fields as well as the calculated results. This allows the JSP to view all attributes the way the user entered them.
- We use the Web Service Java proxy (created in 15.4.4, “Creating Java client proxies for Web Service interfaces” on page 469) to call the currency conversion Web Service. To keep flexible with respect to the actual service endpoint used we do not take the service endpoint defined in the WSDL of this Web Service. Instead, we use a special method of the locator class which passes the service endpoint URL. The actual value of the URL is defined as portlet initialization parameter and retrieved as shown in the **bold** statement in the example.

  You need to add the initialization parameter in the Portlet Deployment Descriptor as shown in Figure 15-41 on page 480. You add a parameter `ws.url.currencyconverter` and set it to the Web Service URL, for example:

  `http://`*`Your_Apache_server`*`/ConvertorServer.php`
- In case the call to the currency conversion Web Service is not successful (for example, because the Web Service is not available, or because the interface changed), the session bean’s error message attribute is set.
- If the user selects the FORM_RESET action, the attributes in the session bean are reset.

Modify the portlet JSP to show the HTML form and result

The JSP of our portlet has to fulfill the following responsibilities:

- Display the HTML form.
- Show the conversion result after the user submits the form.
- Display an error message if the conversion is not successful.

Example 15-39 shows the main parts of the JSP. You can find the full source code in the additional materials.

Example 15-39 Content of CurrencyConverterView.jsp portlet view

```
[...]
<% CurrencyConverterSessionBean sessionBean = (CurrencyConverterSessionBean)
     renderRequest.getPortletSession().getAttribute(
       CurrencyConverter.SESSION_BEAN);
%>

<FORM method="POST" action="<portlet:actionURL/>">

[...include form input fields (currency selection lists, text field for amount...]
<P>
  <INPUT name="<%= CurrencyConverter.FORM_CALCULATE %>" type="submit"
         value="Calculate" />
  <INPUT name="<%= CurrencyConverter.FORM_RESET %>" type="submit" value="Reset" />
</P>

</FORM>

<% if (sessionBean.getMessage() != null) { %>

  <FONT color="red"><%= sessionBean.getMessage() %></FONT>

<% }
   else if (sessionBean.getConvertedAmount() > 0.1) {
%>

  Conversion:
  <B><%= sessionBean.getConvertedAmount() %>
     <%= sessionBean.getTargetCurrency() %></B>
  [exchange rate: <%=sessionBean.getRate()%>]

<% } %>
```

Notes to the implementation of the JSP file:

- First, the portlet retrieves the portlet session bean. The attributes of the session bean have been set by the portlet class when it processed the portlet's action request and before it called the JSP (in the render method).
- An HTML form is defined. The portlet does not have an URL assigned, so the `action` attribute is instead set to the `portlet:actionURL` tag.

  The `porlet:actionURL` tag is replaced at runtime by the portlet container with a link that calls the action handler of the corresponding portlet class. For our portlet, the `processAction()` method of the `CurrencyConverter` portlet class is called. So the HTML form and the form-processing code are linked together.
- The form input fields are defined as HTML `<INPUT>` and `<SELECT>` elements (the implementation is not shown in this example). The `name` values of the Calculate and Reset buttons are used by the portlet action handler to identify which one of these actions was selected by the user.
- If the error message attribute of the session bean has been set, it is displayed below the form. Otherwise the conversion result and the exchange rate are shown.

Run the portlet in the test environment

You can find instructions to add the portlet project to the WebSphere Portal V5.1 Test Environment in "Run the portlet in the portal test environment" on page 484. If you already performed the steps which are listed there, you do not need to repeat them. Instead, it is sufficient to republish the test environment (go to the Servers view in RAD, right-click the test environment and select **Publish**) and restart the test environment server.

You will see the foreign exchange calculator portlet, when you access the portal page in your Web browser using the URL:

```
http://localhost:9081/wps/portal
```

Enter the required field values and select **Calculate** to display the converted amount.

15.4.7 Creating the mortgage accounts portlet

The mortgage accounts portlet integrates characteristics of the other portlets demonstrated in this book, and goes beyond the other portlets in the following respect:

- **Access of local data and services**

  Figure 15-26 on page 458 shows the data flow of this portlet. The DB2 PORTALDB database is accessed to retrieve portal user data. The interfaces to the IMS CMR database and the company's accounting system is realized with DADX Web Services.

- **Interaction with user**

  The portlet requests information about new mortgage accounts from the user via HTML forms.

- **Personalization**

  The portlet is personalized like the credit score portlet (and as opposed to the foreign exchange calculator portlet). Each user which is logged in to the portal views personalized content (customer data and accounting information of the user).

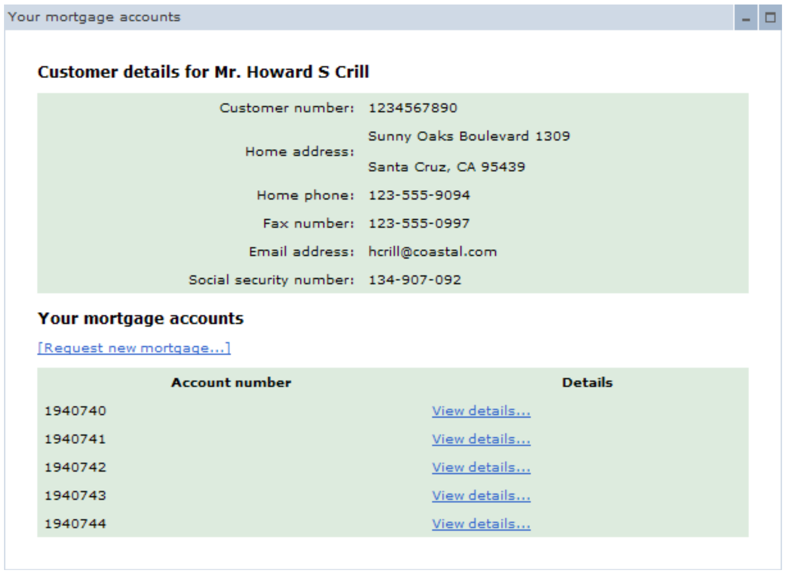
- **Workflow**

  The portlet implements a simplified workflow model. User actions trigger the output of different content to the user.

The portlet implements the following scenario:

1. When a customer logs in to the portal, the mortgage accounts portlet shows customer contact information (user name, home address, phone numbers, and so on), and lists all mortgage accounts of this customer.
2. The customer can view the details (mortgage amount, monthly payment, current balance, and so on) of each mortgage account.
3. The customer can request a new mortgage. For this purpose, the customer submits the details of the requested mortgage, and the portlet forwards the request to the accounting system. If the reply from the accounting system is positive, the portlet shows the new account number.

Figure 15-46 on page 492 shows the main page of the portlet with the customer data and the account list, as well as the URL links to either view the details of an existing account, or to request a new account.



Your mortgage accounts

Customer details for Mr. Howard S Crill

Customer number:	1234567890
Home address:	Sunny Oaks Boulevard 1309 Santa Cruz, CA 95439
Home phone:	123-555-9094
Fax number:	123-555-0997
Email address:	hcrill@coastal.com
Social security number:	134-907-092

Your mortgage accounts

[Request new mortgage...]

Account number	Details
1940740	View details...
1940741	View details...
1940742	View details...
1940743	View details...
1940744	View details...

Figure 15-46 Picture of the mortgage accounts manager portlet

The portlet uses a separate JSP to display the information provided in the different states of the portlet. Figure 15-47 shows the abstract workflow:

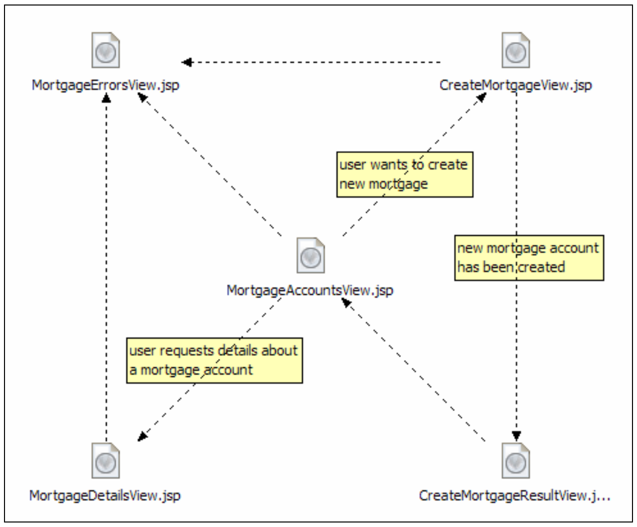

MortgageErrorsView.jsp
CreateMortgageView.jsp
user wants to create new mortgage
new mortgage account has been created
MortgageAccountsView.jsp
user requests details about a mortgage account
MortgageDetailsView.jsp
CreateMortgageResultView.j...

Figure 15-47 Flowchart of mortgage accounts manager portlet

- The MortgageAccountsView.jsp page displays the main page of the portlet, containing overall customer and accounting information. Figure 15-46 shows the rendered output of this JSP.
- If the user chooses to see the details of a mortgage account, the MortgageDetailsView.jsp page displays the mortgage account attributes as well as a link to go back to the main page.
- When the user selects the **Request new mortgage** link from the main page, he is forwarded to the CreateMortgageView.jsp page where he can enter the mortgage attributes.

- Once the mortgage request is processed successfully, the portlet uses the CreateMortgageResultView.jsp page to show the new account number. The user can go back to the main page using a link.
- In case a problem occurs, the user is forwarded to the MortgageErrorsView.jsp page which displays information about the problem.

As it is the case for all portlets in our example, this one follows the MVC principle. The separation of the portlet's components into the different roles is displayed in Figure 15-48:

- The `MortgageRequest` portlet class is the single **Controller** of this portlet. Each request coming from the user (portlet links selected by the user, submitted forms) is processed by this class. It forwards to the user to the appropriate JSP based on the action selected by the user.
- The JSP pages described previous in this chapter comprise the portlet's **View**. They render the HTML content of the portlet which is forwarded to the user.
- The `MortgageRequestSessionBean` as **Model** is used to pass session attributes from the portlet controller to the JSP pages.

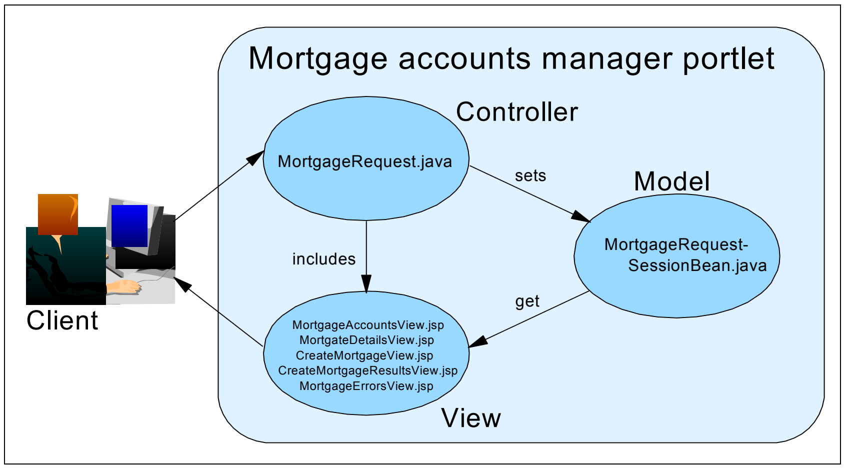


Figure 15-48 Mortgage accounts manager portlet application

This chapter demonstrates how to create this portlet in RAD. Some of the required tasks are already explained in 15.4.5, "Creating the credit score portlet" on page 475 and in 15.4.6, "Creating the foreign exchange calculator portlet" on page 484, and a complete step-by-step introduction goes beyond the context of this book, so this chapter concentrates on the setup of the portlet's workflow. The complete source code of this portlet is included in the additional materials. You can import the portlet project into your RAD workspace and introspect the portlet code from there.

In this chapter, the following topics are described:

- Create a new portlet
- Modify the session bean class
- Implement workflow in portlet class
- Implement user interface in JSPs
- Run the portlet in the test environment

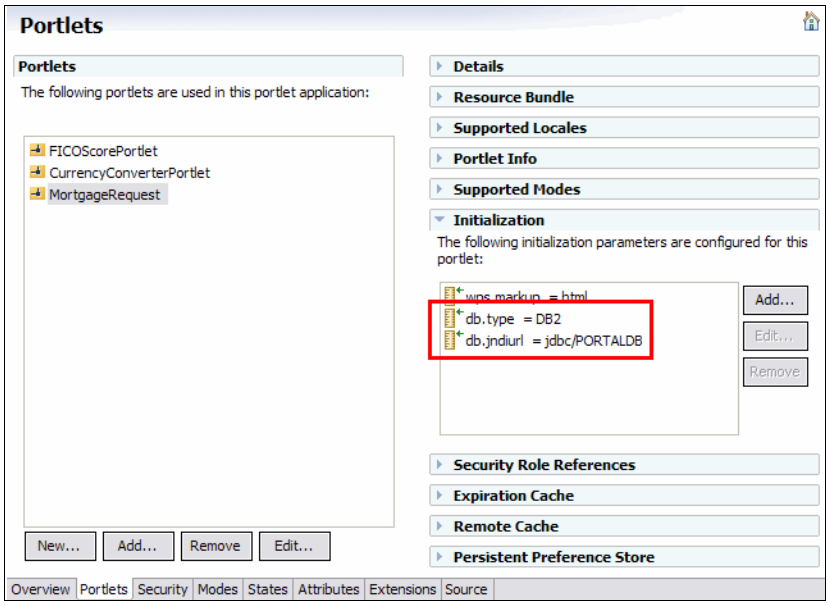
Create a new portlet

The creation of the mortgage accounts manager portlet follows the instructions explained in 15.4.5, "Creating the credit score portlet" on page 475. Open the New portlet wizard by selecting **File** → **New** → **Other** → **Portal** → **Portlet** and perform the steps which are demonstrated in that chapter. When you enter the **Portlet settings** on the third page, use the following values:

- The portlet name is *MortgageRequestPortlet*.
- Enter *Your mortgage accounts* as display name (this text will appear in the portlet title).
- The package prefix is *com.itso.sg247259.portlets*.
- The class prefix is *MortgageRequest*.

On the **Actions and preferences** page check the options Add action request handler and Add form sample. The wizard performs the following actions:

- The portlet deployment descriptor of our ITSOBankPortlets portlet application is extended with an entry for the new *MortgageRequest* portlet. Similar to the FICO score portlet, the mortgage accounts manager portlet accesses the local PORTALDB database using Data Access Objects. For this reason, the data source information is passed in the same way through portlet initialization parameters. As demonstrated in "Create a DB connector factory" on page 479, the parameters **db.type** and **db.jndiurl** need to be added to the portlet deployment descriptor (see also Figure 15-49).



```
wps.markup  = html
db.type     = DB2
db.jndiurl  = jdbc/PORTALDB
```

Figure 15-49 DAO initialization parameters for mortgage accounts manager portlet

- A new class, `com.itso.sg247259.portlets.MortgageRequest` is created. This class extends the `GenericPortlet` class and acts as the portlet **Controller**. We're going to extend this class with the portlet functionality in "Implement workflow in portlet class" on page 496.
- Another new class, `com.itso.sg247259.portlets.MortgageRequestSessionBean`, is the portlet **Model**. We add all attributes which are exchanged between the portlet controller and the portlet JSPs in "Modify the session bean class" on page 494.
- A single JSP page which represents the portlet view is created. We replace this page named `MortgageRequestView.jsp` with the JSP pages shown in Figure 15-48 on page 493.

Modify the session bean class

The portlet's session bean is, as its name indicates, persistent within the context of a portlet user session (that is, a series of interactions between the portal user and the portlet). Any user-related data can be stored in the session bean.

We use the session bean to store the portlet **Model**, that is, store the data which is retrieved from the local database, returned by service calls, entered by the user, and displayed in the rendered portlet pages, which includes the following information:

- Attributes of the portal user, represented by the `PortalUser` class. Portal user attributes are the user name of the authenticated user, and the corresponding ITSOBank customer number.
- Additional customer information which is read from the local PORTALDB database and displayed on the main page of the mortgage accounts portlet (customer name, address, and contact information).
- A list of mortgage account numbers owned by the portal user. This list is retrieved from the accounting system and also displayed on the portlet's main page.
- A mortgage account number, which identifies an existing mortgage account for which details are requested by the user, or a new mortgage account number returned by the accounting system as result of the creation of a new mortgage account.
- The attributes of an existing mortgage account which are displayed on the mortgage account details page of the portlet.
- A (potential) error message which is displayed on the portlet's error page.

The interface of the modified session bean class `MortgageRequestSessionBean` is shown in Example 15-40. You can find the complete implementation of this class in the additional resources.

Example 15-40 Interface of the MortgageRequestSessionBean class

```
package com.itso.sg247259.portlets;

// The session bean of the mortgage accounts manager portlet.
public class MortgageRequestSessionBean;

  // @return Returns the existing mortgage account numbers.
  public String[] getAccountNumbers();

  // @param accountNumbers The existing mortgage account numbers.
  public void setAccountNumbers(String[] accountNumbers);

  // @return Returns the attributes of the customer record.
  public Customer getCustomerDetails();

  // @param customerDetails The attributes of the customer records.
  public void setCustomerDetails(Customer customerDetails);

  // @return Returns the error message.
  public String getMessage();

  // @param message The error message to set.
  public void setMessage(String message);

  // @return Returns details of a mortgage account.
  public MortgageAccountDetailsRow getMortgageAccount();

  // @param mortgageAccount Details of a mortgage account to set.
  public void setMortgageAccount(MortgageAccountDetailsRow mortgageAccount);

  // @return Returns the portal user object.
  public PortalUser getUser();

  // @param user The portal user object to set.
  public void setUser(PortalUser user);

  // @return Returns the mortgage account number.
  public String getDetailAccountNumber();

  // @param detailAccountNumber The mortgage account number to set.
  public void setDetailAccountNumber(String detailAccountNumber);
}
```

Implement workflow in portlet class

When we demonstrated the creation of the other portlets earlier in this chapter, we already introduced a few methods of the portlet class. For the mortgage account manager portlet we use this portlet class more heavily, so a short explanation of the purpose of this portlet class follows.

The portlet class is inherited from the `GenericPortlet` class which is specified in the JSP 168 standard (find more information in 4.2, “The standardization of portlets (Java standardization request - JSR-168)” on page 77). It fulfills (at least) the following purposes:

- It processes *action requests* initiated by the portlet container. An action request is received if either the client submitted a form of the portlet, or if the client selected a link from within the portlet. Both, the form and the link need to be designated in the JSP code by using the `portal:actionURL` tag. When processing this type of request, no portlet content is created.
- It processes *render requests* which are also received from the portlet container. A render request indicates that no (data) processing needs to happen by the portlet, but the portlet content has to be rendered. A portlet receives more render requests than action requests, since a render request always follows an action request (the content of the portlet has to be regenerated after an action is processed), but also if the portal page is reloaded because of other reasons (for example, the portlet view mode changes from default to maximized, or the action of a different portlet on this page has been processed and requires the portal page to be reloaded).

The `GenericPortlet` class provides two methods which are called when requests are received. The `processAction()` is called when the portlet receives an action request, and the `doView()` method is called in case of a render request. The context of the portlet (session, request, and response data) is passed as method arguments.

Note: JSR 168 portlets have different modes, namely a *view* mode, an *edit* mode (normally allow a user to change portlet settings), and a *help* mode to show portlet-related help information. In our scenario we use only the view mode, so all render requests are passed to the `doView()` method. The `GenericPortlet` class provides the `doEdit()` and `doHelp()` methods to render the portlet content in the other modes.

For our mortgage account manager portlet the `processAction()` method has to fulfill the following purposes:

1. It defines which portlet JSP page needs to be displayed next, based on the action which is stored in the portlet request (and ultimately coming from the user by selecting a link of the portlet, or submitting a form). The JSP page shown is saved as a portlet session attribute and defines the **session status**. This attribute is read by the `doView()` method when a render request is received by the portlet.

2. In case the user submitted a mortgage request form, the method calls the appropriate service of the accounting system which processes the mortgage request. The result of the service call is stored in the session bean.
3. If a user selects to show the details of a mortgage account, the method stores the account number in the session bean (the account number is included as a request parameter).

Note: The complete workflow of the portlet (deciding in which status the portlet is at a specific time, that is, which of the portlet's JSP pages needs to be rendered) is controlled by the `processAction()` method. The current status of a session is stored in a session attribute. The session status is persistent across multiple render requests, so the `doView()` method does always know which page to render at a time.

We demonstrate the implementation of the `processAction()` method in Example 15-41. The method first reads the session bean from the portlet session, and tests the request parameters for the user action. Depending on the action, it sets the session state, and executes the actions requested by the user.

Example 15-41 Implementation of the MortgageRequest.processAction() method

```
/**
 * Process an action request.
 */
public void processAction(ActionRequest request, ActionResponse response)
    throws PortletException, java.io.IOException {

  MortgageRequestSessionBean sessionBean = getSessionBean(request);

  if (sessionBean == null) {
    System.out.println("MortgageRequest: No session available!");
    return; // don't do anything
  }

  // reset error message
  sessionBean.setMessage(null);

  if (request.getParameter(ACCOUNTS_VIEW_JSP) != null) {
    // requested action is to show the accounts overview page
    request.getPortletSession().setAttribute(JSP_PAGE, ACCOUNTS_VIEW_JSP);
  }
  else if (request.getParameter(DETAILS_VIEW_JSP) != null) {
    // the details of a specific mortgage account need to be displayed;
    // change to the page showing the details
    request.getPortletSession().setAttribute(JSP_PAGE, DETAILS_VIEW_JSP);

    sessionBean.setDetailAccountNumber(request.getParameter(ACCOUNT_NUM));
  }
  else if (request.getParameter(CREATE_VIEW_JSP) != null) {
    // the user wants to submit a request for a new mortgage, so show
    // the mortgage request form
    request.getPortletSession().setAttribute(JSP_PAGE, CREATE_VIEW_JSP);
  }
  else if (request.getParameter(CREATE_FORM_SUBMIT) != null) {
    // the user submitted data for a mortgage request, so create the request
    // and show the results page
    callCreateMortgageAccount(request, sessionBean);
    request.getPortletSession().setAttribute(JSP_PAGE, RESULT_VIEW_JSP);
  }
  else if (request.getParameter(CREATE_FORM_CANCEL) != null) {
    // the user cancelled the creation of a mortgage request, so
    // we go back to the accounts overview page
    request.getPortletSession().setAttribute(JSP_PAGE, ACCOUNTS_VIEW_JSP);
  }
}
```

The doView() method has the purpose of rendering the actual content, dependent on the current status stored in the portlet session. It performs the following steps:

1. First, it identifies the current session status, by reading which JSP page needs to be shown. This information has been set by the processAction() method and is stored in the portlet session.
2. Based on the session status, it collects all data which needs to be displayed (for example, attributes of a specific mortgage account if the mortgage account details page has to be shown) by calling services and reading data from the local database. The method stores the data in the session bean.
3. Finally, the doView() method forwards to the appropriate JSP page. The JSP page is using the data stored in the session bean to render the portlet content.

Example 15-42 shows the implementation of the doView() method. Most notably, the JSP_PAGE session attribute contains the session status (that is, the name of the JSP page being shown). Also, the data which is displayed in the portlet is stored in the session bean, hence it is persistent across a user session. Data which does not change within a user session (for example, the name of the portal user, or the customer information) has to be read only once within a session.

Example 15-42 Implementation of the MortgageRequest.doView() method

```
/**
 * Serve up the <code>view</code> mode.
 */
public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

  // Set the MIME type for the render response
  response.setContentType(request.getResponseContentType());

  // check if portlet session exists
  MortgageRequestSessionBean sessionBean = getSessionBean(request);
  if (sessionBean == null) {
    response.getWriter().println("<b>NO PORTLET SESSION YET</b>");
    return;
  }

  // initially, no session status is set; we start with the accounts
  // overview page
  if (request.getPortletSession().getAttribute(JSP_PAGE) == null)
    request.getPortletSession().setAttribute(JSP_PAGE, ACCOUNTS_VIEW_JSP);

  String currentJspPage =
      (String) request.getPortletSession().getAttribute(JSP_PAGE);
  // identify the portlet user of this session (if that's not happened already)
  // and store information in sessionbean
  if (sessionBean.getUser() == null)
    getPortalUser(PortalUserInfo.getCurrentUsername(request), sessionBean);

  // set customer details of current user in portlet session bean
  if (sessionBean.getMessage() == null &&
      sessionBean.getCustomerDetails() == null)
    getCustomerRecord(sessionBean);

  if (ACCOUNTS_VIEW_JSP.equals(currentJspPage)) {
    // get list of mortgage accounts in case the accounts overview page is shown
    if (sessionBean.getMessage() == null)
      callListAccountNumbers(sessionBean);
  }
  else if (DETAILS_VIEW_JSP.equals(currentJspPage)) {
    // read details of a specific mortgage account and store it in the session
    // bean if the details page is shown
    if (sessionBean.getMessage() == null)
      callGetAccountDetails(sessionBean);
  }

  // switch to the error page if a message has been set
  if (sessionBean.getMessage() != null)
    request.getPortletSession().setAttribute(JSP_PAGE, ERROR_VIEW_JSP);

  // invoke the JSP to render
  PortletRequestDispatcher rd =
      getPortletContext().getRequestDispatcher(getJspFilePath(
          request, (String) request.getPortletSession().getAttribute(JSP_PAGE)));
  rd.include(request,response);
}
```

Implement user interface in JSPs

The JSP pages comprising the mortgage accounts manager portlet user interface are shown in Figure 15-47 on page 492. The implementation of these files is straightforward since the only purpose of them is to:

- Display the data requested by the portlet user.
- Show HTML forms (and links) to allow the portlet user to submit information to the portlet.

The implementation of fragments of the main JSP page, `MortgageAccountsView.jsp`, is shown in Example 15-43.

Example 15-43 Implementation of the MortgageAccountsView.jsp page

```
<%@ page session="false" contentType="text/html"                          1
    import="java.util.*,javax.portlet.*,com.itso.sg247259.portlets.*" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>          2
<portlet:defineObjects/>                                                  3

<%
  MortgageRequestSessionBean sessionBean =                                4
      (MortgageRequestSessionBean)renderRequest.getPortletSession()
           .getAttribute(MortgageRequest.SESSION_BEAN);
  com.itso.sg247259.dao.transfer.Customer customerDetails =
      sessionBean.getCustomerDetails();
  String[] accountNumbers = sessionBean.getAccountNumbers();
%>

<H3 style="margin-bottom: 10px">                                                    5
  Customer details for
  <%= customerDetails.getSalutation() + " " + customerDetails.getFirstname() + " "
     + customerDetails.getMiddleInitial() + " " + customerDetails.getLastName() %>
</H3>

<!-- table containing customer details -->
<TABLE width="100%" border="0" cellpadding="5" cellspacing="0">
  <TBODY>
    <TR>
      <TD align="right">Customer number:</TD>
      <TD align="left"><%= customerDetails.getCustNum() %></TD>
    </TR>
    <TR>
      <TD align="right">Home address:</TD>
      <TD align="left">
        <%= customerDetails.getAddress1() %><BR />
        <%= customerDetails.getAddress2() %><BR />
        <%= customerDetails.getCity() + ", " + customerDetails.getState() + " "
           + customerDetails.getZipCode() %>
      </TD>
    </TR>
    ...
  </TBODY>
</TABLE>

<H3 style="margin-bottom: 10px">Your mortgage accounts</H3>

<!-- link to 'Request new mortgage account' page -->
<P><A href="<portlet:actionURL>                                                     6
  <portlet:param name="<%= MortgageRequest.CREATE_VIEW_JSP %>" value="true" />
</portlet:actionURL>">[Request new mortgage...]</A></P>

<!-- table containing customer's mortgage accounts -->
<TABLE width="100%" border="0" cellpadding="5" cellspacing="0">
  <THEAD>
    <TR>
      <TH>Account number</TH>
      <TH>Details</TH>
    </TR>
  </THEAD>
  <TBODY>
<% for (int i = 0; i < accountNumbers.length; i ++) { %>                            7
    <TR>
      <TD><%= accountNumbers[i] %></TD>
      <TD>
        <A href="<portlet:actionURL>                                                8
          <portlet:param name="<%= MortgageRequest.DETAILS_VIEW_JSP %>"
                         value="true" />
          <portlet:param name="<%= MortgageRequest.ACCOUNT_NUM %>"
```

```
                         value="<%= accountNumbers[i] %>" />
        </portlet:actionURL>">View details...</A>
      </TD>
    </TR>
<% } %>
  </TBODY>
</TABLE>
```

Notes to Example 15-43 on page 499:

1. The first JSP directive sets the page's content type to HTML, and imports a number of Java packages whose classes are used further on in the code.
2. The second JSP directive imports the portlet tag library defined in the JSR 168 standard. The portlet tag library provides convenient access to portlet-specific functions (for example, the creation of portlet links, or access to the portlet session objects). All tags in the JSP file starting with the `portlet` prefix are defined in this tag library.
3. The `<portlet:defineObjects />` tag creates some Java objects from the portlet API, for example, the `renderRequest` object which is used later on.
4. The portlet's session bean is accessed through the portlet's render request object. The JSP page displays the user's customer details and mortgage account numbers, and this data is taken from the session object.
5. The customer attributes (user's full name, customer number, address, and so on) are read from the `customerDetails` bean and displayed in an HTML table.
6. The `portlet:actionURL` tag is used to create a link that triggers a call of the `processAction()` method of our portlet class. We add a parameter with the `portlet:param` tag to let the portlet class know that we want to show the *Request a new mortgage account* page if the user selects this link.
7. We traverse the `accountNumbers` array to display each of the user's mortgage account numbers in a separate HTML table row.
8. We create a link to the account details page for each listed mortgage account. For this purpose, we use the `portlet:actionURL` tag and pass the indicator for the details page, and the mortgage account number as parameters, using the `portlet:param` tag.

Note: This chapter does not contain listings of the other JSP pages of the mortgage account manager portlet because no new techniques or methods are introduced in those JSP pages. Check the additional materials for the source code of the complete portlet.

Run the portlet in the test environment

You can find instructions to add the portlet project to the WebSphere Portal V5.1 Test Environment in section "Run the portlet in the portal test environment" on page 484. If you already performed the steps which are listed there, you do not need to repeat them. Instead, it is sufficient to republish the test environment (go to the Servers view in RAD, right-click the test environment and select **Publish**) and restart the test environment server.

Enter the following URL in your Web browser:

`http://localhost:9081/wps/portal`

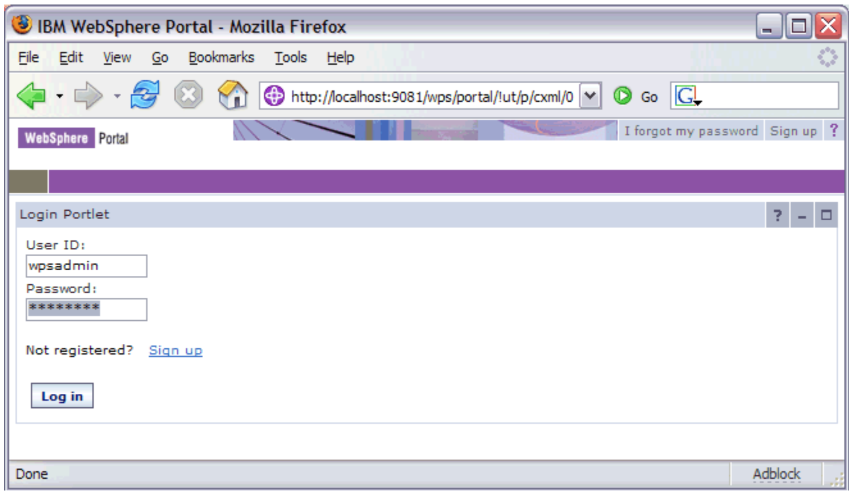
This URL displays the portal test page, including the mortgage account manager portlet. You first see the main page showing the customer information. From this page, you can start creating (requesting) a new mortgage account.

15.4.8 The whole picture

In this chapter we introduced a number of tools and techniques which allow you to aggregate a variety of services and data repositories into a SOA-enabled, state-of-the-art Web portal application. We demonstrated the integration of Web Services by generating Java Web Services client proxies from the WSDL of the services, the access of DB2 data servers using the DAO design pattern (this pattern is not restricted to DB2 though; it can be used for arbitrary relational databases, or even non-relational data repositories). And we showed the design and development of Web portlets with different complexities, using the MVC pattern to create the user interface and to implement a simple workflow.

When you went through the whole chapter and created the components and portlets which comprise the ITSOBank customer Web portal, you can run the portal application in the RAD WebSphere Portal V5.1 test environment and access the Web portal in your Web browser.

The first page you get after opening the Web portal URL contains the login portlet (see Figure 15-50). Enter the credentials of the Web portal administrative user (the default is *wpsadmin* for both, username and password) and select **Log in**.

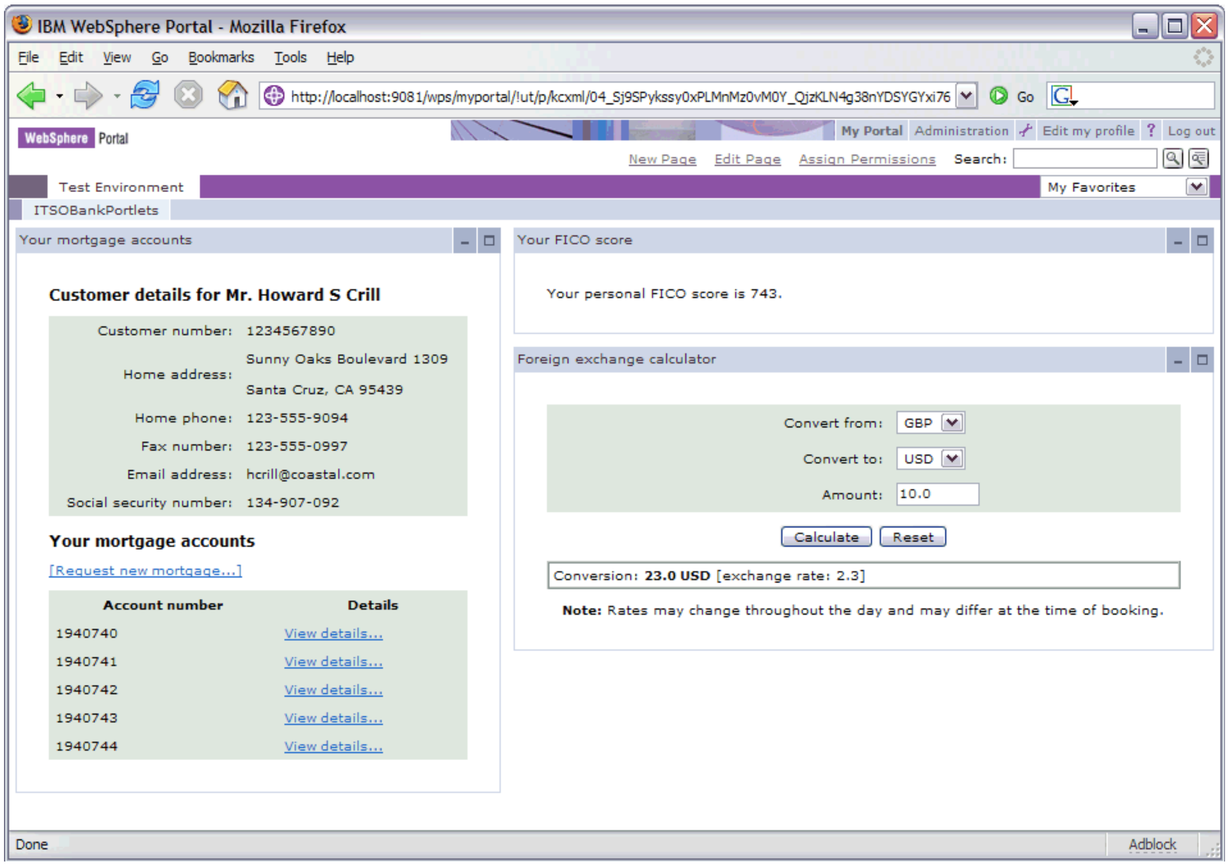

IBM WebSphere Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9081/wps/portal/!ut/p/cxml/0
Go
WebSphere Portal
I forgot my password
Sign up
Login Portlet
Not registered? Sign up
Log in
Done
Adblock

User ID: wpsadmin

Password: ********

Figure 15-50 ITSOBank portal login portlet

After a successful login the main page of the Web portal is displayed, as demonstrated in Figure 15-51 on page 503. It contains the user interface of the three portlets, and you can interact which each of them: performing currency conversions, reviewing the details of your existing mortgage accounts or requesting a new mortgage.


IBM WebSphere Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9081/wps/myportal/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzKLN4g38nYDSYGYxi76
Go
WebSphere Portal
My Portal Administration Edit my profile Log out
New Page Edit Page Assign Permissions Search:
Test Environment
My Favorites
ITSOBankPortlets
Your mortgage accounts
Your FICO score
Foreign exchange calculator
Calculate Reset
Done
Adblock

Customer details for Mr. Howard S Crill

Customer number:	1234567890
Home address:	Sunny Oaks Boulevard 1309 Santa Cruz, CA 95439
Home phone:	123-555-9094
Fax number:	123-555-0997
Email address:	hcrill@coastal.com
Social security number:	134-907-092

Your mortgage accounts

[Request new mortgage...]

Account number	Details
1940740	View details...
1940741	View details...
1940742	View details...
1940743	View details...
1940744	View details...

Your personal FICO score is 743.

Convert from:	GBP
Convert to:	USD
Amount:	10.0

Conversion: **23.0 USD** [exchange rate: 2.3]

Note: Rates may change throughout the day and may differ at the time of booking.

Figure 15-51 ITSOBank portal test page

This chapter provides only a short introduction into the portal technology. To gather additional information about portal development (for example, how to create a portal application which allows to get fine-grained control about the portal user experience by modifying portal themes and skins; or how to use an LDAP user directory for portal access and configure a role-based authorization concept) we recommend the redbook *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*, SG24-6681.

Part 6

SOA operations

In Part 6 we bridge the administration gap by providing information useful in administering and operating in SOA environments.

This part contains these chapters:

© Copyright IBM Corp. 2006. All rights reserved.

16

PHP client design

PHP is a recursive acronym for “Hypertext Preprocessor”. PHP is a powerful server-side scripting language designed for creating dynamic Web applications with non-static content. The PHP code can be either a stand alone program or it can be inserted inside HTML (Hypertext Markup Language) or XHTML (Extensible Hypertext Markup Language). The PHP syntax is similar to C, Java, and Perl.

This chapter discusses these topics:

- A brief introduction to PHP
- Implementing Web services with PHP
- Using native XML with PHP 5
- Connecting PHP to data servers
- Access an enterprise application using PHP
- Zend Core for IBM
- Conclusion

16.1 A brief introduction to PHP

Hypertext Preprocessor (PHP) is a powerful server-side scripting language for Web servers. PHP is popular for its ability to process database information and create dynamic Web pages. *server-side* refers to the fact that PHP language statements, which are included directly in your Hypertext Markup Language (HTML), are processed by the Web server. *Scripting language* means that PHP is not compiled. Since the results of processing PHP language statements is standard HTML, PHP-generated Web pages are quick to display and are compatible with most all Web browsers and platforms.

To “run” PHP scripts with your HTTP server, a PHP engine is required. The PHP engine is an open source product.

16.1.1 What PHP is

PHP code can easily access database files and output HTML, resulting in non-static, up-to-date Web pages. It is a technique similar to JavaServer pages (JSPs) or CGI binary programming. Also, PHP is an open-source project, there are many open-source PHP applications and code sample available on the Internet. Hundreds of ready-made applications written in PHP are available as shareware, and many commercial products employ it. PHP is considered reliable in terms of security.

Figure 16-1 shows the difference between standard static Web pages and dynamic Web pages using server-side PHP processing. In the first scenario (on the left), a standard URL request arrives at the Web server asking for the Web page:

```
http://www.example.com/index.html
```

The Web server sees this request and returns the HTML.

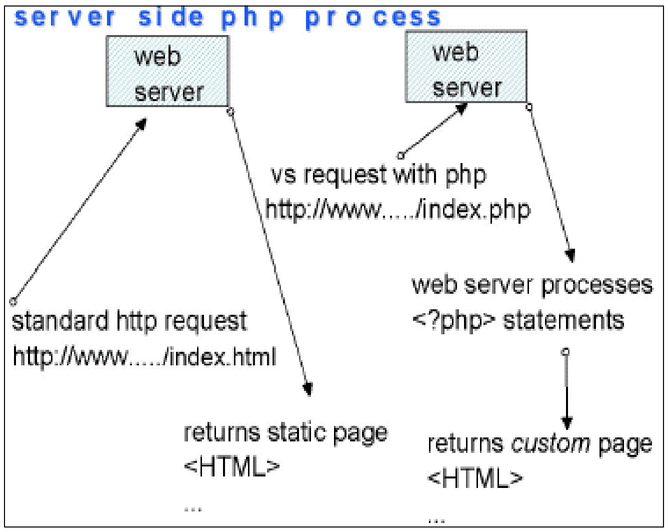


Figure 16-1 Left: Standard request for a Web page; Right: Request with PHP

Still looking at Figure 16-1 with the second scenario (on the right), the index.php file contains the special <?php tag that tells the Web server to process embedded PHP statements. After PHP processes those statements, it returns HTML statements to the Web server. Those statements are then sent back to the user included in the original HTML found in the file index.php. Because PHP statements can run a command or SQL statements, we can say that the Web page is *dynamically generated*, as opposed to our previous static HTML page.

PHP is popular for the following reasons:

- **Easy to use:** PHP is a scripting language included directly in HTML. There's no need to compile PHP programs or spend time learning tools that create PHP. You can simply insert statements and get quick turnaround as you make changes.
- **Fully functional:** The PHP language has built-in functions to access your favorite database.With PHP, your HTML pages can reflect current information by querying those databases, or you can use information about the user viewing your HTML Web page to customize the page specifically for that user. In addition to relational database support, PHP is a complete language that includes powerful functions to create classes for object-oriented programming and user flat file or Lightweight Directory Access Protocol (LDAP) databases. Plus, it includes a spell checker, Extensible Markup Language (XML) functions, image generation functions, and more.
- **Compatible and quick:** Because PHP generates plain HTML, it is compatible with all Web browsers and refreshes quickly.
- **Secure:** Although PHP is open source, it is a secure environment. One of its advantages (over JavaScript, for example) is that all that Web clients see is pure HTML. Because the logic of the PHP program is never exposed to the client, security exposures are reduced.

For information about the installation and configuration process of PHP (using Apache and IBM HTTP Server) refer to *IBM HTTP Server (powered by Apache) An Integrated Solution for IBM eServer iSeries Servers,* SG24-6716*.*

For more information about PHP programming refer to:

http://www.php.net

Note: The code samples in this chapter were executed using PHP Version 5.1.2 and IBM HTTP Server (powered by Apache) Version 6.0.2.

The PHP 5.1.2 (source code and Windows Binaries) can be download from the following link:

http://www.php.net/downloads.php

The IBM HTTP Server can be download from the product page:

http://www-306.ibm.com/software/webservers/httpservers

16.2 Implementing Web services with PHP

In this section we present three methods for implementing Web services using PHP:

- XML-RPC
- NuSOAP
- PHP 5's SOAP extension

16.2.1 XML-RPC

RPC (Remote Procedure Calls) are used to establish and facilitate transactions between two remote systems. Example of popular RPC implementation include Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA). XML-RPC is an established implementation of RPC that allows you to transport XML encoded data between two servers using HTTP.

XML-RPC is a specification and a set of implementations that allow software running in disparate operating systems, running in different environment to make procedure calls over the internet.

Figure 16-2 shows how the XML-RPC method works:

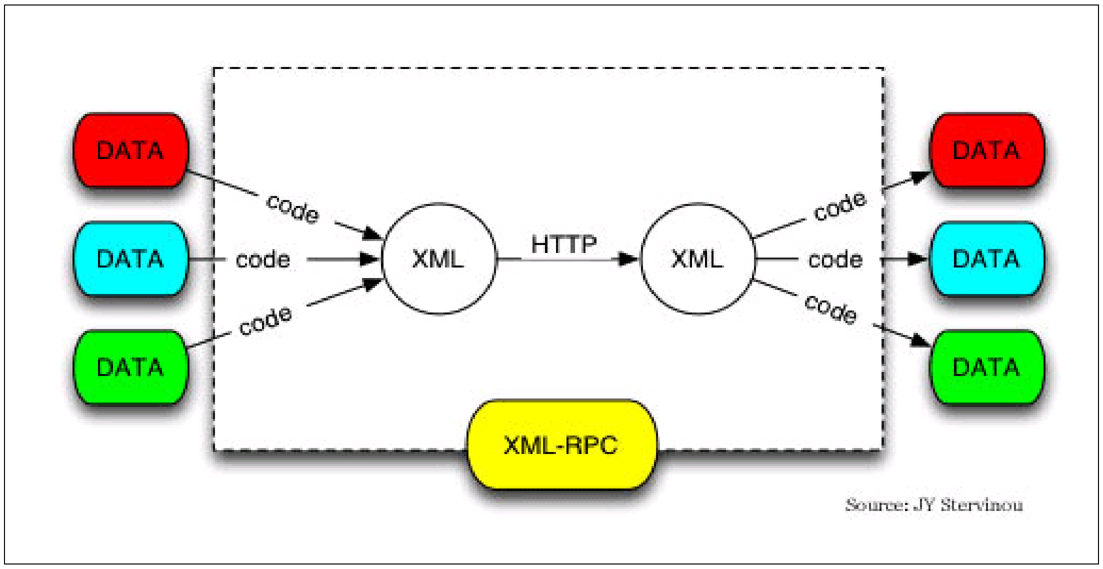


Figure 16-2 XML-RPC implementation

Note: To enable XML-RPC functionality, you must download the XML-RPC toolkit available at the following link:

`http://sourceforg.net/project/showfiles.php?group_id=34455`

Creating an XML-RPC Web service

The main include files we will be using are xmlrpc.inc (the base class library) and xmlrpcs.inc (the server class library). Here is how you implement a simple XML-RPC server using PHP. Example 16-1 shows how we first bring in both the client and server libraries using include statements.

Example 16-1 include statements

```
<?
include (“xmlrpc.inc”);
include (“xmlrpcs.inc”);
```

Example 16-2 on page 510 shows how we define a new function called ITSOcalcTax. This function will be the backbone of a Web service that will calculate the 8% sales tax for California, USA. A parameter which corresponds to the dollar amount, is passed into the function. The parameter is then converted to a scalar value. Once the calculation is completed, a response is created (using the xmlrpcresp class) returning the value of the sales class.

Example 16-2 ITSOcalcTax function

```
function ITSOcalcTax($param) {
$amont = $param->getParam(0);
```

```
$amountval=$amount->scalarval();
$taxcalc=#amountvar * .08;
return new xmlrpcresp(new xmlrpcval($taxcalc, “string”));
```

Example 16-3 shows that instantiation of the server and serialization of ITSOcalcTax back to the caller:

Example 16-3 Instantiation and serialization

```
$server=new
xmlrpc_server(array(“taxcalc.ITSOcalcTax=>array(“function”=>”ITSOcalcTax”)));
```

Example 16-4 shows how we instantiate our XML-RPC client which connect to our new server. The value of *$amount* passed to the server using the *xmlrpcmsg* object:

Example 16-4 instantiate the XML-RPC client

```
$format=new xmlrpcmsg(‘taxcalc.ITSOcalcTax’,
              array(new cmlrpcval($amount, “double”)));
$client=new xml-rpc_client(“/xmlrpc-server.php” , “localhost”, 80);
?>
```

Consuming an XML-RPC Web service

Now that we have built a server, the next step is to develop a client to call our Web service. First, we need to bring the base class library and define a scalar variable called *$amount* and assign it the value $20.00 (Example 16-5):

Example 16-5 Assign value

```
<?
include (“xmlrpc.inc”);
$amount = “20.00”;
```

Once a connection is made, the request is sent to the server. The response, which corresponds to the sales tax calculation, is passed along to $value. $value is then converted to a scalar variable and returned to the user. After receiving the $value, you can continue and manipulate this value for your needs. Example 16-6 shows the PHP code.

Example 16-6 Sending and receiving response from the server

```
$request=$client->send($format);
$value=$request->value();
print $value->scalarvar();

?>
```

XML-RPC is a simple, effective method of transmitting XML data. for more information consider reading Jean-Luc David article at:

http://www.xml.com/pub/a/ws/2004/03/24/phpws.html

Or visit the XML-RPC official Web site at:

http://www.xmlrpc.com

16.2.2 NuSOAP

SOAP is designed as an XML wrapper for Web services requests and responses. SOAP's strength lies in its use of namespaces, XML schema data types, and its flexibility with regard to transports. The disadvantage of SOAP is the fact that the specs and implementation is more complex, especially when you compare it to the simple XML-RPC approach. SOAP is the basics for Web services developers. It has been deeply integrated into Microsoft .NET and IBM WebSphere. Based on its popularity, Google and Amazon.com have both created SOAP-based Web services.

NuSOAP is a powerful API developed for the PHP platform. It allows you to build both Web service clients and servers. One of the great features of NuSOAP is the built-in WSDL support. installing the API is a snap: all you need is a PHP enabled server. The required libraries are contained in a file called nusoap.php.

Tip: you can download both the toolkit and the documentation at this link:

http://sourceforge.net/projects/nusoap

Creating a NuSOAP client

You can create a SOAP client with the NuSOAP easily. To illustrate how the SOAP client works, we call a Web service form XMethods.net. This Web site has a wide range of useful Web services you can access freely. In our particular example, we are querying the currency convertor Web service to determine the exchange rate between Canadian and American dollars.

The following examples provide the PHP code needed to access the currency convertor Web service.

First we need to bring the library file. We use the *require_once* command to do so. See Example 16-7.

Example 16-7 require_once

```
<?
require_once(“nusoap.php”);
```

The next step is to define where the WSDL is located and create an instance of the soapclient class to access the Web service (Example 16-8).

Example 16-8 locate WSDL and create client instance

```
$wsdl=”http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl”;
$client=new soapclient($wsdl,’wsdl’);
```

We then send to parameters through the SOAP client, the two countries we want to compare to get the currency exchange rate. We then call the *getRate* function and pass our parameters to get a response from the remote server. See Example 16-9.

Example 16-9 Send parameters and get response

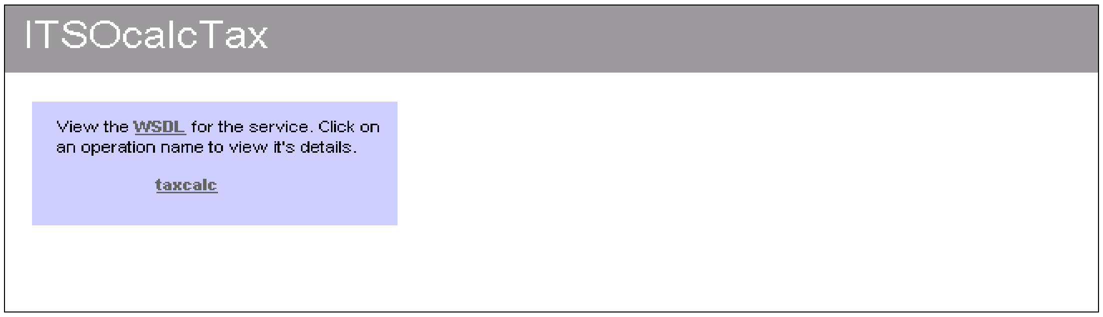
```
$param=array(
‘country1’=>’usa’,
‘country2’=>’canada’
);
echo $client->call(‘getRate’,$param);
```

```
?>
```

Now, after the execution of the echo command, the exchange rate is displayed on your browser.

Creating a NuSOAP Web service

The next example will demonstrate how to create a Web service using NuSOAP. One of the very useful features of NuSOAP is the Web service summery information and the dynamically generated WSDL files. Figure 16-3 shows an example for a Web service summary:


ITSOcalcTax

View the WSDL for the service. Click on an operation name to view it's details.

taxcalc

Figure 16-3 Web service summary

Just like the XML-RPC example, now we will create the same Web service called ITSOcalcTax using the NuSOAP API. Example 16-10 shows the import of the *nusoap.php* API and the definition of the service namespace.

Example 16-10 Definition of the service namespace

```
<?
require_once("nusoap.php");
$ns="http://localhost/nusoap";
```

Next, we instantiate the SOAP server and define the settings for our WSDL file such as the service name and the namespace (Example 16-11 on page 513).

Example 16-11 instantiate SOAP server and define WSDL settings

```
$server = new soap_server();
$server->configureWSDL('ITSOcalcTax',$ns);
$server->wsdl->schemaTargetNamespace=$ns;
```

Then, we register out PHP tax calculation function (Example 16-12).

Example 16-12 Register PHP tax calculation function

```
$server->register('taxcalc',
array ('amount' => 'xsd:string'),
array ('return' => 'xsd:string'),
$ns);
```

This step will make the server “aware” of the existence of the *taxcalc* method and the values that will pass to and from the method. In case of several methods, you must register each one separately. See Example 16-13.

Example 16-13 taxcalc function

```
function taxcalc($amount) {
$calc=$amount * .08;
return new soapval(‘return’,’string’,$calc);
```

And then we invoke the service, as shown in Example 16-14.

Example 16-14 Invoking the service

```
$server->service($HTTP_RAW_POST_DATA);
?>
```

The next step is to save the PHP file on the Web server (“localhost” on our example) and that’s it, now you are ready to provide a tax calculation Web service.

Creating a NuSOAP Web service consumer

In order to create consume the Web service that we have just created, we will present a small PHP program the connects to the server and displays the result.

First, we must instantiate the NuSOAP client object and pass the WSDL file with the relevant Web service definitions into the client. (assume that the service file name is *server.php*) Example 16-15 shows the code.

Example 16-15 instantiate the client

```
<?php
require_once(‘nusoap.php’);
$wsdl=”http://localhost/server.php$wsdl”;
$client=new soapclient($wsdl, ‘wsdl’);
```

The next step is to create a set of parameters and pass it into the Web service. Then, remotely call the *calctax* method. Example 16-16 shows the code.

Example 16-16 Remotely call the calctax method

```
$param=array(
‘amount=>’20.00’,
);
echo $client->call(‘calctax’ , $param);
?>
```

16.2.3 PHP 5 SOAP extension

PHP SOAP extension is the most popular PHP implementations of SOAP 1.1 and 1.2, developed by the PHP group.

PHP 5 SOAP extension is the first attempt to implement the SOAP protocol for PHP in C. It has some advantages over the implementations written in PHP itself (NuSOAP for example), the main one being execution time.

The SOAP extension implements a large subset of SOAP 1.1, SOAP 1.2, and WSDL 1.1 specifications. The key goal is to use the RPC features of the SOAP protocol. WSDL is used where possible in order to make the implementation of Web services more straightforward.

For more information about PHP SOAP extension refer to the specification and implementation documents at:

http://www.us3.php.net/soap

For PHP 5 SOA extension detailed examples, refer to 16.5, “Access an enterprise application using PHP” on page 528.

16.3 Using native XML with PHP 5

PHP has had XML support from its early days. While this was “only” a SAX based interface, it did at least allow parsing any XML document without to much hassle. Later XML support came with PHP 4 and the domxml extension. Then additional features like HTML, XSLT and DTD-validation were added to the domxml extension, but there were several issues. PHP XML developers have followed commonly used standards and rewritten almost everything regarding XML support with PHP 5. All the XML extensions are now based on the excellent libxml2 library by the GNOME project. This allows for interoperability between the different extensions, so that the core developers only need to work with one underlying library. Given the increasing importance of XML, the PHP developers have enabled more XML support by default. This means that you now get SAX, DOM and simpleXML enabled out of the box, which ensures that they will be installed on many more servers in the future.

In this section we discuss the following:

- DOM
- Validation
- XSLT

For presenting the PHP native XML capabilities we use, thorough this section, the XML file shown in Example 16-17 (the XML file name is “books.xml”).

Example 16-17 Sample XML file - books.xml

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<books>
    <item>
        <title>Redbooks weekly newsletter: Issue # 2</title>
        <link>http://ibm.redbooks.ibm.com/week/week2.php</link>
    </item>
    <item>
        <title>RedBook: Powering SOA with IBM data servers</title>
        <link>http://www.ibm.redbooks.com/redbooks/abstracts/sg247259.php</link>
    </item>
</books>
```

16.3.1 DOM

DOM (Document Object Model) is a standard for accessing XML documents trees, defined by W3C. The DOM specification defines the Document Object Model, a platform and language neutral interface that will allow programs and scripts to dynamically access and update the content and style of documents. The DOM provides a set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. The PHP DOM extension is completely based on the W3C standard, including method and property names.

Reading the DOM

The DOM library reads the entire XML document into the memory ad represents it as a tree of nodes. Figure 16-4 shows how XML DOM represents the file in Example 16-17 on page 515 using tree of nodes.

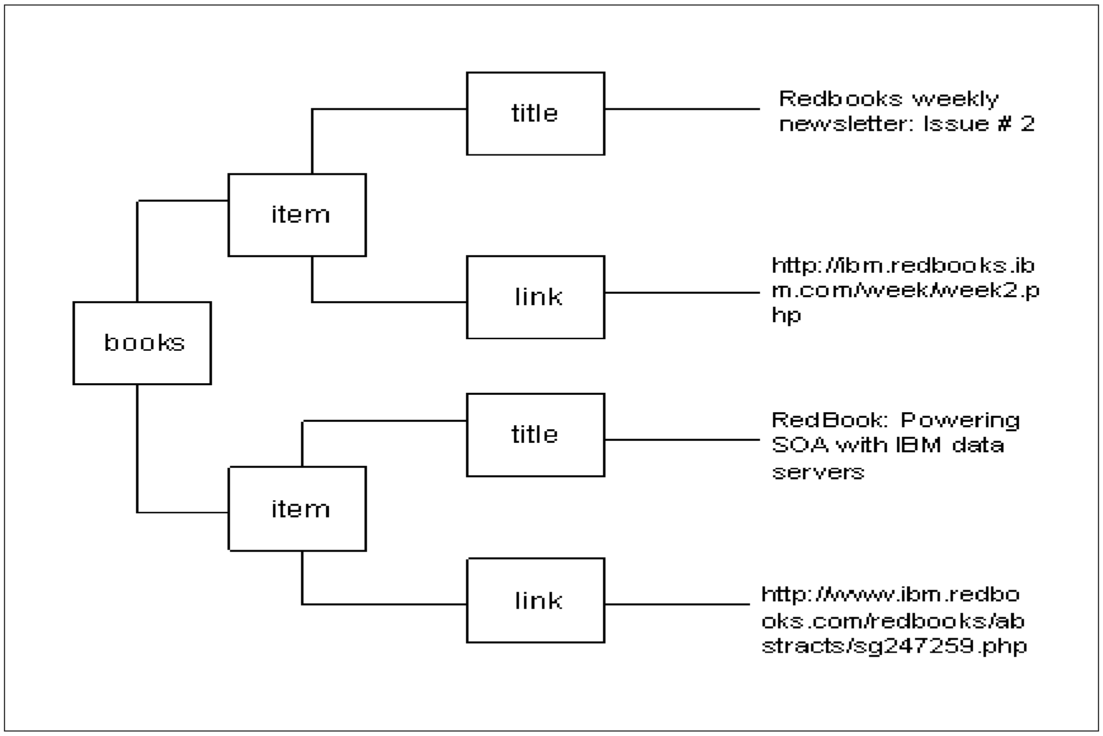

books
item
title
Redbooks weekly newsletter: Issue # 2
link
http://ibm.redbooks.ibm.com/week/week2.php
item
title
RedBook: Powering SOA with IBM data servers
link
http://www.ibm.redbooks.com/redbooks/abstracts/sg247259.php

Figure 16-4 XML DOM tree node representation

The *books* node at the to of the tree has two child *item* tags. within each book, there are *title* and *link* node, each have child text nodes that contains the text. The code to read the books XML file and display the contents using the DOM shown in Example 16-18:

Example 16-18 Reading the books XML file using the DOM

```
<?php
  $doc = new DOMDocument();
  $doc->load( 'books.xml' );

  $items = $doc->getElementsByTagName( "item" );
  foreach( $items as $item )
  {
  $titles = $item->getElementsByTagName( "title" );
  $title = $titles->item(0)->nodeValue;

  $links = $book->getElementsByTagName( "link" );
  $link = $links->item(0)->nodeValue;

echo "$title - $link \n";
  }
  ?>
```

The script starts by creating a new DOMdocument object and loading the books XML into that object using the load method. After that, the script uses the getElementsByName method to get a list of all of the elements with the given name.

Within the loop of the item nodes, the script uses the getElementsByName method to get the nodeValue for the title and link tags. The nodeValue is the text within the node. The script then displays those values.

As you can see in Example 16-19, a line is printed for each item block.

Example 16-19 DOM read results

```
Redbooks weekly newsletter: Issue # 2 - http://ibm.redbooks.ibm.com/week/week2.php
RedBook: Powering SOA with IBM data servers -
http://www.ibm.redbooks.com/redbooks/abstracts/sg247259.php
```

Writing XML using the DOM

Reading XML is only one part of the equation. What about writing it? The best way to write XML is to use the DOM. Example 16-20 shows how the DOM builds a new *books.*xml file.

Example 16-20 Writing a new "books.xml" with the DOM

```
?php
  $books = array();
  $books [] = array(
  'title' => 'Redbooks weekly newsletter: Issue # 2',
  'author' => 'IBM ITSO',
  'publisher' => "IBM"
  );
  $books [] = array(
  'title' => 'Powering SOA with IBM data servers',
  'author' => 'IBM ITSO',
  'publisher' => "IBM"
  );

  $doc = new DOMDocument();
  $doc->formatOutput = true;

  $r = $doc->createElement( "books" );
  $doc->appendChild( $r );

  foreach( $books as $book )
  {
  $b = $doc->createElement( "book" );

  $author = $doc->createElement( "author" );
  $author->appendChild(
  $doc->createTextNode( $book['author'] )
  );
  $b->appendChild( $author );

  $title = $doc->createElement( "title" );
  $title->appendChild(
  $doc->createTextNode( $book['title'] )
  );
  $b->appendChild( $title );
  $publisher = $doc->createElement( "publisher" );
  $publisher->appendChild(
  $doc->createTextNode( $book['publisher'] )
  );
  $b->appendChild( $publisher );

  $r->appendChild( $b );
  }

  echo $doc->saveXML();

?>
```

At the top of the script, the book's array is loaded with some example books. That data could come from the user or from a database. After the example books are loaded, the script creates a new DOMDocument and adds the root books node to it. Then the script creates an element for the author, title, and publisher for each book and adds a text node to each of those nodes. The final step for each book node is to re-attach it to the root books node.

The end of the script dumps the XML to the console using the saveXML method. You can also use the save method to create a file from the XML. The output of the script is shown in Example 16-21.

Example 16-21 Output from the DOM build script

```
<?xml version="1.0"?>
  <books>
  <book>
  <author>IBM ITSO</author>
  <title>Redbooks weekly newsletter: Issue # 2</title>
  <publisher>IBM</publisher>
  </book>
  <book>
  <author>IBM ITSO</author>
  <title>Powering SOA with IBM data servers</title>
  <publisher>IBM</publisher>
  </book>
  </books>
```

16.3.2 Validation

Validation of XML documents is getting more and more important. For example, if you get an XML document from some foreign source, you need to verify that it follows a certain format before you can process it. Luckily you can use one of the three widely used standards for doing this: DTD, XML Schema or RelaxNG.

- DTD is a standard that comes from SGML days, and lacks some of the newer XML features (like namespaces). Also, because it is not written in XML, it is not easily parsed and/or transformed.
- XML Schema is a standard defined by the W3C. It is very extensive and has taken care of almost every imaginable need for validating XML documents.

- RelaxNG was an answer to the complex XML Schema standard, and was created by an independent group. More and more programs support RelaxNG, since it's much easier to implement than XML Schema.

If you do not have existing or earlier schema documents, or overly complex XML documents, use RelaxNG. It is easier to write, easier to read, and many tools support it. There is even a tool called Trang, which automatically creates a RelaxNG document from sample XML document(s). Furthermore, only RelaxNG (and the aging DTDs) is fully supported by libxml2, although full XML Schema support is coming along.

The syntax for validating XML documents is quite simple:

```
$dom->validate('articles.dtd');
$dom->relaxNGValidate('articles.rng');
$dom->schemaValidate('articles.xsd');
```

At present, these all simply return true or false. Errors are dumped out as PHP warnings. This will be enhanced In one of the releases after PHP 5.0.0. The exact implementation is currently under discussion, but will certainly lead to better error reporting for parse errors and so on.

16.3.3 XSLT

XSLT is a language for transforming XML documents into other XML documents. XSLT is itself written in XML, and belongs to the family of functional languages, which have a different approach to that of procedural and object-orientated languages like PHP.

There were two different XSLT processors implemented in PHP 4: Sablotron (in the more widely used and known xslt extension), and libxslt (within the domxml extension). The two APIs were not compatible with each other, and their feature sets were also different.

In PHP 5, only the libxslt processor is supported. Libxslt was chosen because it is also based on libxml2 and therefore fits perfectly into the XML concept of PHP 5.

For more information about using native XML with PHP (including the usage of SimpleXML), refer to Christian Stocker's article *XML in PHP 5 - What's New?* available at:

http://www.zend.com/php5/articles/php-5-xmlphp.php

16.4 Connecting PHP to data servers

We have discussed the methods for implementing Web services and accessing XML documents using PHP. To complete the picture and to provide an end to end view of the SOA development process using PHP, we now discuss the ways to connect PHP to IBM data servers.

There are three PHP drivers that we can use to connect to IBM data servers:

- Unified ODBC (ext/odbc): Support all IBM data servers
- Extension for DB2 (ibm_db2): support IBM DB2 on all platforms
- PHP Data Objects(PDO): supports all IBM data servers

All three PHP drivers are offered under open-source licenses and available from:

http://www.php.net

In this section, we focus on DB2 for all platform. It is important to understand that *all* IBM data servers have the capabilities of communicating with PHP using the ODBC extension, or by implementing PHP Data Object (PDO).

16.4.1 Unified ODBC

To demonstrate some of the basic operations you can perform with PHP and DB2, we create a set of PHP scripts to help you manage a database table that contains data on a set of authors. First we create a table to hold our author data, then we write a PHP script that inserts a row to the AUTHOR table, and finally, we write a script that browses through existing authors.

Connecting to a database

To use this database, we need to insert some data into the table. We could issue some Data Manipulation Language (DML) statements, but because we have PHP installed, we create and use a simple PHP form for inserting new records in the database. All of the following PHP scripts use the Unified ODBC functions described in the PHP documentation.

Before we can insert data, we must create a database connection within a PHP script. Once we have confirmed that the connection is successful by returning a list of the tables with our user name, we can re-use that connection function within the rest of the scripts that we write.

The syntax for connecting to a database using PHP is shown in Example 16-22.

Example 16-22 Connecting to a database using PHP

```
int odbc_connect() (string dsn, string user, string password [, int cursor_type]);
```

Where:

dsn: The name of the database as registered in the DB2 catalog.

user: The name of the user that will connect to the database.

password: The password for user.

cursor_type: Optional arguments to specify cursor behavior.

Creating the author table

The AUTHOR table contains four columns: LAST_NAME, FIRST_NAME, MIDDLE_INITIAL, and AUTHOR_ID, a unique identifier generated by the database server that serves as the primary key of this table. The Data Definition Language (DDL) statement contained in the PHP script of Example 16-23 creates this table.

Example 16-23 Create the AUTHOR table

```
<?php
// connect to the database
$conn = odbc_connect('SAMPLE', 'db2inst1', 'ibmdb2');

// define our SQL
$sql = 'CREATE TABLE author (last_name VARCHAR(32) NOT NULL,
   first_name VARCHAR(32) NOT NULL,
   middle_initial VARCHAR(1),
   author_id INTEGER GENERATED ALWAYS AS IDENTITY,
   PRIMARY KEY (author_id))';

// issue our SQL statement directly
```

```
odbc_exec($conn, $sql);

// close the database connection
odbc_close($conn); ?>
```

Issuing INSERT statements

Once you have successfully connected to the database, you can start doing some more interesting work such as inserting, updating, and retrieving data. You can issue a simple SQL statement, one that contains no parameter markers for variable input--using the odbc_exec() function. The script of Example 16-24 inserts new rows into the AUTHOR table.

Example 16-24 Static INSERT statement

```
<?php
// include the dbconnect() and my_header() functions
include_once("db2lib.php");

echo(my_header"INSERT data into AUTHOR table");

$author_insert = "INSERT INTO author" .
"(last_name, first_name, middle_initial)" .
"VALUES('IBM', 'ITSO', ' ')";

$verbose = TRUE;
$dbconn = dbconnect($verbose);

if ($dbconn != 0) {
  // odbc_exec returns 0 if the statement fails; otherwise
  // it returns a result set ID
  $result = odbc_exec($dbconn, $author_insert);

  if ($result == 0) {
    echo("INSERT statement failed.");
    $sqlerror = odbc_errormsg($dbconn);
    echo($sqlerror);
  }
  else {
    echo("Successfully inserted one row.");
  }
}
else {
  echo("<p>Connection failed.</p>");
}
echo("</body></html>");
?>
```

SELECT statements and result sets

SELECT statements normally return multiple rows of data. When you call the odbc_exec() function for a SELECT statement, the function returns a result set identifier. A result set is an array consisting of 0 or more rows that match a database query; a result set identifier is simply a value that you pass to other functions to work with the rows in the result set.

Once we retrieve our result set identifier, we can retrieve the contents of the result set in a number of ways. One of the most convenient methods is to iterate over the odbc_fetch_array() function as demonstrated in Example 16-25.

Example 16-25 Simple SELECT statement

```
<?php
// include our custom function libraries
include_once("db2lib.php");
include_once("db2form.php");

echo(my_header('Simple SELECT statement'));

function display_authors($dbconn) {
  // select all rows from the AUTHOR table
  $select_stmt = 'SELECT last_name, first_name, middle_initial, author_id
    FROM author';

  if ($dbconn != 0) {
    // odbc_exec returns 0 if the statement fails;
    // otherwise it returns a result set ID
    $result = odbc_exec($dbconn, $select_stmt);

    if ($result == 0) {
      echo("SELECT statement failed.");
      $sqlerror = odbc_errormsg($dbconn);
      echo($sqlerror);
    }
    else {
      print '<table>
      <tr><th>Last</th><th>First</th><th>Initial</th><th>ID</th></tr>';
      while ($row = odbc_fetch_array($result)) {
        print '<tr><td>' . $row['LAST_NAME'] . '</td>';
        print '<td>' . $row['FIRST_NAME'] . '</td>';
        print '<td>' . $row['MIDDLE_INITIAL'] . '</td>';
        print '<td>' . $row['AUTHOR_ID'] . '</td></tr>';
      }
      print '</table>';
    }
  }
}

$verbose = TRUE;
$dbconn = dbconnect($verbose);

display_authors($dbconn);

echo('</body></html>');

// always close your database connection
odbc_close($dbconn);
?>
```

The script on Example 16-25 on page 522 defines a new function called display_authors(). The function displays a list of all authors in the AUTHOR table by iterating over the odbc_fetch_array() function in a while() loop. Each time the while() condition is evaluated, odbc_fetch_array() returns an array variable named $row representing the requested row. The fields of the array are named fields that map to the upper case column names that were requested in the SELECT statement. When there are no more rows to fetch from the result set, odbc_fetch_array() returns FALSE and the while() loop ends.

For more information about the Unified ODBC extension for PHP, refer to Dan Scott's article *Develop IBM Cloudscape and DB2 Universal Database applications with PHP* at:

http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0502scott/

16.4.2 ibm_db2 extension

The ibm_db2 extension gives you the most of general database management functions that allow you to connect to you database, execute SQL, and troubleshoot any problems you run into along the way. In this section we discuss about these functions.

Note: You must download the ibm_db2 extension before executing the following script. The extension can be downloaded from:

http://pecl.php.net/package/ibm_db2

To illustrate the use of the ibm_db2 extension function, we present a simple PHP script for connecting the database and retrieving data. The DB2-related functions are in **boldface**. Following the script are the descriptions of the functions that are used in the script.

Example 16-26 demonstrates the use of ibm_db2 extension.

Example 16-26 ibm_db2 extension

```
<?php
$database = 'ITSODB';
$user = 'db2admin';
$password = 'db2admin';

$conn = db2_connect($database, $user, $password);

if ($conn) {
   echo "Connection succeeded.<br />";

   $sql = "SELECT name FROM authors ORDER BY author_id";
   $stmt = db2_prepare($conn, $sql);
   db2_execute($stmt);
   $authorlist = array();
   $i=0;

   while (db2_fetch_row($stmt)) {
      $authorlist[$i] = db2_result($stmt, 0);
      echo "$authorlist[$i]<br />";
      $i += 1;
   }

   echo count($authorlist) . " author listed.<br /><br />";

   db2_close($conn);
}
else {
   echo "Connection failed.<br />";
   echo db2_conn_errormsg();
}
?>
```

Here are important function descriptions:

- **db2_connect** (string database, string user name, string password [, array options]): Creates a new database connection.
- **db2_prepare** (resource connection, string statement [, array options]): Prepares a SQL statement for execution. Parameter markers can be included to represent input, output, or input/output parameters.
- **db2_execute** (resource connection, string statement [, array options]): Executes a statement prepared by *db2_prepare*. It returns a TRUE or FALSE indicating success of failure.
- **db2_close** (resource connection): Closes a non-persistent database connection and frees the resource back to the database server.
- **db2_conn_errormsg** ([resource connection]): Returns an error message, including SQLSTATE and SQLCODE, for a failed connection attempt. if *db2_connect* returns FALSE, user this to retrieve a somewhat detailed explanation.

For more information about ibm_db2 extension refer to the extension documentation at:

http://us3.php.net/ibm_db2

16.4.3 PHP data objects (PDO)

PHP 5.1 is set to ship with a brand-new database connectivity layer known as PHP Data Objects (PDO). While PHP has always had very good database connectivity, PDO takes PHP to the next level.

Some of the design goals behind PDO are:

- To provide a consistent API for the common features found in most database APIs
- To be extensible, so that database vendor X can still expose feature Y and remain PDO compatible
- To provide a number of basic compatibility quirks, to make it easier to create cross-database compatible applications
- To NOT provide full abstraction or emulation of features (such as sequences) that are otherwise missing from a given database API. The PDO class is intended to give you consistent access to native features of the database, with minimal interference.
- To simplify the creation of PHP database drivers by centralizing the code that deals with the PHP internals (which are the hardest part to write)

That last point is quite important. PDO is modular in structure, separated into a common core that provides the API that you use in your scripts (PDO itself), and one or more driver extensions that bridge PDO to the native RDBMS client API libraries. DB2 users will want to use the PDO_ODBC driver, which boasts the following features:

- It was written from the ground-up to support ODBC V3 compliant drivers and driver managers. Support for DB2 specific features and optimizations was considered and incorporated as part of this design process--it was not an afterthought.
- It supports large objects and stored procedures has been tried and tested. Not only does it work, but it is very pleasant to use.
- Performance of DB2 access through the PDO_ODBC driver for simple fetches of 10,000 rows is approximately 10 times faster than the traditional PHP Unified ODBC extension. This dramatic difference is due to the light-weight forward-only cursor that is the default in PDO.

There are four key concepts that you need to be aware of, to get the most out of PDO. They are:

- Connections and connection management
- Transactions and auto-commit
- Prepared statements and stored procedures
- Errors and error handling

We look at them in more detail.

Connections and connection management

Connections are established by creating instances of the PDO base class. It doesn't matter which driver you want to use; you always use the PDO class name. The constructor accepts parameters for specifying the database source (known as the DSN) and optionally for the username and password (if any). The final parameter is used for passing additional tuning parameters through to PDO or the underlying driver--more on that shortly. Example 16-27 shows a short sample script that connects to DB2.

Example 16-27 Connect to DB2 using PDO

```
          try {
  $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2');
  echo "Connected\n";
} catch (Exception $e) {
  echo "Failed: " . $e->getMessage();
}
```

If the connection succeeds, you will see the message "Connected", otherwise, PDO will throw a PDOException explaining why the connection failed. Possible reasons include invalid parameters, incorrect user/password or even just that you forgot to load the driver.

Transactions and auto-commit

Now that you're connected via PDO, you should to understand how PDO manages transactions before you start issuing queries.

Transactions are typically implemented by "saving-up" your batch of changes to be applied all at once; this has the nice side effect of drastically improving the efficiency of those updates. In other words, transactions can make your scripts faster and potentially more robust (you still need to use them correctly to reap that benefit).

Unfortunately, not every database supports transactions, so PDO needs to run in what is known as "auto-commit" mode when you first open the connection. Auto-commit mode means that every query that you run has its own implicit transaction, if the database supports it, or no transaction if the database doesn't support transactions. If you need a transaction, you must use the PDO::beginTransaction() method to initiate one. If the underlying driver does not support transactions, a PDOException will be thrown (regardless of your error handling settings: this is always a serious error condition). Once you are in a transaction, you may use PDO::commit() or PDO::rollBack() to finish it, depending on the success of the code you run during the transaction.

When the script ends or when a connection is about to be closed, if you have an outstanding transaction, PDO will automatically roll it back. This is a safety measure to help avoid inconsistency in the cases where the script terminates unexpectedly—if you didn't explicitly commit the transaction, then it is assumed that something went awry, so the rollback is performed for the safety of your data.

Prepared statements and stored procedures

Prepared statements are so useful that PDO actually breaks the rule set out in Goal number 4: if the driver doesn't support prepared statements, PDO will emulate them.

Here are two examples of using prepared statements; Example 16-28 performs an insert by substituting a name and a value for the named placeholders.

Example 16-28 Repeated inserts using prepared statements

```
$stmt = $dbh->prepare("INSERT INTO AUTHORS (last_name, first_name, middle_initial)
VALUES (:last_name, :first_name, :middle_initial)");
$stmt->bindParam(':last_name', $last_name);
$stmt->bindParam(':first_name', $first_name);
$stmt->bindParam(':middle_initial', $middle_initial);

// insert one row
$last_name = 'ITSO';
$first_name = ‘IBM’;
$middle_initial = ‘I’;
$stmt->execute();

// insert another row with different values
$last_name = 'IBM';
$first_name = ‘ITSO’;
$middle_initial = ‘B’;
$stmt->execute();
```

Example 16-29 performs a select statement, using the alternative question mark placeholders.

Example 16-29 Fetching the data using prepared statements

```

           $stmt = $dbh->prepare("SELECT * FROM AUTHORS where last_name = ?");
if ($stmt->execute(array('one'))) {
  while ($row = $stmt->fetch()) {
    print_r($row);
  }
}
```

You may also bind parameters for output as well as input. Output parameters are typically used to retrieve values from stored procedures. Output parameters are slightly more complex to use than input parameters, in that you must know how large a given parameter might be when you bind it. If the value turns out to be larger than the size you suggested, an error is raised. Example 16-30 demonstrate calling a stored procedure with an output parameter.

Example 16-30 Calling stored procedure with an output parameter

```
          $stmt = $dbh->prepare("CALL sp_returns_string(?)");
$stmt->bindParam(1, $return_value, PDO_PARAM_STR, 4000);

// call the stored procedure
$stmt->execute();

print "procedure returned $return_value\n";
```

You may also specify parameters that hold values both input and output; the syntax is similar to output parameters. In Example 16-31, the string 'hello' is passed into the stored procedure, and when it returns, hello is replaced with the return value of the procedure.

Example 16-31 Calling a stored procedure with an input/output parameter

```
          $stmt = $dbh->prepare("CALL sp_takes_string_returns_string(?)");
$value = 'hello';
$stmt->bindParam(1, $value, PDO_PARAM_STR|PDO_PARAM_INPUT_OUTPUT, 4000);

// call the stored procedure
$stmt->execute();

print "procedure returned $value\n";
```

Errors and error handling

PDO offers three different error handling modes to suit different styles of programming:

PDO_ERRMODE_SILENT

This is the default mode. PDO will simply set the error code for you to inspect using the errorCode() and errorInfo() methods on both the statement and database objects; if the error resulted from a call on a statement object, you would invoke the errorCode() or errorInfo() method on that object. If the error resulted from a call on the database object, you would invoke those methods on the database object instead.

PDO_ERRMODE_WARNING

In addition to setting the error code, PDO will emit a traditional E_WARNING message. This setting is useful during debugging/testing, if you just want to see what problems occurred without interrupting the flow of the application.

PDO_ERRMODE_EXCEPTION

In addition to setting the error code, PDO will throw a PDOException and set its properties to reflect the error code and error information. This setting is also useful during debugging, as it will effectively "blow up" the script at the point of the error, very quickly pointing a finger at potential problem areas in your code (remember: transactions are automatically rolled back if the exception causes the script to terminate).

Exception mode is also useful because you can structure your error handling more clearly than with traditional PHP-style warnings, and with less code/nesting than by running in silent mode and explicitly checking the return value of each database call.

For more information about PDO implementation refer to Wez Furlong article *Connect PHP to DB2 and Cloudscape via PDO* at:

`http://www.ibm.com/developerworks/db2/library/techarticle/dm-0505furlong`

16.5 Access an enterprise application using PHP

In this section we present the implementation of one part of our usage scenario, the *currency conversion application*.

16.5.1 Lab environment description

For implementing the access to the enterprise application using PHP we used the following:

- DB2 for z/OS Version 8
- DB2 Connect for Windows version 8.2
- Apache HTTP server for Windows version 2.0.55
- PHP for Windows version 5.1.2
- Microsoft Internet Explorer® version 6.0 (service pack 1)

For more information about Apache and PHP installation and configuration, refer to *Developing PHP Applications for IBM Data Servers*, SG24-7218.

16.5.2 Usage scenario

Referring to the usage scenario shown on Figure 16-5 on page 529, the PHP implementation will concentrate on the currency convertor database cube.

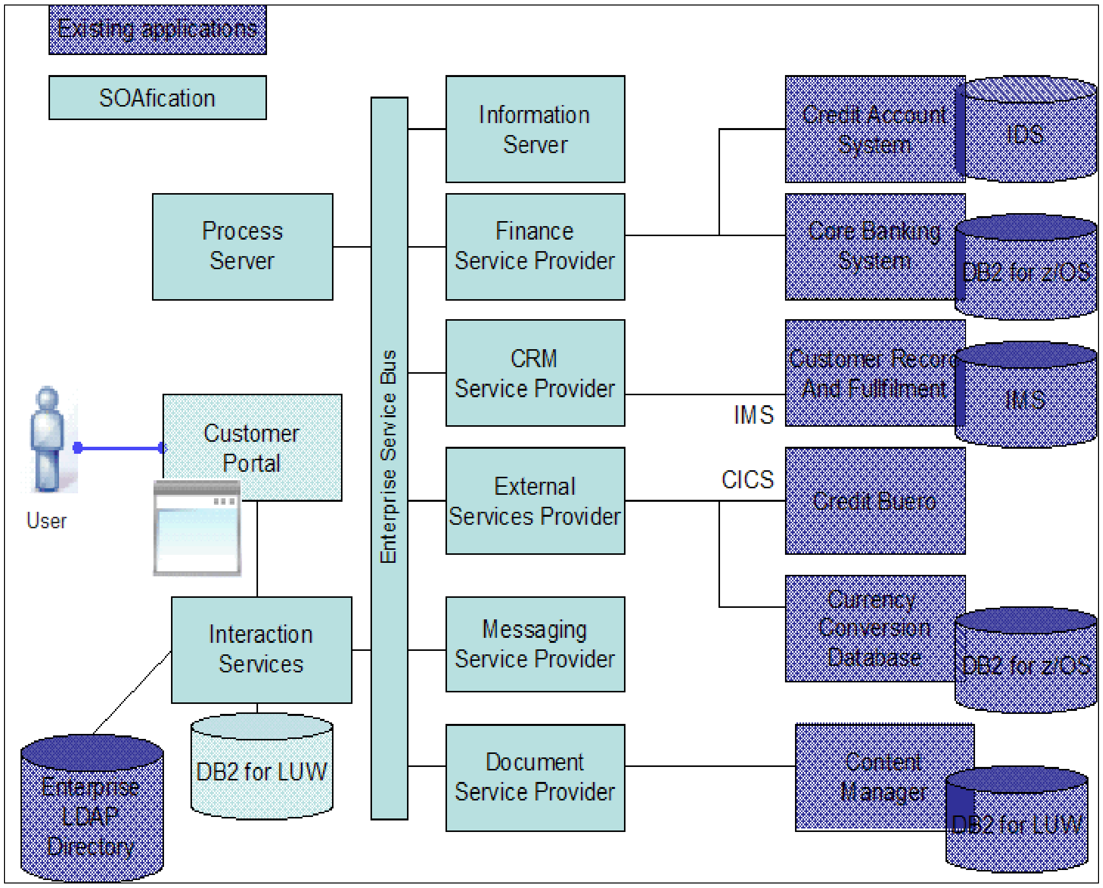


Figure 16-5 Usage scenario

The ITSO bank have a simple currency convertor application which accesses, via stored procedure, to DB2 table (on z/OS platform) which contains the exchange currency data. To publish the exchange data as a Web service we created a PHP program that calls the DB2 stored procedure with 2 parameters: *country1* and *country2* which represent the source and target countries for the exchange rate. The stored procedure accesses the DB2 table, that contains all the exchange rates of the countries that ITSO bank support, and returns the *rate* parameter which is the exchange rate between the countries requested on the input parameters. The next step was to create a PHP Web service consumer program within an HTML page, which will allow the portlet we presented on the previous chapters to access the currency conversion application in a simple way.

The DB2 table has three columns: the source county, the target country, and the exchange rate as shown on Example 16-32:

Example 16-32 Rate table structure

```
CREATE TABLE ITSO.RATE_TABLE (
       COUNTRY1 CHAR(3) NOT NULL ,
       COUNTRY2 CHAR(3) NOT NULL ,
       RATE DECIMAL(5,2) NOT NULL )
       IN DBSOA.TS1
COMMENT ON TABLE ITSO.RATE_TABLE IS 'table for exchange_rate';
```

The stored procedure that accesses this table is presented in Example 16-33 on page 530.

Example 16-33 itso.exchange_rate stored procedure

```
CREATE PROCEDURE ITSO.EXCHANGE_RATE (IN PCOUNTRY1 CHAR(3), IN PCOUNTRY2 CHAR(3),
INOUT RATE DECIMAL(5,2) )
      LANGUAGE SQL
      BEGIN
      SELECT RATE INTO RATE FROM ITSO.RATE_TABLE
      where COUNTRY1 = PCOUNTRY1
      and   COUNTRY2 = PCOUNTRY2;
      END
```

The next step was to build a PHP Web service provider program that call the stored procedure. We will present the program step by step:

First we have to connect DB2 and call the stored procedure. For doing that, we used the ibm_db2 extension (see 16.4.2, “ibm_db2 extension” on page 523) as shown in Example 16-34.

Example 16-34 Connect DB2 using ibm_db2 extension

```
<?php


/* The getExchangeRate function connects the database and call stored procedure to
retrieve the currency rate between 2 countries.
 */

function getExchangeRate ($country1,$country2) {

$db_name='ITSOBANK';
$usr_name = 'db2admin';
$password='db2admin';

   /*connect to database */

   $conn_res = db2_pconnect($db_name,$usr_name,$password);

   if ($conn_res) {

      $rate = 0.1;

   /*prepare the statement that calls the stored procedure */

      $sql = 'CALL ITSO.EXCHANGE_RATE (?,?,?)';
      $stmt = db2_prepare ($conn_res,$sql);

      if (!$stmt) {
         echo 'The prepare failed. <BR>';
         echo 'SQLSTATE value: ' . db2_stmt_error();
         echo 'with Message: ' . db2_stmt_errormsg();
      } else {

   /*bind stored procedure params */

         db2_bind_param($stmt, 1, "country1", DB2_PARAM_IN);
         db2_bind_param($stmt, 2, "country2", DB2_PARAM_IN);
```

```
        db2_bind_param($stmt, 3, "rate", DB2_PARAM_INOUT);
        }
  /*execute the statement */

  $result = db2_execute($stmt);

  if (!$result) {
     echo 'The execute failed. <BR>';
     echo 'SQLSTATE value: ' . db2_stmt_error();
     echo 'with Message: ' . db2_stmt_errormsg();
     }


  } else {
     echo 'Connection to database failed <BR>';
     echo 'SQLSTATE value: <BR>' . db2_conn_error();
     echo 'With Message: <BR>' . db2_conn_errormsg();
  }

return $rate
```

As you can see, we implemented the call to the stored procedure as a PHP function (*getExchangeRate)* since we want, later on, to publish this function as a Web service. The function return a SOAP object that contains the *rate* parameter which is the exchange rate value.

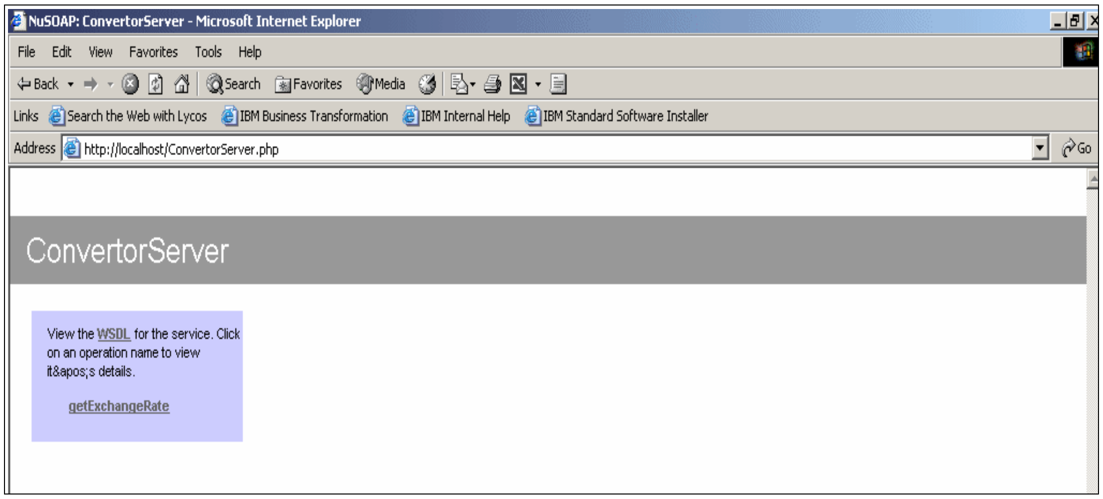
The next step will naturally be publishing our PHP function as a Web service, as shown in Example 16-35.

Example 16-35 Publishing getExchangeRate as a Web service

```
<?php
require("nusoap.php");

$ns = "http://localhost/nusoap";

$server = new soap_server();
$server->configureWSDL("ConvertorServer",$ns);
$server->wsdl->schemaTargetNamespace="http://localhost/nusoap/xsd";
$err = $server->getError();
echo $err;


$server->register ("getExchangeRate",
array ("country1" => "xsd:string" , "country2"=> "xsd:string" ),
array ("result" => "xsd:string" ),
$ns);
$HTTP_RAW_POST_DATA = isset($GLOBALS["HTTP_RAW_POST_DATA"])?
$GLOBALS["HTTP_RAW_POST_DATA"]
  : "";
$server->service($HTTP_RAW_POST_DATA);
?>
```

For implementing Web services, we used the NuSOAP extension. We created a new SOAP server and registered the *getExchangeRate* function to this Web service. There are two input parameters that are passed to the Web service: *country1* and *country2*. The *result* is an output parameter that will contain the exchange rate returned from the function.

The output of this program is shown in Figure 16-6.


NuSOAP: ConvertorServer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Search the Web with Lycos IBM Business Transformation IBM Internal Help IBM Standard Software Installer
Address http://localhost/ConvertorServer.php
Go

ConvertorServer

View the WSDL for the service. Click on an operation name to view it&apos;s details.

getExchangeRate

Figure 16-6 Currency convertor WSDL

And the WSDL file shown in Example 16-36.

Example 16-36 ITSO currency converter- WSDL structure

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <definitions 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"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://localhost/nusoap"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://localhost/nusoap">
- <types>
- <xsd:schema targetNamespace="http://localhost/nusoap">
<xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
</xsd:schema>
</types>
- <message name="getExchageRateRequest">
<part name="country1" type="xsd:String" />
<part name="country2" type="xsd:String" />
</message>
- <message name="getExchageRateResponse">
<part name="result" type="xsd:String" />
</message>
- <portType name="ITSOcurrencyConvertServicePortType">
- <operation name="getExchageRate">
<input message="tns:getExchageRateRequest" />
<output message="tns:getExchageRateResponse" />
</operation>
</portType>
```

```
- <binding name="ITSOcurrencyConvertServiceBinding"
type="tns:ITSOcurrencyConvertServicePortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="getExchageRate">
<soap:operation soapAction="http://localhost/ConvertorServer.php/getExchageRate"
style="rpc" />
- <input>
<soap:body use="encoded" namespace="http://localhost/nusoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
- <output>
<soap:body use="encoded" namespace="http://localhost/nusoap"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
</binding>
- <service name="ITSOcurrencyConvertService">
- <port name="ITSOcurrencyConvertServicePort"
binding="tns:ITSOcurrencyConvertServiceBinding">
<soap:address location="http://localhost/ConvertorServer.php" />
</port>
</service>
</definitions>
```

The next step consists of creating a Web consumer program that can be accessed from our portlet. Example 16-37 presents the PHP program.

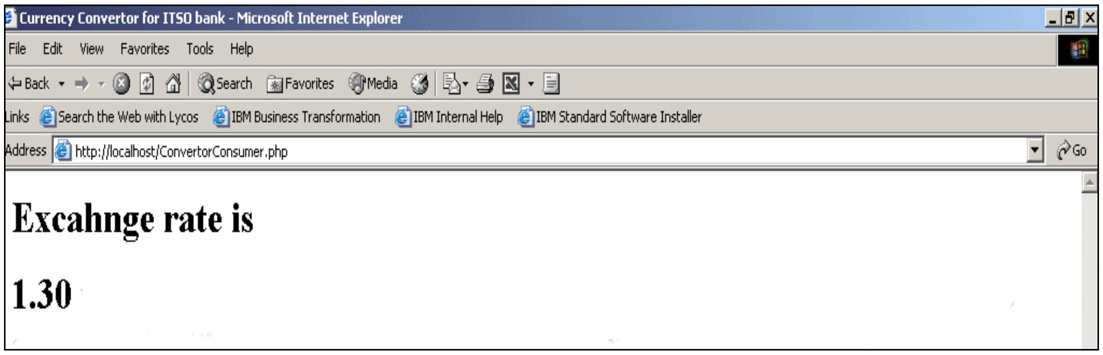
Example 16-37 ITSO currency converted Web service consumer program

```
<html>
<head>
<html>
<head>
<title> Currency Convertor for ITSO bank </title>
</head>
<body>
<h1> Excahnge rate is </h1>

<?php

require_once('nusoap.php');

/*$wsdl="http://localhost/ITSOcurrencyConvertService";*/
$wsdl="http://localhost/ConvertorServer.php?wsdl";
$client=new soapclient($wsdl, "wsdl");
$client->useHTTPPersistentConnection();

$param=array(
'country1'=>'USA',
'country2'=>'CAN',
'result'=>'',
);
$value =  $client->call('getExchangeRate', $param);
echo "<h1> $value </h1>";
```

```
?>

</html>
```

The Web service consumer program create (using NuSOAP extension) a SOAP client and access the *ITSOcurrencyConverService* with two input parameters: USA and CAN (Canada).The SOAP client call the *getExchangeRate* function and print the result in an HTML page.

The program output is shown in Figure 16-7.


Currency Convertor for ITSO bank - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Search the Web with Lycos IBM Business Transformation IBM Internal Help IBM Standard Software Installer
Address http://localhost/ConvertorConsumer.php
Go

Excahnge rate is

1.30

Figure 16-7 ITSO Currency converter Web service consumer result

16.6 Zend Core for IBM

On February 25, 2005 IBM and Zend Technologies announced a strategic partnership to collaborate on the development and support of the PHP environment. Under this collaboration, Zend made *Zend Core for IBM* available. Beyond this initial announcement, IBM and Zend are working together on furthering PHP technology directions. There are two goals:

- The first one is to bring the simplicity inherent in the 'do-it-yourself' infrastructure and PHP development to enterprise customers.
- The second goal is to add enterprise-class support to PHP in a non-intrusive way, without adding complexity for those who don't need the support, and without losing sight of what has made PHP so successful in the first place.

Zend Core for IBM is the only certified PHP development and production environment that includes tight integration with the Cloudscape and DB2 family of database servers, and native support for XML and Web Services. Certified by both Zend and IBM, Zend Core for IBM delivers a rapid development and production PHP foundation for applications using PHP with IBM databases and offers an easy upgrade path from DB2 Express or Cloudscape to the entire DB2 family without requiring any modifications to your PHP applications.

Developers and businesses already using PHP with an IBM database should use Zend Core for IBM. It is certified by both IBM and Zend and is fully supported by Zend. For those using PHP and currently evaluating databases, Zend Core for IBM includes all of PHP, plus a free DB2 Express database, and all the necessary extensions for out of the box development of PHP applications. Zend Core works with the entire DB2 family using a consistent API allowing easy migration between all the databases. Developers and businesses alike should seriously consider using DB2 Express with their PHP applications as it uniquely combines the power, functionality, and reliability of an open standards-based database server with simplicity in packaging, installation and deployment.

16.7 Conclusion

PHP is a powerful and most popular server side scripting language for Web servers. You can learn and implement PHP in a manner of few days, which make it a very cost affective scripting language.

PHP popularity is increasing every year since it is an open source language, it is easy to use, it is compatible, and it is has the full functionality for working with market leading database software.

PHP provide the full functionality for implementing Web services. It provides three major methods which can be easily used to publish existing business logic as a Web service with only a few lines of PHP code. Our scenario presented one program that calls a DB2 stored procedure which contains the business logic and provide this logic as a Web service. The second program presented was a Web service consumer program which can be implemented using portlet.

The investment in PHP application is very low and you can actually expose your organization business logic in short period of time and almost with no investment.

In this chapter we did not discuss about PHP security and PHP transactional support, but those issues are supported by PHP and for more information about that you should refer to the reference sources mentioned in the chapter.

In conclusion, we recommend that you consider PHP for implementing Web services as part of your SOA architecture.

17

WebSphere Application Server administration

This chapter describes the rules of a WebSphere Administrator involving SOA architecture and discusses what a WebSphere Application Server administrator does involving databases. This chapter should help the DBA understand the basics of WebSphere Application Server administration.

This chapter contains these topics:

- WebSphere foundation
- Responsibilities of a WebSphere Application Server administrator
- What does WebSphere Application Server Administrator do involving databases

© Copyright IBM Corp. 2006. All rights reserved.

17.1 WebSphere foundation

Figure 17-1 shows the job roles that are involved in the WebSphere Application Server environment.

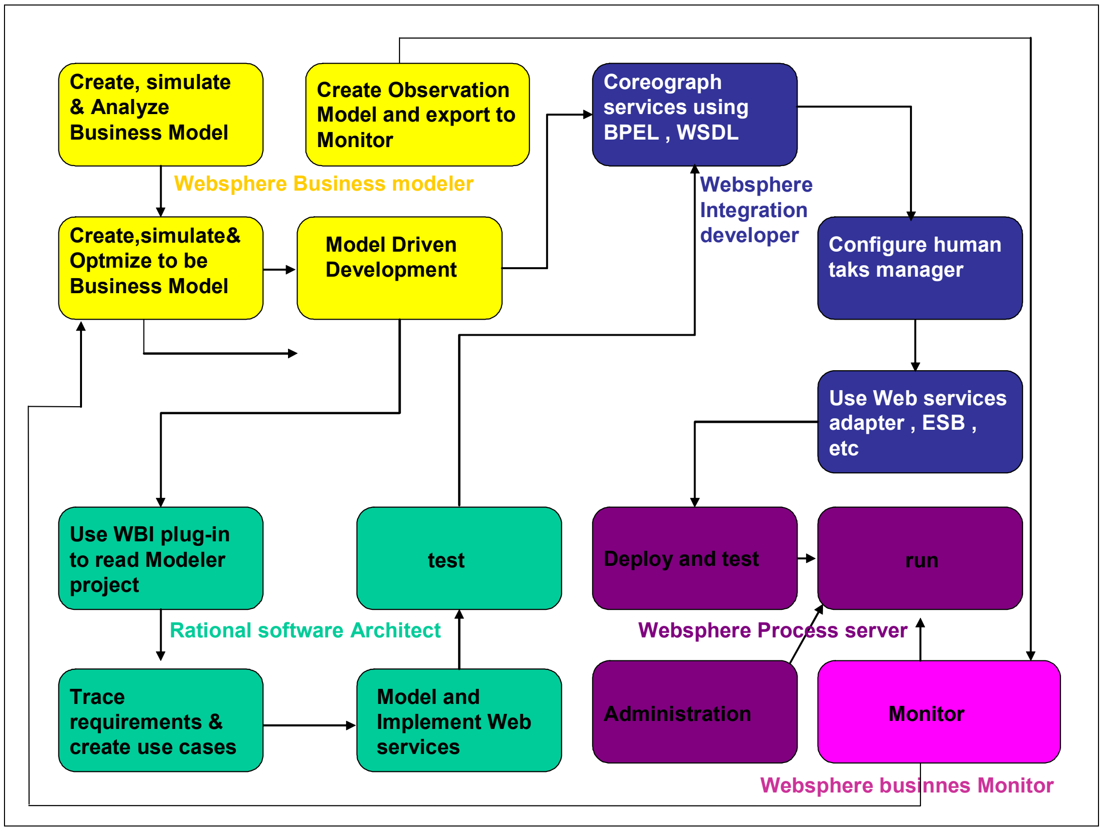

Create, simulate & Analyze Business Model
Create Observation Model and export to Monitor
Coreograph services using BPEL , WSDL
Websphere Business modeler
Websphere Integration developer
Create,simulate& Optmize to be Business Model
Model Driven Development
Configure human taks manager
Use Web services adapter , ESB , etc
Use WBI plug-in to read Modeler project
test
Deploy and test
run
Rational software Architect
Websphere Process server
Trace requirements & create use cases
Model and Implement Web services
Administration
Monitor
Websphere businnes Monitor

Figure 17-1 WebSphere job roles

In this big environment, we describe in the next section about the roles to an administrator of WebSphere Application Server (WAS).

17.2 Responsibilities of a WebSphere Application Server administrator

These are the basics roles to an Administrator of WebSphere Application Server:

- Describe architectural concepts related to WebSphere Application Server.
- Install and configure WebSphere Application Server (base and Network Deployment).
- Use the concepts of the Java 2 Platform, Enterprise Edition (J2EE).
- Assemble and install server-side Java enterprise applications.
- Use WebSphere Application Server administrative tools to configure and manage enterprise applications.
- Configure security for server-side application resources.
- Deploy applications in clustered environments.
- View performance information about server and application components.

- Use problem determination tools and log files to troubleshoot problems.
- Understand the role of the Administrative Console and the wsadmin administrative clients.
- Understand the managed processes that participate in the administrative domain.
- Understand the structure of the local node and master configuration repositories.
- Discuss the issues surrounding file synchronization between the local node and master configuration repositories.
- Describe JCA, JDBC Providers, Data sources and Persistence manager.
- Describe Web Services, Web Services Gateway and the UDDI Registry
- Define an EJB Module
- Define a Web Module
- Define an Application Client Module
- Define an Enterprise Application
- Use the Administrative Console to create a Data Source for installed applications to use.
- Locate and view some important log files
- Use the Log Analyzer to view the activity.log
- Enable tracing on an Application Server
- Define Workload Management.

17.3 What does WebSphere Application Server Administrator do involving databases

These are the main procedures that a WebSphere Application Server administrator does involving databases, we describe briefly about each one:

- Creating connection pooling.
- Creating JDBC providers.
- Creating Data source.
- Creating Data source security.
- Problem Determination.
- Handling Data source exceptions.
- Handling JDBC exceptions.
- Handling WebSphere exceptions.

17.3.1 Creating connection pooling

Each time an application attempts to access a back-end store (such as a database), it requires resources to create, maintain, and release a connection to that datastore. To mitigate the strain this process can place on overall application resources, WebSphere Application Server enables administrators to establish a pool of back-end connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

WebSphere Application Server supports JDBC 3.0 APIs for connection pooling and connection reuse. The connection pool is used to direct JDBC calls within the application, as well as for enterprise beans using the database.

Figure 17-2 on page 540 shows a example of a connection pooling.

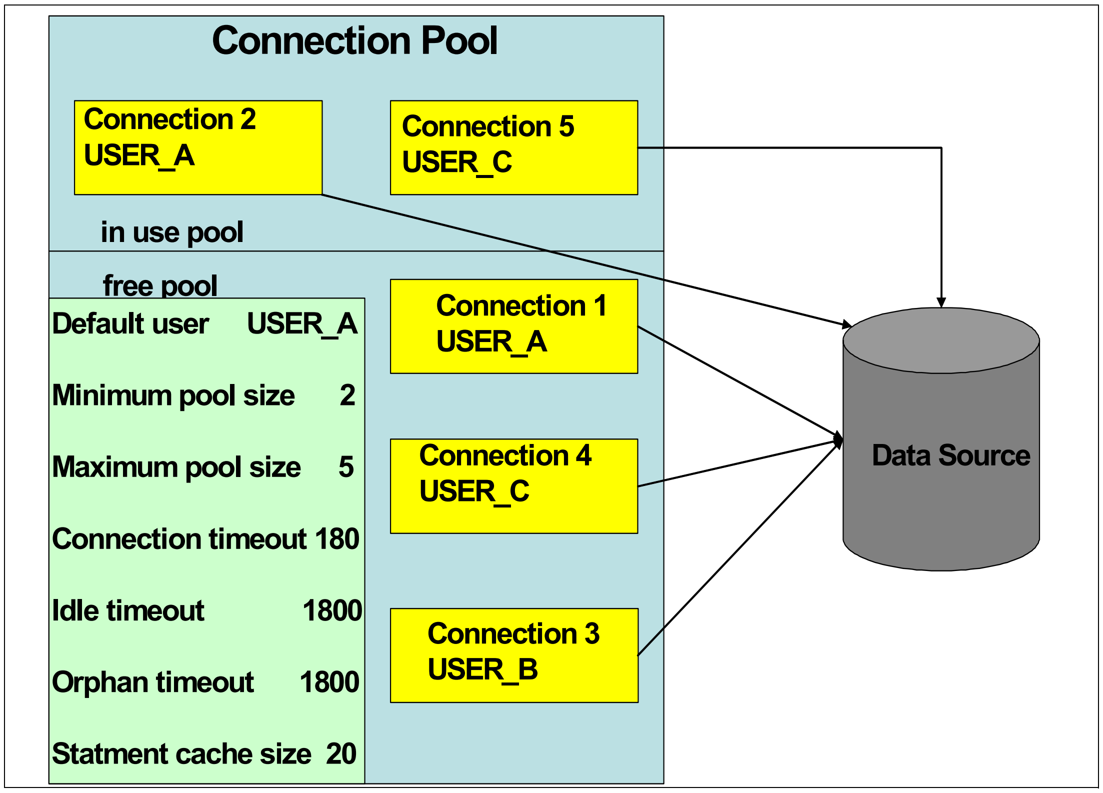


Figure 17-2 Connection pooling example

Note: For a basic understanding of the JDBC 3.0 Core API and the JDBC 3.0 Optional Package API, refer to:

`http://java.sun.com/products/jdbc/download.html`

Benefits of connection pooling

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a data source. Because users connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total datastore overhead quickly becomes high for Web-based applications, and performance deteriorates. When connection pooling capabilities are used, however, Web applications can realize performance improvements of up to 20 times the normal results.

With connection pooling, most user requests do not incur the overhead of creating a new connection because the data source can locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. The overhead of a disconnection is avoided. Each user request incurs a fraction of the cost for connecting or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused.

When to use connection pooling

Use WebSphere connection pooling in an application that meets any of the following criteria:

- It cannot tolerate the overhead of obtaining and releasing connections whenever a connection is used.
- It requires Java Transaction API (JTA) transactions within WebSphere Application Server.
- It needs to share connections among multiple users within the same transaction.
- It needs to take advantage of product features for managing local transactions within the application server.
- It does not manage the pooling of its own connections.
- It does not manage the specifics of creating a connection, such as the database name, user name or password.

17.3.2 Best practices

How connections are pooled together

Whenever you configure a unique data source or connection factory, you are required to give it a unique Java Naming and Directory Interface (JNDI) name. This JNDI name, along with its configuration information, is used to create the connection pool. A separate connection pool exists for each configured data source or connection factory.

A separate instance of a given configured connection pool is created on each application server that uses that data source or connection factory. For example, if you run a three server cluster in which all of the servers use myDataSource, and myDataSource has a maximum connections setting of 10, then you can generate up to 30 connections (three servers times 10 connections). Be sure to consider this fact when determining how many connections to your back-end resource you can support.

It is also important to note that when using connection sharing, it is only possible to share connections obtained from the same connection pool.

Avoiding a deadlock

Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:

- Each thread has its first database connection and all are in use.
- Each thread is waiting for a second database connection and none would become available since all threads are blocked.

To prevent the deadlock in this case, the maximum connections value for the database connection pool should be increased by at least one. Doing this allows for at least one of the waiting threads to obtain its second database connection and to avoid a deadlock.

To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require C concurrent database connections per thread, the connection pool must support at least the following number of connections, where T is the maximum number of threads.

```
T * (C - 1) + 1
```

The connection pool settings are directly related to the number of connections that the database server is configured to support.If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed.

17.3.3 Data Sources

Installed applications use a data source to obtain connections to a relational database. A data source is analogous to the J2EE Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

A data source is associated with a JDBC provider, which supplies the driver implementation classes that are required for JDBC connectivity with your specific vendor database. Application components transact directly with the data source to obtain connection instances to your database. The connection pool that corresponds to each data source provides connection management.

You can create multiple data sources with different settings, and associate them with the same JDBC provider. For example, you might use multiple data sources to access different databases within the same vendor database application. WebSphere Application Server requires JDBC providers to implement one or both of the following data source interfaces, which are defined by Sun Microsystems. These interfaces enable the application to run in a single-phase or two-phase transaction protocol.

ConnectionPoolDataSource - a data source that supports application participation in local and global transactions, excepting two-phase commit transactions.

Note: In one case a connection pool data source does support two-phase commit transactions: when the JDBC provider is DB2 for z/OS Local JDBC provider (RRS).

When a connection pool data source is involved in a global transaction, transaction recovery is not provided by the transaction manager. The application is responsible for providing the backup recovery process if multiple resource managers are involved.

XADataSource - a data source that supports application participation in any single-phase or two-phase transaction environment. When this data source is involved in a global transaction, the WebSphere Application Server transaction manager provides transaction recovery.

In WebSphere Application Server releases prior to version 5.0, the function of data access was provided by a single connection manager (CM) architecture. This connection manager architecture remains available to support J2EE 1.2 applications, but another connection manager architecture is provided, based on the JCA architecture supporting the new J2EE 1.3 application style (also for J2EE 1.4 applications).

These two separate architectures are represented by two types of data sources. To choose the right data source, administrators must understand the nature of their applications, EJB modules, and enterprise beans.

Data source - This data source uses the JCA standard architecture to provide support for J2EE version 1.3 and 1.4 applications. It runs under the JCA connection manager and the relational resource adapter.

17.3.4 JDBC providers

Installed applications use JDBC providers to interact with relational databases.The JDBC provider object supplies the specific JDBC driver implementation class for access to a specific vendor database.

To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the J2EE Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

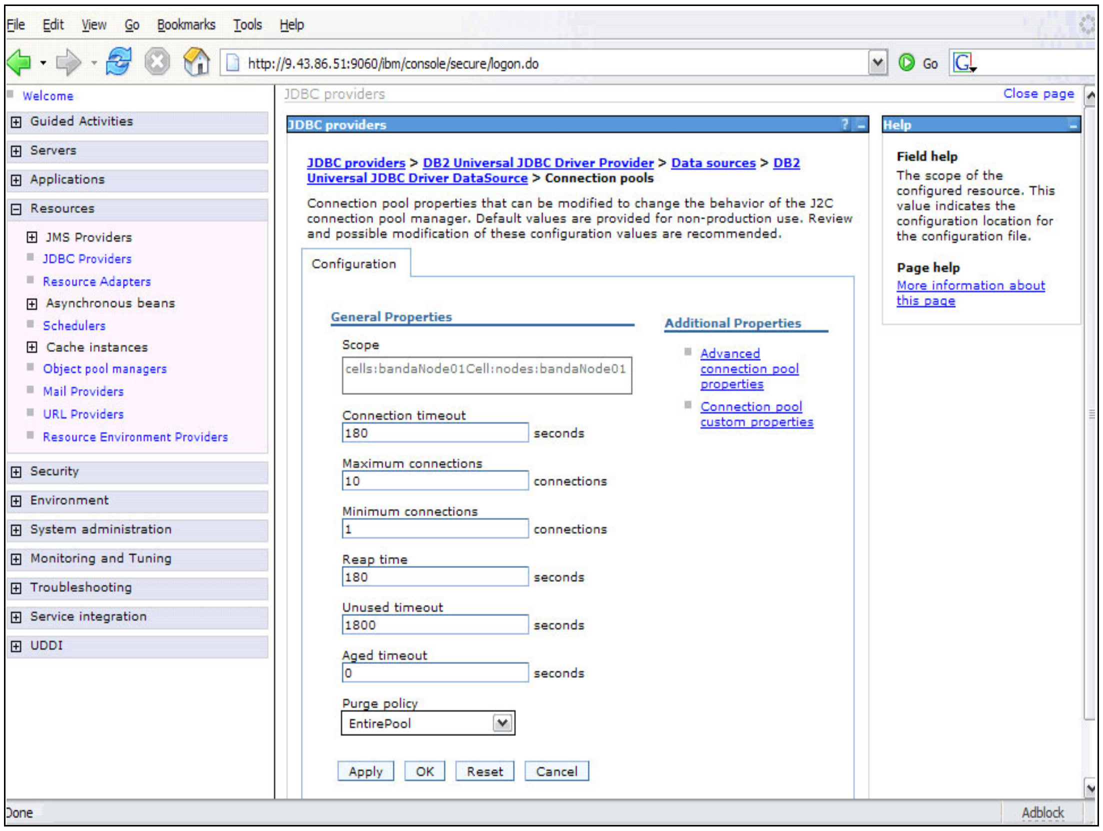
The WebSphere Application Server prerequisite Web site has a current list of supported providers.If your database is DB2, you can proceed directly to vendor-specific data sources minimum required settings to learn which DB2 JDBC provider is appropriate for your database configuration and application requirements.

This document contains descriptions of the following providers, including the supported data source classes and their required properties.

Note: For more information about JDBC drivers, see 6.7, “Connecting your services to DB2 for z/OS through JCC (JDBC)” on page 152.

17.3.5 Create a data source

Figure 17-3 shows how the WebSphere Application Server admin set up a data source using the admin console via Web browser.



JDBC providers > DB2 Universal JDBC Driver Provider > Data sources > DB2 Universal JDBC Driver DataSource > **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. Review and possible modification of these configuration values are recommended.

Scope: cells:bandaNode01Cell:nodes:bandaNode01

Connection timeout: 180 seconds

Maximum connections: 10 connections

Minimum connections: 1 connections

Reap time: 180 seconds

Unused timeout: 1800 seconds

Aged timeout: 0 seconds

Purge policy: EntirePool

Field help
The scope of the configured resource. This value indicates the configuration location for the configuration file.

Page help
More information about this page

Figure 17-3 Configuring data sources

A Datasource has many properties. The following properties are vendor-neutral:

Name: Specifies the display name for the data source like *TestDataSource.*

JNDI name: Specifies the Java Naming and Directory Interface (JNDI) name. For example, you can use the name jdbc/TestDataSource. If you leave this field blank a JNDI name is generated from the name of the data source like *jdbc/TestDataSource*.

Description: Assigns a text description for the resource.

Category: Specifies a category string you can use to classify or group the resource.

Statement cache size: Specifies the number of free prepared statements that are cached per connection.

Datasource helper classname: Specifies the datastore helper that is used to perform database-specific functions. This helper is used by the Relational Resource Adapter at runtime. The default DataStoreHelper implementation class is set based on the JDBC driver implementation class, using the structure:

com.ibm.websphere.rsadapter.<database>DataStoreHelper

See Example 17-1 that shows the class of DataStoreHelper with DB2 JDBC.

Example 17-1 DataStoreHelper class

```
com.ibm.websphere.rsadapter.DB2DataStoreHelper
```

Note: You can change to your subclass of this DataStoreHelper if necessary.

Connection timeout: Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

Maximum connections: Specifies the maximum number of physical connections that can be in the pool.

Minimum connections: Specifies the minimum number of physical connections to maintain in the pool.

Reap time: Specifies the interval, in seconds, between runs of the pool maintenance thread

Unused timeout: Specifies the interval in seconds after which an unused or idle connection is discarded.

Aged timeout Specifies the interval in seconds before a physical connection is discarded.

Purge policy: Specifies how to purge connections when a "stale connection" or "fatal connection error" is detected. Valid values are EntirePool and FailingConnectionOnly.

17.3.6 Security

WebSphere provides robust security management including authentication, authorization, and auditing infrastructure for WebSphere-based applications. These facilities interact with the database Security to augment the security infrastructure provided directly within WebSphere.

Setting the JDBC driver securityMechanism

To set the securityMechanism value within WebSphere, a property must be added to the custom properties of the datasource.

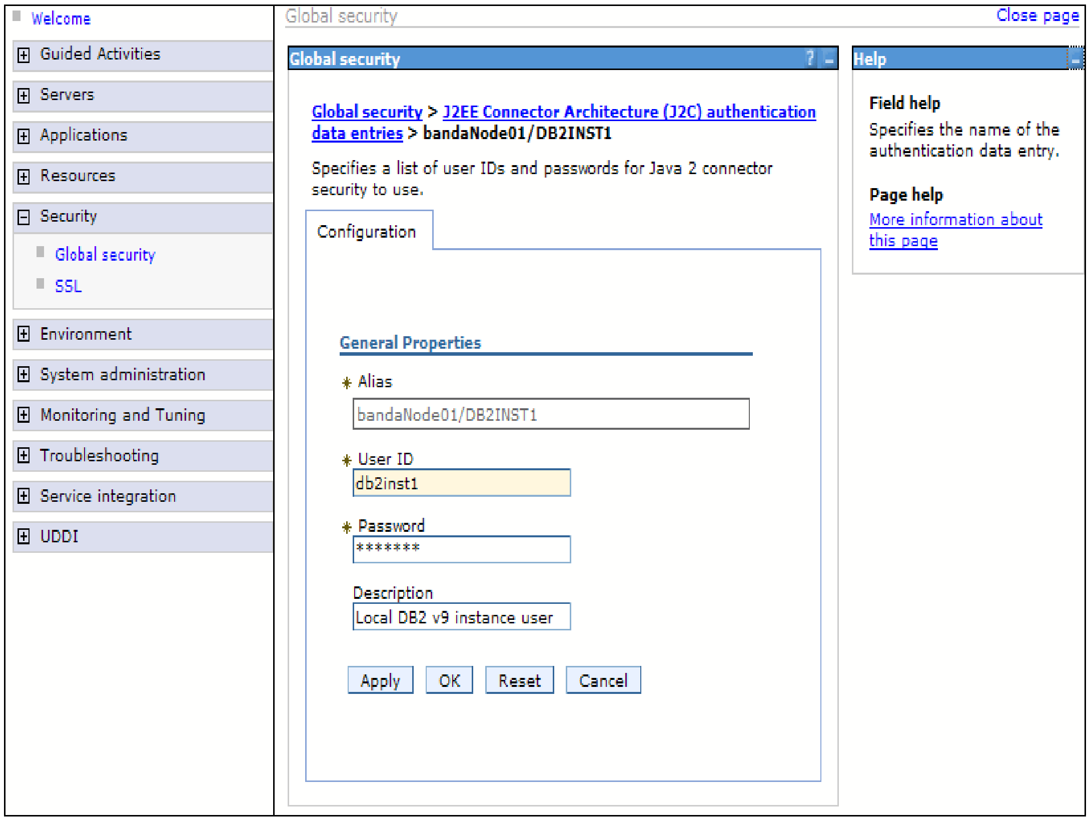
WebSphere datasource settings for authentication

Within the WebSphere datasource definition, there are two potential fields for specifying the identity to be utilized. Since the datasource can be referenced by an application wanting to have the container determine the identity, or allow the program to specify the user ID and password for connectivity, the datasource definition supports the setting of an authorization.

The potential authentication values themselves are defined within the Security -JAAS Configuration - JCA Authentication option of the administration console application.

The user ID and password must be a valid SAF user ID, if they are to be used for connectivity to DB2 and referenced in the datasource definition. Once the JCA authorization IDs are defined, the datasource can specify an ID to be used if the datasource is accessed with res-auth=container, and a separate ID if res-auth=application is used.

Figure 17-4 shows how to setup authentication security in WebSphere.



Global security > J2EE Connector Architecture (J2C) authentication data entries > bandaNode01/DB2INST1

Specifies a list of user IDs and passwords for Java 2 connector security to use.

* Alias: bandaNode01/DB2INST1
* User ID: db2inst1
* Password: *******

Description: Local DB2 v9 instance user

Field help
Specifies the name of the authentication data entry.

Page help
More information about this page

Figure 17-4 Configuring JAAS

Adding JAAS authentication to a WebSphere Application Server

You can use the Security tab to define a Java Authentication and Authorization Extension (JAAS) alias for a username and password to connect to the data source.

The Java Authentication and Authorization Service (JAAS) Version 1.0 extends the Java 2 Security Architecture of the Java 2 Platform with additional support for authentication and for enforcing access control upon users. The development environment supports the JAAS architecture and extends the access control architecture to support role-based authorization for J2EE resources including servlet, JSP, and EJB components. JAAS maps an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for a specified back-end Enterprise Information System (EIS).

Prerequisite: Create an enterprise application and target the server to WebSphere Application Server V6.0.

To add JAAS authentication, follow these steps:

1. Switch to the J2EE perspective.
2. In the Project Explorer view, expand the Enterprise Applications folder.
3. Under the enterprise application project folder for which you want to add JAAS authentication, double-click the Deployment Descriptor to open the Application Deployment Descriptor editor.
4. Select the Deployment tab at the bottom of the editor.
5. Expand the Authentication section.
6. Click the Add button beside the JAAS authentication entries list table. The Add JAAS Authentication Entry dialog box opens.
7. In the dialog box, fill in an alias, user id, password, and description for the authentication entry. For example, you could enter the alias, DB2 user ID, and DB2 password to access a DB2 database. Click OK.
8. Save your changes and close the editor. A JAAS authentication alias has been added to the deployment descriptor files.

17.3.7 Problem determination

Problem determination in DBMS and WebSphere has become a difficult process for some problems with the use of multi-threading in Java, as well as, but not limited to, the ideas of connection pooling and PreparedStatement cache.

When contacting either support team, you will be asked to produce some basic information. Here we will cover what that information is, as well as what it means to a problem. After initial analysis of the basic information, each support team may ask for some specific traces. The traces are used to look inside each product as a problem happens. Through the use of the traces, the support and development teams can determine the cause of most problems.

Basic information for problem determination

With the integration of many different products, the problem determination process is becoming quite difficult in some cases.The idea of problem determination in a DBMS and WebSphere environment is to determine where the actual problem lies, and then determine what it is. The first step of determining where the problem lies can be as difficult if not more so than determining what the problem actually is.

Determining the location

As with most products, a lot of problems are due to how a product is used. Customer error is the most common reason for calls to the support teams. Of course this should be expected with so many different types of each product on the market, what are the chances of them all working the same? The biggest concern is how long it takes to determine if it is a customer error, or a problem with the product.

The location of a problem usually means which piece of software or hardware is the real cause. In most environments the problem that the user sees is not always the actual problem, some of the time it is only a consequence of the real problem.

Tracing

Tracing can also play a big part in determining where the root of a problem is. In the DBMS and WebSphere environment there is a lot of traces available for this purpose.Traces like a JDBC, CLI, or WebSphere tracing may not present much of a performance hit, but may produce the information needed.

When you get to the JDBC trace section of this chapter you will see that the trace provides a trace point for each method call. From this information we can determine if the right methods were called in the right order before a problem occurred. This is a great method for determining if there was a customer error in JDBC coding. From this trace it is also possible to determine if the error came from deeper within DB2 or if it was something that the JDBC driver did not like.

A WebSphere trace can work the same way. When the WebSphere trace is produced it can list each call made to the database. This information can be important in determining if the right call was made or not.

17.3.8 Database connection problems

WebSphere Application Server diagnostic tools provide services to help troubleshoot database connection problems. Additionally, the IBM Web site provides flexible searching capabilities for finding documented solutions to database-specific connection problems.

The following steps help you quickly isolate connectivity problems.

1. Browse the log files of the application server for clues.
2. Browse the Helper Class property of the data source to verify that it is correct and that it is on the WebSphere Application Server class path. Mysterious errors or behavior might result from a missing or misnamed Helper Class name. If WebSphere Application Server cannot load the specified class, it uses a default helper class that might not function correctly with your database manager.
3. Verify that the Java Naming and Directory Interface (JNDI) name of the data source matches the name used by the client attempting to access it. If error messages indicate that the problem might be naming-related, such as referring to the name server or naming service, or including error IDs beginning with NMSV
4. Enable tracing for the resource adapter using the trace specification, RRA=all=enabled. Follow the instructions for dumping and browsing the trace output, to narrow the origin of the problem.

Note: For a comprehensive list of database-specific troubleshooting tips, see the WebSphere support page at:

`http://www-306.ibm.com/software/webservers/appserv/was/support/`

General data access problems

There are several errors, the list shows a few of these errors, we give a example showing the details about the first one:

- An exception "IllegalConnectionUseException" occurs.
- WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred.
- ConnectionWaitTimeoutException.

- com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N. The database alias name or database name "NULL" could not be found. SQLSTATE=42705.
- java.sql.SQLException: java.lang.UnsatisfiedLinkError.

An exception "IllegalConnectionUseException" occurs

This error can occur because a connection obtained from a WAS40DataSource is being used on more than one thread. This usage violates the J2EE 1.3 programming model, and an exception generates when it is detected on the server. This problem occurs for users accessing a data source through servlets or bean-managed persistence (BMP) enterprise beans.

DBMS errors

WebSphere receives the errors from the DBMS, like IMS, INFORMIX, Cloudscape, SQL Servers, DB2 UDB, and so on.

See Example 17-2 for typical DB2 for z/OS errors.

Example 17-2 DB2 for z/OS typical errors

```
# SQL0567N "DB2ADMIN " is not a valid authorization ID. SQLSTATE=42602
# SQL0805N Package package-name was not found
# SQL0805N Package "NULLID.SQLLC300" was not found. SQLSTATE=51002
# SQL30082N Attempt to establish connection failed with security reason "17"
("UNSUPPORTED FUNCTION') SQLSTATE=08001
```

17.3.9 JDBC trace configuration

If your application displays JDBC-related exception messages, activate the JDBC trace service. The resulting log text can help you identify the problem.

Turn on tracing for most database JDBC implementations through the administrative console; see the article Tracing and logging configuration for instructions. This method activates JDBC trace for all applications that run in the server you specify. Identify your database type by selecting the trace group WAS.database and typing one of the following trace strings in the console:

`com.ibm.ws.database.logwriter`

Trace string for databases that use the GenericDataStoreHelper. Table 17-1 shows the trace string that you can also use for unsupported databases.

Table 17-1 JDBC traces

Trace string	Database
com.ibm.ws.db2.logwriter	DB2
com.ibm.ws.oracle.logwriter	Oracle
com.ibm.ws.derby.logwriter	Derby
com.ibm.ws.informix.logwriter	Informix
com.ibm.ws.sqlserver.logwriter	Microsoft SQL Server
com.ibm.ws.sybase.logwriter	Sybase

A few JDBC drivers require that you set trace differently, at the data source level. These drivers include:

- DB2 former or earlier CLI-based JDBC driver
- WebSphere embedded ConnectJDBC driver for MS SQL Server
- DataDirect ConnectJDBC driver for MS SQL Server
- DataDirect SequeLink JDBC driver for MS SQL Server, which is deprecated in WebSphere Application Server V6.0
- Microsoft JDBC driver for MS SQL Server 2000, which is deprecated in WebSphere Application Server V6.0

Configuring trace for these drivers through the WAS.database group results in corrupt trace information. WebSphere Application Server sets trace for the group at the server level, causing the trace service to begin only after your application establishes an initial connection. Because that first connection does not carry trace information, re-use of it is never tracked. Consequently the application cannot accurately match trace information to connection use.

Set trace for the previously mentioned JDBC drivers through data source custom properties. For example, use custom property spyAttributes to enable the JDBC trace for SequeLink or Connect JDBC drivers. Consult your driver documentation for details on the custom property that enables trace for your JDBC implementation.

Tip: If the JDBC tracing service cannot help you isolate and fix your problem, consult the IBM Support Web site for WebSphere Application Server at:

`http://www.ibm.com/support/search.wss?rs=180&tc=SSEQTP&tc1=SSCMP9Y`

Use the site search function to find current information about known problems and their resolutions. Locating the right troubleshooting tip can save time that, otherwise, you might spend on opening and tracking a PMR.

17.3.10 WebSphere exceptions

WebSphere connection pooling monitors specific SQLExceptions thrown by the database. A set of these exceptions are mapped to WebSphere Application Server specific exceptions. WebSphere Application Server connection pooling provides these exceptions to ease development by not requiring the developer to know all of the database-specific SQLExceptions that could be thrown for very common occurrences.

Two commonly encountered exceptions are:

- ConnectionWaitTimeoutException
- StaleConnectionException

ConnectionWaitTimeoutException

ConnectionWaitTimeoutException accessing a data source or resource adapter.If your application receives these exception when attempting to access a WebSphere Application Server data source or JCA-compliant resource adapter, respectively, some possible causes are:

- The maximum number of connections for a given pool is set too low. The demand for concurrent use of connections is greater then the configured maximum value for the connection pool.

- If your connection wait timeout value is too low, you might timeout shortly before a user returns a connection back to the pool. Adjusting the connection wait time can give you some relief. One indication of this problem is that you use close to the maximum number of connections for an extended period and receiving this error regularly.
- You are not closing some connections or you are returning connections back to the pool at a very slow rate.

To correct these problems, either:

- Modify an application to use fewer connections.
- Properly close the connections.
- Change the pool settings of MaxConnections or ConnnectionWaitTimeout.
- Adjust resources and their configurations.

StaleConnectionException

This exception (com.ibm.websphere.ce.cm.StaleConnectionException) indicates that the connection currently being held is no longer valid. This can occur for a number of reasons, including these:

1. The application tries to get a connection and fails, as when the database is not started.

2. A connection is no longer usable due to a database failure. When an application tries to use a connection it has previously obtained, the connection is no longer valid. In this case, all connections currently in use by an application could get this error when they try to use the connection.

3. The application tries to use a JDBC resource, such as a statement, obtained on a now-stale connection.

4. The application using the connection has already called close() and then tries to use the connection again.

5. The connection has been orphaned because the application had not used it within a time interval of twice the orphan timeout value, and then the application attempted to use the orphaned connection.

Applications are not required to explicitly catch a StaleConnectionException. A StaleConnectionException is a subclass of java.sql.SQLException, which applications are already required to catch. However, catching a StaleConnectionException makes it possible for applications to recover from bad connections in many instances.

The most common time for StaleConnectionException to be thrown is the first time that a connection is used, just after it is retrieved. Because connections are pooled, a database failure is not detected until the operation immediately following its retrieval from the pool, which is the first time communication to the database is attempted. And it is only when a failure is detected that the connection is marked stale. StaleConnectionException occurs less often if each method that accesses the database gets a new connection from the pool.

Examining the sequence of events that occur when a database fails to service a JDBC request shows that this occurs because all connections currently handed out to an application are marked stale; the more connections the application has, the more connections can be stale.

Generally when StaleConnectionException is caught, the transaction in which the connection was involved needs to be rolled back, and a new transaction begun with a new connection.

Note: Details can be found in the “WebSphere Connection Pooling” document by Deb Ericson, Shawn Lauzon, and MelissaModjeski, which is located at:

`http://www.ibm.com/software/webservers/appserv/whitepapers/connection_pool.pdf`

See also the Redbooks:

- *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- *DB2 UDB/WebSphere Performance Tuning Guide*, SG24-6417
- *Using Informix Dynamic Server with WebSphere*, SG24-6948

And see the Redpaper:

- *WebSphere for z/OS to CICS and IMS Connectivity Performance*, REDP-3959

18

Managing and monitoring SOA applications

This chapter describes the tools for monitoring SOA applications. We discuss offerings for composite application management in the context of IBM Tivoli Composite Application Manager Version 6 Family. We describe IBM Tivoli Composite Application Manager for SOA (ITCAM for SOA) and provide a case study implementation. For detailed information about IBM Tivoli products, refer to the standard IBM Tivoli publications.

The discussion includes:

- IBM Tivoli Composite Application Manager V6 Family
- ITCAM for product features
- ITCAM for SOA product components
- Monitoring performance in DB2
- Stand alone monitoring tools for SOA

© Copyright IBM Corp. 2006. All rights reserved.

18.1 IBM Tivoli Composite Application Manager V6 Family

In this section we discuss the IBM Tivoli products for service management.

18.1.1 Why manage?

SOA applications, as mentioned in Chapter 4, “SOA and user interfaces with portals” on page 69, tend to have multiple layers, often distributed across different servers, different platforms, and different components. SOA application monitoring, management, operational settings, problem determination, and performance management post a challenge for management tools.

SOA applications as business-critical entity must be available with adequate response time for users to perform their tasks. With application components spread throughout the enterprise, problem determination and performance management are typically a nightmare. There is no clear path for finding which component has the problem; sometimes these components even belong to different organizations. Is it the database? Or maybe a network problem? Or the application server experiencing a bottleneck? Maybe an end-user machine is stall?

18.1.2 IBM Tivoli system management portfolio

Tivoli product solutions are aligned toward an overall IBM IT Service Management approach. Figure 18-1 shows the IBM IT Service Management portfolio structure.

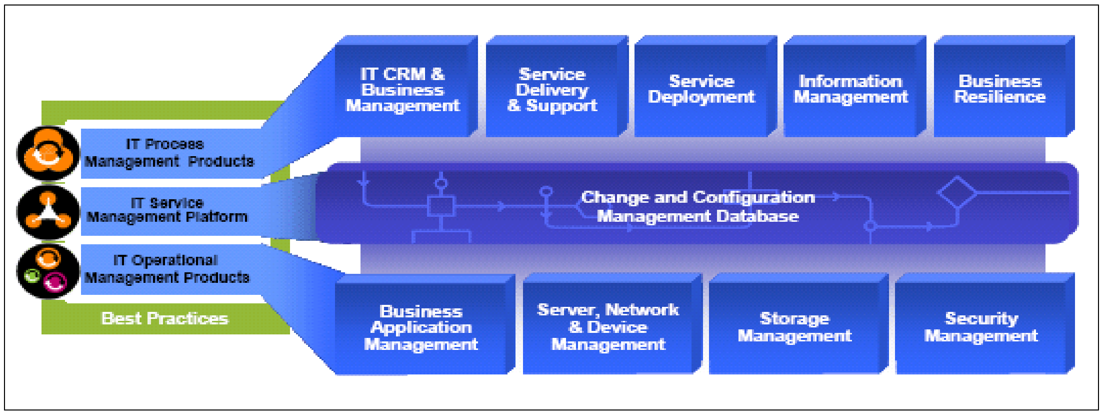

IT CRM & Business Management
Service Delivery & Support
Service Deployment
Information Management
Business Resilience
IT Process Management Products
IT Service Management Platform
IT Operational Management Products
Best Practices
Change and Configuration Management Database
Business Application Management
Server, Network & Device Management
Storage Management
Security Management

Figure 18-1 IBM IT service management

This approach provides IT infrastructure library (ITIL®) aligned automation workflows. Future offerings will provide an open- standard based, configuration management database (CMDB) based solution as well as a workflow engine.

The operational management pillar, as shown in Figure 18-1, is divided into software families. The availability solution addressed in business applications management and server, network, and device management can be viewed as an integrated offering as show in Figure 18-2 on page 555.

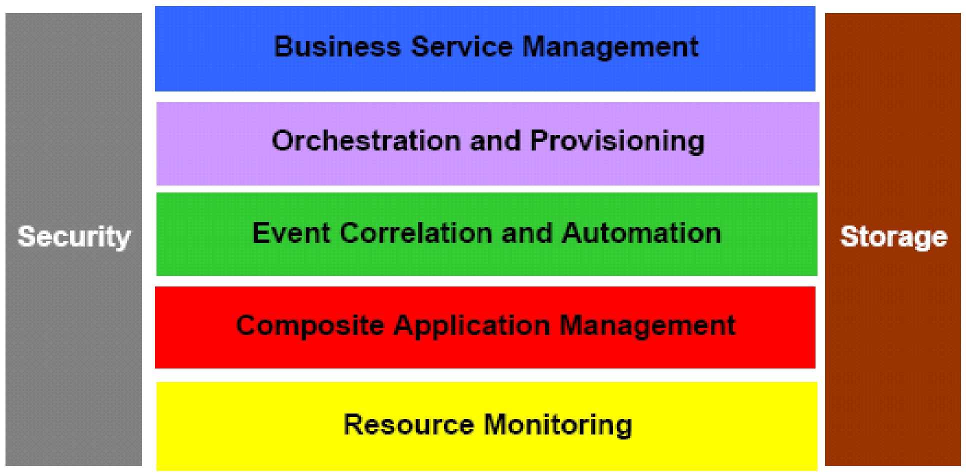

Business Service Management
Orchestration and Provisioning
Security
Event Correlation and Automation
Storage
Composite Application Management
Resource Monitoring

Figure 18-2 Tivoli software portfolio

Tivoli availability portfolio is divided into:

- Resource monitoring

  Measuring and managing IT resource performance, including servers, databases, and middleware.
- Composite application management

  Monitoring and managing an application and its components, understanding applications from the availability standpoint.
- Event Correlation and automation

  Correlates and automates events or faults that are generated by resource monitoring, application monitoring, or both to provide a concise root-cause analysis of failure in the environment.
- Orchestration and provisioning

  Provides the ability to deploy or re-deploy servers or components as requested, on demand, to fulfill processing needs, if the need arises as indicated by the correlation engine.
- Business Services Management

  Provides a high-level view of business status as reflected by its underlying monitoring components; the view can be either be in real time or based on a Service level Agreement.

18.1.3 Tivoli composite application solution

The IBM Tivoli Composite Application Manager family resides in the application manager pillar from the Tivoli software portfolio. The current application management portfolio consists of the following products:

- ITCAM for Response Time Tracking V6.0
- ITCAM for SOA V6.0
- ITCAM for WebSphere V6.0
- ITCAM for CICS Transactions V6.0
- ITCAM for IMS Transaction V6.0
- OMEGAMON® XE for WebSphere Business Integration V1.1

Figure 18-3 on page 556 shows the application management scope.

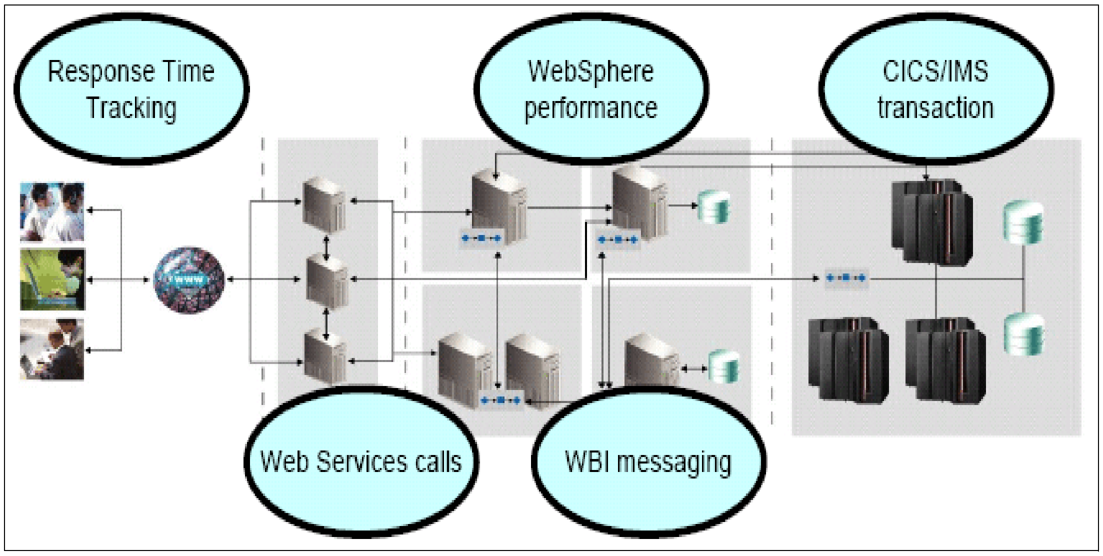


Figure 18-3 Application management

In this redbook we focus on ITCAM for SOA (IBM Tivoli Composite Application Manager V6.0 for SOA). For more information about the ITCAM solutions mentioned on the software portfolio, refer to *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151, or the ITCAM product documentation.

18.2 ITCAM for product features

ITCAMS for SOA manages service-oriented architecture (SOA). It can monitor, manage, and control Web service layer of IT architecture while drilling down to the application or resource layer to identify the source of bottlenecks or failures and to pinpoint services that are most time consuming or use the most resources. ITCAM for SOA:

- Provides service monitoring views in Tivoli Enterprise Portal.

  ITCAM for SOA workspace consists of:

  – Performance summary

    Shows the response time information for Web services calls as viewed from the client or the server.

  – Message summary

    Shows the message statistics, including the volume and size of the message information.

  – Fault summary

    Shows failure analysis for Web services calls.

  – Service Management Configuration summary

    Provides a summary of the Web services configuration.

- Leverages Tivoli Enterprise Portal situation to check threshold. ITCAM for SOA provides some predefined situations that you need to tailor. The Predefined situations concern:

  – Number of messages received by the service within a time window.

  – Size of the message.

- Offers heterogeneous platform coverage:
  - Support for IBM WebSphere Application Server, Microsoft .NET, and BEA WebLogic.
  - Target IBM Enterprise Service Bus platforms: WebSphere Application Server 5.x, and 6.x, WebSphere Integration Server Foundation 5.1.1.
- Displays a list of services and operations monitored in environment.
- Leverages Tivoli Enterprise Portal workflow and policy editor for threshold-triggered action sequences.
- Offers the ability to include Service-Layer views in Tivoli Enterprise Portal.

Figure 18-4 shows a sample of ITCAM for SOA workspace.

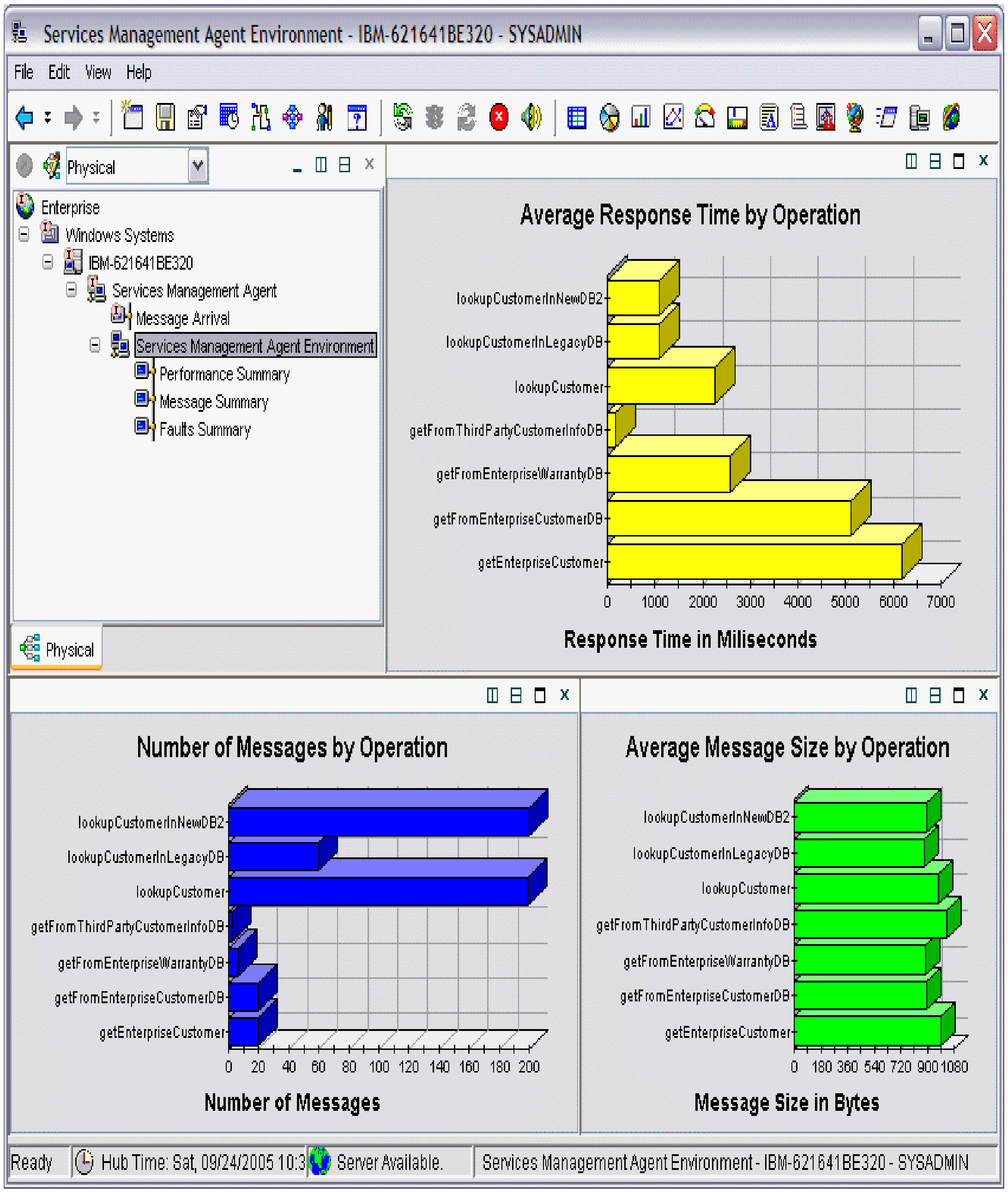

Services Management Agent Environment - IBM-621641BE320 - SYSADMIN
File Edit View Help
Physical
Enterprise
Windows Systems
IBM-621641BE320
Services Management Agent
Message Arrival
Services Management Agent Environment
Performance Summary
Message Summary
Faults Summary
Physical
Average Response Time by Operation
lookupCustomerInNewDB2
lookupCustomerInLegacyDB
lookupCustomer
getFromThirdPartyCustomerInfoDB
getFromEnterpriseWarrantyDB
getFromEnterpriseCustomerDB
getEnterpriseCustomer
0 1000 2000 3000 4000 5000 6000 7000
Response Time in Miliseconds
Number of Messages by Operation
lookupCustomerInNewDB2
lookupCustomerInLegacyDB
lookupCustomer
getFromThirdPartyCustomerInfoDB
getFromEnterpriseWarrantyDB
getFromEnterpriseCustomerDB
getEnterpriseCustomer
0 20 40 60 80 100 120 140 160 180 200
Number of Messages
Average Message Size by Operation
lookupCustomerInNewDB2
lookupCustomerInLegacyDB
lookupCustomer
getFromThirdPartyCustomerInfoDB
getFromEnterpriseWarrantyDB
getFromEnterpriseCustomerDB
getEnterpriseCustomer
0 180 360 540 720 900 1080
Message Size in Bytes
Ready
Hub Time: Sat, 09/24/2005 10:3
Server Available.
Services Management Agent Environment - IBM-621641BE320 - SYSADMIN

Figure 18-4 ITCAM for SOA workspace

The context-rich views and inter-workspace linkage in the Tivoli Enterprise Portal enables users to drill down to IT resources to identify of Web service bottlenecks and failures. By providing built-in and extensive alerts, situations, and workflows, users can create powerful automated mediation scenarios via the Tivoli Enterprise Portal.

18.3 ITCAM for SOA product components

ITCAM for SOA manages Web services. Web services can be viewed as a remote processing facility that is defined through the use of Web Services Definition Language (WSDL). Usual access uses simple Object Access Protocol (SOAP) over HTTP. Internally, Web services are implemented using Java API for XML-based Remote Procedure Call (JAX-RPC). ITCAM for SOA installs itself as the JAX-RPC handler to capture and manage Web services Calls.

ITCAM for SOA consists of these logical components:

- Web services data collector, which acts as the JAX-RPC handler and intercepts Web services calls to collect statistical information and write to a log file.
- Tivoli Enterprise Monitoring Agent collects information from all of the data collectors on a machine and forwards them to Tivoli Enterprise Monitoring Server. The data collectors and Tivoli Enterprise Monitoring Agent are discussed in 18.3.1, “Monitoring agent data collector” on page 558.
- An Eclipsed-based viewer that processes log files that are generated by the Web services data collector. it generates visual representations of various characteristics of monitored Web services. see 18.4, “Monitoring performance in DB2” on page 561.

Note: the data collectors and the IBM Web Service Navigator are available as well as stand alone tools. Refer to 18.5, “Stand alone monitoring tools for SOA” on page 561 for more information.

18.3.1 Monitoring agent data collector

ITCAM for SOA works with several applications server environments:

- IBM WebSphere Application Server
- Microsoft .NET
- BEA WebLogic server

Figure 18-5 on page 559 shows the ITCAM for SOA data collection conceptual architecture.

The monitoring agent data collector is implemented as a JAX-RPC handler or service extension that is installed into the application servers that are hosting the monitored Web services. The handler is given control when either of the following events occurs:

- A client application invokes a Web service, which is referred to as a *client size interception.*
- The Web service requester is received by the hosting application server, which referred to as a *server-side interception.*

The monitoring agent records and collects monitored information into one or more local log files. The information is then transferred to Tivoli Enterprise Monitoring Server and can be archived into a historical database for later retrieval with IBM Service Navigator.

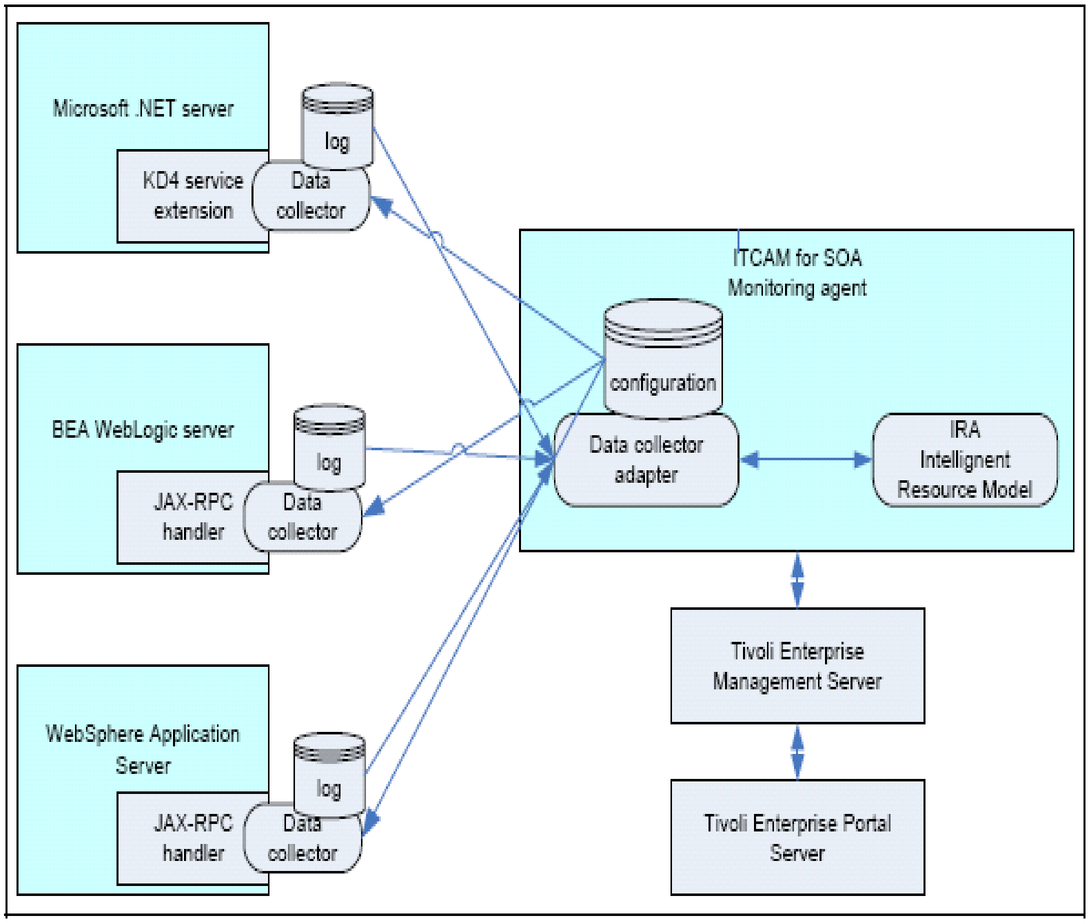


Figure 18-5 ITCAM for SOA structure

ITCM for SOA 6.0 focuses on the Simple Object Protocol (SOAP) engine of IBM WebSphere Application Server, WebSphere Service Integration Bus, the Microsoft .NET Framework and BEA WebLogic.

The Web services data collector supports both J2EE application client and server container environments because JAX-RPC handlers are supported only by these environments. The Web services must be compliant with JSR-109 specifications. For more information about the JSR-109 specification and enterprise Web services implementation refer to:

http://www.jcp.org/aboutJava/communityprocess/final/jsr109

18.3.2 IBM Web Service Navigator

IBM Web Service Navigator is an Eclipse-based tool used to visualize Web services in an SOA environment. it provides a graphical display of:

- Web services transaction flows
- Service topology
- Flow patterns

Figure 18-6 on page 560 illustrates the Web Services Navigator concepts.

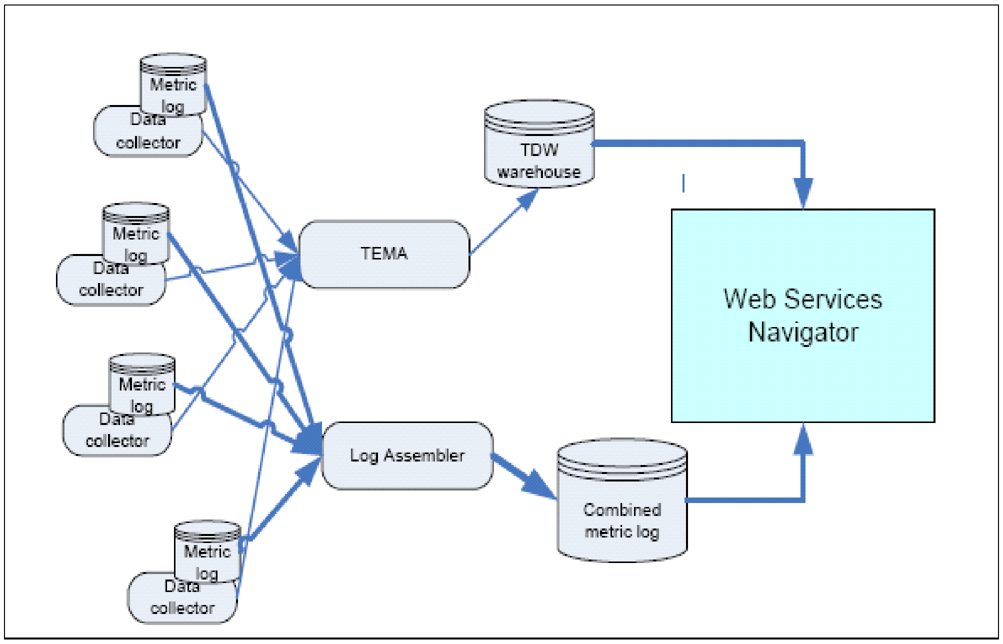


Figure 18-6 Web Service Navigator

The Web Services Navigator is a log-browsing tool intended for offline analysis of SOA Web services. The primary views that are provided:

- Statistic tables:
  - Message statistics

    Pre-message statistics including requestor, provider, send/receive time, and message size.
  - Invocation statistics

    Response time, network delay, message size, and more for each Web service invocation.
  - Transaction statistics

    Provides statistics for aggregated transactions, including elapsed time, number of faults, number of machines this transaction involves, and number of invocations comprising this transaction.
  - Pattern invocation statistics

    Provides statistics for discovered patterns, including operation names, number of occurrences, response times, and message sizes.
- Service topology view

  A graphical representation of the monitored Web services that displays aggregated information and information about the relationships between Web services.
- Transaction flows view

  Transaction flow view displays Universal Markup Language (UML) style sequence diagrams. Transaction flow shows a chronological view of each transaction, the flow between the various Web services over time, and the topology and statistics for each transaction. The view can be zoomed to see the details of individual transactions.

- Flow pattern view

  Flow pattern view is a visual representation of the aggregated pattern of transactions represented in the log file. The view also represents each pattern as a distinct sequence of Web services calls and displays the frequency of each pattern.

18.4 Monitoring performance in DB2

You can monitor, analyze and tune the performance of the Web services inside IBM DB2 Universal Database for z/OS using IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS that gives you a single, comprehensive tool to help assess the efficiency and optimize performance from your DB2 Universal Database on z/OS environment.

Notes: For complete information about this tool, see the redbook *A Deep Blue View of DB2 Performance: IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS*, SG24-7224.

For DB2 on distributed platforms, see*DB2 Performance Expert for Multiplatforms V2.2*, SG24-6470.

18.5 Stand alone monitoring tools for SOA

This topic describes the IBM Web Services Navigator and the Data Collector for the IBM Service Navigator which are stand alone tools for monitoring SOA and Web services applications.

18.5.1 IBM Web Services Navigator

The IBM Web Services Navigator, a plug-in to IBM Rational and other Eclipse-based tools. The Web Services Navigator is used for visualization of Web service transactions.

IBM Web Services Navigator addresses the complexity of understanding and debugging collections of Web services, such as those found in the SOA applications, by visualizing the actual execution of the Web service transactions. The plug-in visualizes logs of Web service activity from IBM WebSphere Application Servers collected by a Web service Data Collector, a companion technology that is released separately. The Web Service Data Collector is discussed in more details in 18.5.2, “Data Collector for IBM Service Navigator” on page 562.

IBM Web Service Navigator allows users to observe and explore the dynamic behavior of their Web Service applications through a new perspective with four new interactive views of messages and transactions:

- The Services Topology view, highlights the services that participated in the transaction and summarizes the messages they exchanged.
- The Transaction Flows view, diagrams the flow of messages from service to service for each transaction.
- The Flow Patterns view, exposes repeated patterns of service interactions between transactions, as well as repeated patterns of invocations within transactions.
- The Message Contents, view reveals the actual contents of individual messages and tracks selected data values through transactions.

Note: for more information and downloading the Web Service Navigator plug-in, refer the the following link:

`http://www.alphaworks.ibm.com/tech/wsnavigator/download`

18.5.2 Data Collector for IBM Service Navigator

The Data Collector for IBM Service Navigator is a tool that intercepts and instruments Web service requests and response and writes information about the Web services to a log file used by IBM Web Services Navigator.

The IBM Service Navigator processes the log files that are generated by the Data Collector and generated visual representation of various characteristics of the monitored Web Services.

The Data Collector is implemented as a JAX-RPC handler that is installed into the application servers that are hosting the monitored Web services and their clients. This handler is given control when either of the following actions occur:

- A client application invokes a Web service. This is referred as a client-side interception.
- The Web service request is received by the hosting application server. This is referred as a server-side interception.

The Web Service Data Collector supplements the Web service request with correlation information do that the flow of the logical transaction can be followed through a series of Web services invocations. The Web services data collector also records monitoring information in a local log file that is imported into the IBM Web Services Navigator. Two or more log files from various monitored applications servers can be manually combined into a single log file. When this combined log file is imported into the IBM Service Navigator, information can be displayed about all of the monitored Web services and the interaction between them at the same time.

The Data Collector focuses on the simple object access protocol (SOAP) engine of WebSphere Application Server.

Notes: The Web services data collector supports only Java 2 Enterprise Edition (J2EE) client and server container environment because JAX-RPC handlers supported only in these environments.

For more information and downloading the Web Service Navigator plug-in, refer the following link:

`http://www.alphworks.ibm.com/tech/wsdatacollector/download`

Part 7

Appendixes

In Part 7 we provide appendixes for additional information about XML and IMS services:

© Copyright IBM Corp. 2006. All rights reserved.

A

XML and DB2

In this appendix we describe the new native XML features common to both DB2 Version 9.1 for z/OS, and DB2 Version 9.1 for Linux, UNIX and Windows. We also describe the new SQL/XML functions available on DB2 9 products on z/OS and Linux, UNIX and Windows. Wherever appropriate we indicate differences among DB2 for z/OS and DB2 for Linux, UNIX and Windows. The details of the two implementations are in Appendix B, “XML and DB2 for z/OS” on page 575 and Appendix C, “XML and DB2 for Linux, UNIX and Windows” on page 593.

© Copyright IBM Corp. 2006. All rights reserved.

A.1 Why use XML in DB2?

Today, XML is predominant in most organizations and hosts an abundance of business information on public and private Web sites. This is because XML is vendor and platform independent, and is a very flexible data model for structured data, semi-structured data, and schema-less data. It is also easy to extend and self-describing. Further, XML can be easily transformed into other XML documents or even different formats such as HTML. Therefore, XML is the de facto standard for exchanging data between different systems, platforms, applications, and organizations.

Beyond XML for data exchange, enterprises are keeping large amounts of business critical data permanently in XML format. This has various reasons, such as a need to keep it for auditing and regulatory compliance. Also, for example in life science applications the data is highly complex and hierarchical in nature and yet may contain significant amounts of unstructured information. Most of today's genomic data is still kept in proprietary flat file formats but major efforts are under way to move to XML. These proprietary flat files can be accessed using WebSphere Federated Server technology.

As a consequence, XML is a core technology for Web applications and Web services. Almost every implementation of service-oriented architecture (SOA) has XML at some point.

We can say that XML is s key technology for:

- Data exchange
- Data integration
- Data evolution and flexibility
- Service-oriented architectures
- Information on demand

DB2 provides significant new capabilities for supporting XML, including a new XML data type and underlying engine-level components that automatically store and process XML data. Let us look at what is new in DB2 V9.1, and how DB2 V9 is significantly different from previous versions of DB2 in terms of XML support it has to offer.

A.2 Native XML support versus XML Extender

Relational is a data model with:

- Relations (tables)
- Attributes (columns)
- Set based with some sequences
- Strict schema

XML is a data model with:

- Nodes (elements, attributes, comments, and so on)
- Relationships between nodes
- Sequence based
- Flexible schema

When we need to store XML data in a Database Management System, traditionally we either store the entire XML document as a CLOB or VARCHAR in a column, or decompose the XML document into rows and columns, or shred the XML into edge table. DB2 XML Extender is a fully-integrated component of DB2 that provides data types to store XML documents in DB2 databases and functions to work with these structured documents. DB2 manages these documents and stores them as character data or external files. You can retrieve complete documents or individual elements via the functions provided by XML Extender. DB2 XML Extender provided rich functionality to store and retrieve XML documents. However, storing XML documents intact in a column, or shredding them to rows in tables still creates a number of problems:

1. The decomposition or shredding methods attempted to fit XML into DB2 which stored the data using the relational model. Usability and performance became an issue as the shredded data is no longer in XML and become unmanageable. Storing the document as CLOB or VARCHAR in an XML column prevents XML parsing at insert
2. Re-construction of the XML document is difficult and expensive. Further it does not make sense to decompose or shred the document when the document always need to re-composed before it can be use efficiently
3. DAD file uses RDB node mapping and XML Extender uses this mapping to determine where to store and retrieve XML data. Mapping becomes very challenging when dealing with large and complex XML documents. Further, mapping is often invalidated when the XML changes.
4. There is no support for XQuery to query the XML documents natively. SQL has to be used instead and application code has to be written to parse the document

Therefore, XML Extender works well, but it does not provide good performance and flexibility. Further it introduces administrative challenges and does not scale well for large, complex applications. You will now learn how the new native XML support in DB2 V9.1 will solve these problems.

A.3 DB2 native XML store

DB2 V9.1 for Linux, UNIX and Windows and DB2 V9.1 for z/OS offer leading-edge technology for storing, managing, and searching XML data in a secure, highly scalable environment. You can now seamless integrate XML with their existing relational data, exploit both tabular and hierarchical data models, and enjoy the flexibility of using both SQL and XQuery in your applications. To provide this first-class support for managing XML data, DB2 features new storage management, indexing, and optimization techniques. It also interfaces to a wide range of popular programming languages, allows users to optionally validate their XML data prior to storage, and extends popular database utilities important for importing data and administering the environment. The new XML support in DB2 manages XML in a hierarchical format, representing the XML data model in a natural way. This provides key advantages over existing offerings that require XML data to be stored as large objects which can be expensive to search and update, or decomposed across multiple relational tables which requires complex schema mappings and query statements.

The ability to integrate business data from multiple sources and services is key to making insightful decisions in today's competitive marketplace. In Version 9.1, DB2 introduces a multi-structure data server with the ability to store both relational and native XML. This feature provides a powerful mechanism for integrating and storing data from various data services such as eForms, documents, XML messages, or other business critical data traditionally not found in a relational data server.

DB2 native XML data store offers several key advantages over other methods of XML data storage:

- Native XML data store is a new storage model which stores the data in a hierarchical format representing the XML data model, rather than confining the XML data to a relational model.

- Native XML data store protects the integrity of your XML data. Shredding XML data into relational tables compromises the digital signatures and other critical metadata that accompany your data. Because DB2 native XML data store does not shred or decompose your XML data, your original XML document, including digital signatures is protected. Native XML data store also allows you to avoid the resource and performance costs associated with rebuilding the XML document every time it is retrieved.
- Native XML data store indexing provides even higher speed search retrieval.
- One of the key benefits of XML is the ability to standardize information which allows for seamless communication with vendors, partners, and customers. DB2 native XML data store provides a robust and flexible foundation upon which service-oriented applications can be built. In addition to fast data retrieval and XML data integrity, DB2 native XML data store provides flexible schema capabilities that allow you to seamlessly and cost-effectively modify application structures without disrupting your data server.
- With DB2 native XML data store, you get the security and stability of DB2 Version 9.1 along with flexible access to XML data using XQuery (only available on DB2 for Linux, UNIX and Windows), XPath, SQL, and standard reporting tools.

Developers in your organization can take advantage DB2 GUI tools (on DB2 UDB for Linux, UNIX and Windows only) to easily create and manage XML structures and build XQuery and SQL statements. You never have to compromise. The DB2 server incorporates the best XML and relational technologies into one server without forcing your XML developers to think like relational developers.

Note: XML Extender is still supported in DB2 Version 9.1 but no enhancements are planned. We recommend you to consider the XML native support for all new applications.

The native XML data store enables well-formed XML documents to be stored in their hierarchical form within columns of a table. XML columns are defined with the XML data type. By storing XML data in XML columns, the data is kept in its native hierarchical form, rather than stored as text or mapped to a different data model. Because the native XML data store is fully integrated into the DB2 database system, the stored XML data can be accessed and managed by leveraging DB2 functionality.

You can use SQL/XML to access XML data on DB2 for z/OS, and you can use both SQL/XML as well as XQuery to access XML data on DB2 for Linux, UNIX and Windows. Figure A-1 on page 568 shows the DB2 V9 architecture, where both relational data storage and native XML storage are accessed by the DB2 engine.

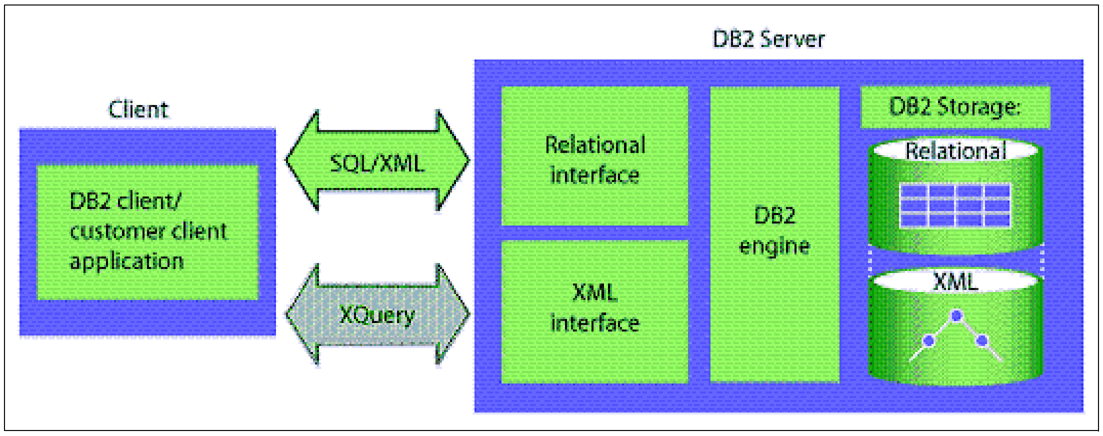


Figure A-1 DB2 Architecture

The XQuery arrow is greyed out, to indicate that XQuery is available on DB2 for Linux, UNIX and Windows only.

Restriction: XQuery is not supported in DB2 for z/OS. Only XPath is supported on DB2 for z/OS. XQuery is available on DB2 for Linux, UNIX and Windows.

The storage of XML data in its native hierarchical form enables efficient search and retrieval of XML. XQuery, SQL, and SQL/XML or a combination of them can be used to query XML data. SQL/XML can enable XML data to be constructed or published from values retrieved from the database. In addition, SQL allows full document retrieval from the native XML store. An XML application can access the native XML store through the XML interface using XQuery, and can also contain optional SQL statements to combine XML data with relational data.

Now that you've learned how to store XML data using DB2's new "native" XML capabilities, you're ready to query that data. You'll see how to do that in subsequent articles, which will introduce you to DB2's new XQuery support as well as its XML extensions to SQL (sometimes called "SQL/XML").

A.4 The XML Data Type

DB2 V9.1 for z/OS, and DB2 for Linux, UNIX and Windows, now features the new support for XML as a first class data type just like any other SQL type. The XML data type can be used in a **`CREATE TABLE`** statement to define a new column of type XML. For example:

```
CREATE TABLE PAOLO.TABLE1 (ID INTEGER, INFO XML)
```

Any column of XML data type can hold one well-formed XML document for every row of the table. The XML data type can also be used as a data type for host variable, and can be accessed via languages like C, Java, COBOL and so on. You can also pass XML type parameters and variables to stored procedures and UDFs.

The introduction of this native XML data type provides the ability to store well-formed XML documents in the database along side other relational data. All XML data is stored in the database in the UTF-8 code set. XML values are processed in an internal representation that is not a string and not directly comparable to string values. An XML value can be transformed into a serialized string value representing the XML document using the XMLSERIALIZE function or by binding the value to an application variable of an XML, string, or binary type. Similarly, a string value that represents an XML document can be transformed to an XML value using the XMLPARSE function or by binding an application string, binary, or XML application type to an XML value. In SQL data change statements (such as INSERT) involving XML columns, a string or binary value that represents an XML document is transformed into an XML value using an injected XMLPARSE function. An XML value can be implicitly parsed or serialized when exchanged with application string and binary data types.

XML columns can be added to existing relational tables using the **`ALTER TABLE`** statement. For example, we have an existing table `PAOLO.TABLE2` and now we want to add a new XML column `CUSTOMER` to this table. Then we can issue the following command:

```
ALTER TABLE PAOLO.TABLE2 ADD CUSTOMER XML
```

A.5 Comparing various XML stores

DB2 provides four ways to store XML. DB2 V8 supports all methods *except* the native XML store. Only DB2 V9.1 provides native XML storage:

- CLOB/VARCHAR: Stored as linear text in database as a CLOB column
- Shred: Decomposed into rows and columns
- Edge: Decomposed into a relational model based on graph theory
- Native: Parsed, stored as interconnected nodes on fixed size database pages

Table A-1 shows the performance attributes, among these store types.

Table A-1 Performance involving XML stores

Measure	CLOB	Shred	Edge	Native
Schema Flexibility	Best	Bad	Best	Best
Search Performance	Bad	Good	Bad	Best
Full document return performance	Good	Bad	Bad	Best
Partial document return performance	Bad	Good	Good	Best
Insert performance	Best	Bad	Bad	Good
Update performance	Bad	Good	Bad	Best
Delete performance	Best	Bad	Bad	Good

As you can see from the table above, native XML store provides the most flexibility to database or XML operations, and best performance across a number of performance considerations.

A.6 XML Index for DB2

Native XML data store provides support for indexing the XML data stored in XML columns. You can create an index on an XML column for efficient evaluation of XPath expressions to improve performance during queries on XML documents. Unlike relational indexes where index keys are composed of one or more table columns that you specified, XML indexes provide access to nodes within the document by creating index keys based on XML patterns. XML index uses a particular XPath expression to index paths and values in XML documents stored in a single XML column. In an XML index, only the attribute nodes, text nodes, or element nodes that match the XML path expression are actually indexed. However, you should note the following differences:

1. **DB2 for z/OS**: When you add an XML column to a table, an XML table and XML table space are implicitly created to store the XML data. If the new XML column is the first XML column that you created for the table, DB2 for z/OS also implicitly creates a BIGINT

DOCID column to store a unique document identifier for the XML columns of a row. To improve performance, DB2 for z/OS also implicitly creates indexes. If this is the first XML column that you created for the table, DB2 implicitly creates an index on the DOCID column of the base table. For each XML column that you add to the base table DB2 implicitly creates a NODEID index on the XML table. The NODEID index maps a given DOCID and NODEID value to a RID value for the XML table. By using the DOCID column, the NODEID index also provides the association from the base table row to the XML data for an XML column of that row.

2. **DB2 for Linux, UNIX and Windows**: You can add XML columns only to tables that do *not* have type-1 indexes defined on them. (Type-1 indexes are deprecated indexes; new indexes since DB2 UDB Version 8.1 are created as type-2 indexes.) Tables to which you add XML columns must be in Unicode databases that exist in instances with only a single database partition defined.

We do not go into detail about XML indexes and performance considerations related to query XML data since it is outside the scope of our book.

Restriction: Partitioned XML indexes are not currently supported in DB2 for z/OS or DB2 for Linux, UNIX and Windows.

A.7 SQL/XML

DB2 now manages both conventional relational data and the new native XML data. The DB2 engine processes SQL, SQL/XML and XPath in an integrated manner since DB2 treats both SQL and XPath as independent primary query languages. Applications can continue to use SQL, and additionally SQL/XML extensions which allows publishing of relational data in XML format. XQuery (available on DB2 for Linux, UNIX and Windows only) is typically used to access the native XML store, and optionally use SQL statements to combine XML data with SQL data. You will learn more detail about XQuery on DB2 for Linux, UNIX and Windows in C.5.3, “XQuery examples for DB2 for Linux, UNIX and Windows” on page 616.

SQL/XML functions are SQL functions that return XML data or take XML arguments. XML data can be queried using an SQL fullselect or with the SQL/XML query functions of XMLQUERY and XMLTABLE. The XMLEXISTS predicate can also be used in SQL queries on XML data. When querying XML data using only SQL, without any XQuery, you can only query at the column level by issuing a fullselect. For this reason, only entire XML documents can be returned from the query; it is not possible to return fragments of a document using only SQL. SQL/XML is designed to bridge the gap between SQL and XML worlds. SQL/XML can be combined with XQuery or XPath expressions within the same statements to query both relational and XML data. Further, you can also use SQL/XML to publish and generate relational data in XML.

Table A-2 shows the SQL/XML functions included in DB2 for z/OS and DB2 for Linux, UNIX and Windows. Some of the existing SQL/XML functions from DB2 V8 is changed, and new functions are provided in DB2 V9 that allow you to construct or publish XML using the new XML data type. Note that some of the SQL/XML functions does not exists on DB2 for z/OS, as indicated in Table A-2.

Table A-2 SQL/XML functions

Function Name	Type	Description	z/OS Version	Linux, UNIX & Windows Version
XML2CLOB	Casting function	Returns a CLOB representation of an XML value.	8 and 9	8 and 9
XMLAGG	Aggregate function	The function produces a forest of XML elements from a collection of XML elements.	8 and 9	8 and 9
XMLATTRIBUTES	Scalar function	The function constructs XML attributes from the arguments.	8 and 9	8 and 9
XMLELEMENT	Scalar function	The function generates an XML element from a variable number of arguments.	8 and 9	8 and 9
XMLNAMESPACES	Scalar function	The function generates XML namespace declarations.	8 and 9	8 and 9
XMLFOREST	Scalar function	The function produces a forest of XML elements that all share a specific pattern from a list of columns and expressions.	8 and 9	8 and 9
XMLCONCAT	Scalar function	The function concatenates a variable number of arguments to generate a forest of XML elements.	8 and 9	8 and 9
XMLCOMMENT	Scalar function	Returns an XML value with a single comment node from a string expression.	only 9	only 9
XMLDOCUMENT	Scalar function	Returns an XML value with a single document node and zero or more nodes as its children.	only 9	only 9
XMLPI	Scalar function	Returns an XML value with a single processing instruction node.	only 9	only 9
XMLQUERY	Scalar function	Returns an XML value from the evaluation of an XPath expression against a set of arguments.	only 9	only 9
XMLSERIALIZE	Scalar function	Returns a SQL character string or a BLOB value from an XML value.	only 9	only 9
XMLTEXT	Scalar function	Returns an XML value with a single text node that contains the value of the argument.	only 9	only 9
XMLPARSE	Scalar function	Parses the argument as an XML document and returns an XML value.	only 9	only 9

Function Name	Type	Description	z/OS Version	Linux, UNIX & Windows Version
XMLVALIDATE DSN_XMLVALIDATE for DB2 z/OS (see [1])	Scalar function	Returns a copy of the input XML value augmented with information obtained from XML schema validation, including default values and type annotations.	does not exist	only Linux, UNIX and Windows V9
XMLOBJECTID	Scalar function	Returns the XSR object identifier of the XML schema used to validate the XML document specified in the argument. The XSR object identifier is returned as a BIGINT value and provides the key to a single row in SYSCAT.XSROBJECTS.	does not exist	only Linux, UNIX and Windows V9
XMLTABLE	Table function	Returns a table from the evaluation of XQuery expressions, possibly using specified input arguments as XQuery variables. Each sequence item in the result sequence of the row XQuery expression represents a row of the result table.	does not exist	only Linux, UNIX and Windows V9

1. The result of DSN_XMLVALIDATE is a varying length binary value of up to 50 MB. The result has no meaning outside of providing input to the XMLPARSE function.

There is also the XMLEXISTS predicate. XMLEXISTS is a predicate that evaluates an XPath expression and returns a Boolean value. If the result of the XPath expression is an empty sequence, XMLEXISTS returns false. If the result of the XPath expression is not empty, it returns true. If the evaluation of the XPath expression returns an error, XMLEXISTS will return an error.

You can now combine strengths of both SQL and XML by using XML extensions in SQL to query both XML and relational data. Further, you can use SQL/XML in additional to conventional SQL to publish relational data in XML format. The advantages of SQL/XML functions can be summarized as follows:

- Easy to extend existing applications with XML using SQL
- Lightweight Web server application
- Submit SQL queries and return results to Web or XML clients
- SQL data to be available in XML form
- Easy integration of XML data with SQL data

Please refer to specific SQL/XML examples for DB2 on z/OS in B.1.2, “The XML publishing functions reference” on page 578, and SQL/XML examples for DB2 on Linux, UNIX and Windows in C.4, “SQL/XML examples” on page 601.

B

XML and DB2 for z/OS

In this appendix we provide additional information related to the contents of the current and the new XML support in DB2 for z/OS.

- The XML support in DB2 for z/OS
- What DB2 Version 9.1 for z/OS brings to XML support

© Copyright IBM Corp. 2006. All rights reserved.

B.1 The XML support in DB2 for z/OS

For the past few years, XML has been increasingly become the de fact data format on the Internet, on corporate intranets, and for data exchange. In DB2 for z/OS V7, if you need to create XML data from traditional relational databases (this is called XML publishing or XML composition), you must create your own application that converts the DB2 data to the XML format, or use DB2 XML Extender. Version 8 of DB2 for z/OS provides a set of SQL built-in functions that allow applications to generate XML data from relational data. These functions reduce application development efforts for generating XML data for data integration, information exchange, and Web services.

Note: DB2 Version 9.1 for z/OS provides several enhancements to XML like XPath access, Native Storage, XML column and indexes, we describe it in the next sections.

B.1.1 What has DB2 already provided?

XML values

An XML data type is an internal representation of XML, and can be used only as input to built-in functions that accept this data type as input. XML is a transient data type that cannot be stored in the database or returned to an application. Valid values for the XML data type include the following:

- An element
- A forest of elements
- The textual content of an element
- An empty XML value

Mappings from SQL to XML

To construct XML data from SQL data, the following mappings are performed:

- SQL character sets to XML character sets.
- SQL identifiers to XML names.
- SQL data values to XML data values.

DB2 maps SQL to XML data according to industry standards.

See Mapping XML to Relational (Shredding) on Figure B-1 on page 577.

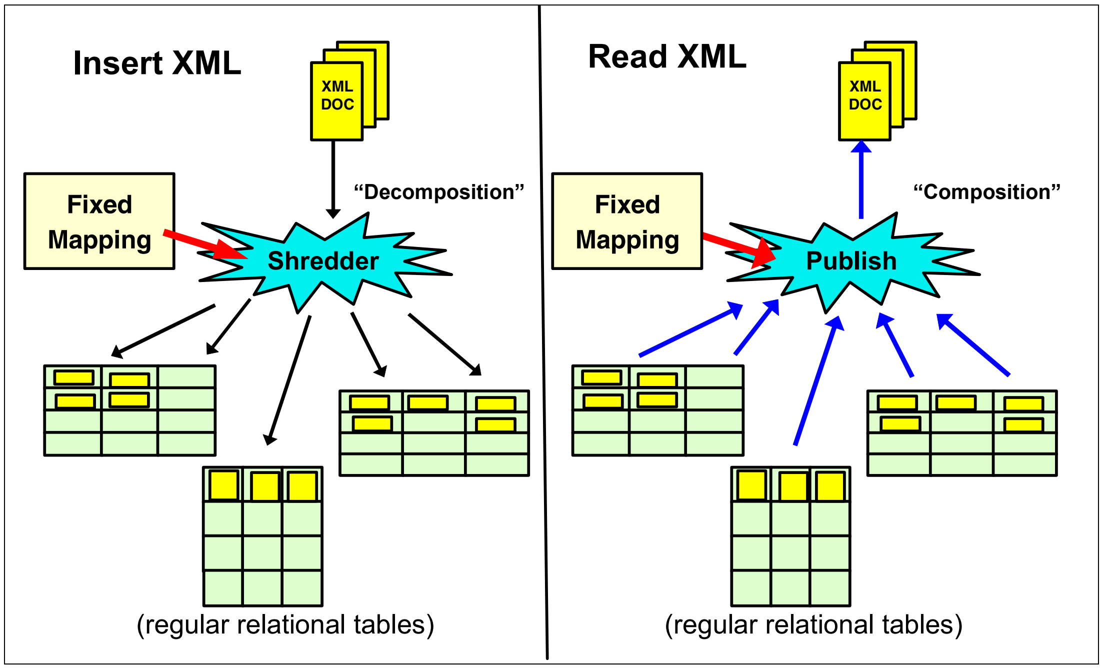


Figure B-1 Shredding

Note: For more information, see Information technology - Database languages - SQL- Part 14: XML-Related Specifications (SQL/XML) ISO/IEC 9075-14:2003.

Mapping SQL character sets to XML character sets

The character set used for XML data is Unicode UTF-8. SQL character data is converted into Unicode when it is used in XML built-in functions.

Mapping SQL identifiers to XML names

Strings that start with "XML", in any case combination, are reserved for standardization, and characters such as "#"," {", and "}" are not allowed in XML names. Many SQL identifiers containing these characters have to be escaped when converting into XML names.

Full escaping is applied to SQL identifiers that are column names to derive an XML name. The mapping converts a colon (:) to _x003A_, _x to _X005F_x, and other restricted characters to a string of the form _xUUUU_ where xUUUU_ is the Unicode value for the character. An identifier with an initial "xml" (in any case combination) is escaped by mapping the initial "x" or "X" to _x0058_ or _0078_, respectively, while the partially escaped variant does not.

Mapping SQL data values to XML data values

SQL data values are mapped to XML values based on SQL data types. The following data types are not supported and cannot be used as arguments to XML value constructors:

ROWID
Character sting defined with the FOR BIT DATA attribute.
BLOB
Distinct types based on ROWID, FOR BIT DATA character string or BLOB.

Note: For supported data types, the encoding scheme for XML values is Unicode.

B.1.2 The XML publishing functions reference

Note: This a quick reference to the SQL commands, but for more information use *DB2 UDB for z/OS V8 SQL Reference,* SC18-7426.

Figure B-2 shows the Syntax command to use, XML2CLOB.

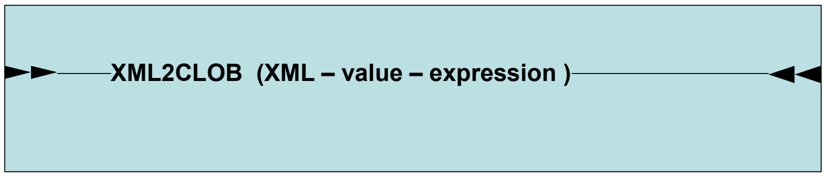


Figure B-2 Syntax command to use XML2CLOB

The XML data type is a new data type introduced by DB2 V8. However, it is not like any other existing data type. It is a so-called transient data type. Transient means that this data type only exists during query processing. There is no persistent data of this type and it is not an external data type that can be declared in application programs. In other words, the XML data type cannot be stored in a database or returned to an application.

To allow an application to deal with the result of a SQL/XML function (that results in a value with an XML data type), DB2 supplies a new conversion function XML2CLOB, which converts an XML value into a CLOB.

There are some restrictions that apply to the transient XML data type:

- A query result cannot contain this type.
- The columns of a view cannot be of this type. XML data cannot be used in SORT (GROUP BY and ORDER BY). XML data cannot be used in predicates.
- The XML data type is not compatible with any other data types. The only cast function that may be used is XML2CLOB.

The resulting CLOB is MIXED character data and the CCSID is the mixed CCSID for UNICODE encoding scheme UTF-8 (CCSID 1208).

The maximum length of the resulting CLOB is 2 GB -1.

To Construct a CLOB from the XML value returned by the XMLELEMENT function, which is a simple XML element with “Emp” as the element name and employee name as the element content. See Example B-1.

Example: B-1 Using XML2CLOB

```
SELECT E.EMPNO, XML2CLOB(XMLELEMENT ( NAME "EMP", E.FIRSTNME || ' ' ||
E.LASTNAME ) ) AS "RESULT" FROM DSN8910.EMP E
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO   RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
000010  <EMP>CHRISTINE HAAS</EMP>
000020  <EMP>MICHAEL THOMPSON</EMP>
000030  <EMP>SALLY KWAN</EMP>
000050  <EMP>JOHN GEYER</EMP>
000060  <EMP>IRVING STERN</EMP>
000070  <EMP>EVA PULASKI</EMP>
000090  <EMP>EILEEN HENDERSON</EMP>
```

```
000100  <EMP>THEODORE SPENSER</EMP>
000110  <EMP>VINCENZO LUCCHESI</EMP>
```

Note: All functions described in this book have schema SYSIBM.

XMLELEMENT

Figure B-3 shows the Syntax command to use XML2CLOB.

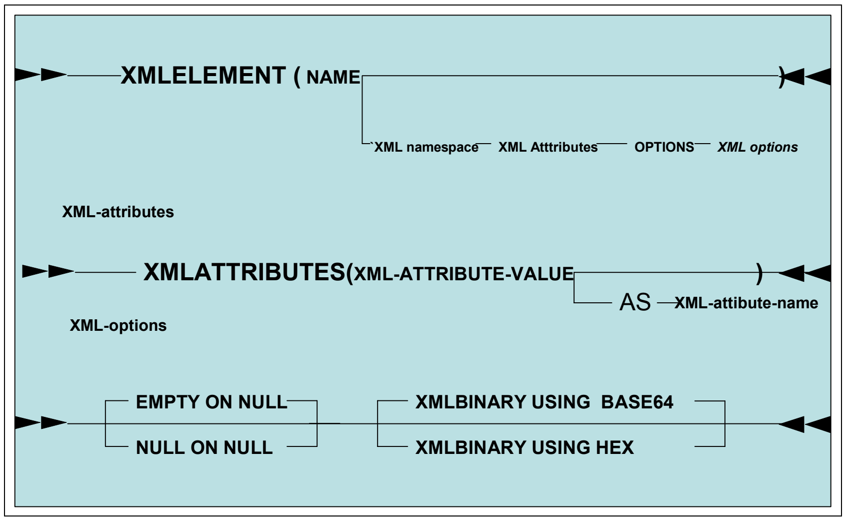


Figure B-3 Syntax command to use XMLELEMENT

The XMLELEMENT function returns an XML element from one or more arguments. The arguments can be:

- An element name
- An optional collection of attributes
- Zero or more arguments that make up the element's content.

The result type is the transient XML data type.

Let us look at the components of the XMLELEMENT function:

NAME

keyword marks the identifier that is supplied to XMLELEMENT for the element name.

XML-element-name

Specifies an identifier that is used as the XML element name. (No mapping is applied to this identifier.)

XML-namespaces

Specifies the XML namespace for the XML element.

XML-attributes

Specifies the attributes for the XML element

expression

Specifies an expression making up the XML element content.

The expression cannot be:

- A ROWID
- A character string defined with the FOR BIT DATA attribute
- A BLOB
- A distinct type sourced on these types

The result of the XMLELEMENT function cannot be null.

Refer to the SELECT statement shown on the visual for a short and simple example of the use of the XML2CLOB and XMLELEMENT function. As you can see in the SQL statement above, the XMLELEMENT function is used to create an element called EMP, which contains the concatenation of the contents of columns FIRSTNME and LASTNAME, see Example B-2.

Example: B-2 Using XMLELEMENT

```
SELECT e.empno, XML2CLOB(
XMLELEMENT ( NAME "Emp",
XMLELEMENT ( NAME "name", e.firstnme ||' ' ||e.lastname ),
XMLELEMENT ( NAME "hiredate", e.hiredate ))
AS "Result"
FROM dsn8810.emp e;
```

Note: As you can see, element <Emp> itself contains two nested elements <name> and <hiredate>.

Example B-3 shows the result of the SELECT statement used in the previous example:

Example: B-3 Result of XMLELEMENT SELECT

```
---------+---------+---------+---------+---------+---------+---------+---------+
EMPNO RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
000010 <EMP><NAME>CHRISTINE HAAS</NAME><HIREDATE>1965-01-01</HIREDATE></EMP>
000020 <EMP><NAME>MICHAEL THOMPSON</NAME><HIREDATE>1973-10-10</HIREDATE></EMP>
000030 <EMP><NAME>SALLY KWAN</NAME><HIREDATE>1975-04-05</HIREDATE></EMP>
```

For each employee ID, create an empty XML element named Emp with an ID attribute equal to the ID, see Example B-4.

Example: B-4 Selecting XML

```
SELECT EMPNO, XML2CLOB ( XMLELEMENT ( NAME "EMP", XMLATTRIBUTES (EMPNO)))
AS "RESULT" FROM DSN8910.EMP ;
EMPNO   RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
000010  <EMP EMPNO="000010"></EMP>
000020  <EMP EMPNO="000020"></EMP>
000030  <EMP EMPNO="000030"></EMP>
000050  <EMP EMPNO="000050"></EMP>
000060  <EMP EMPNO="000060"></EMP>
000070  <EMP EMPNO="000070"></EMP>
000090  <EMP EMPNO="000090"></EMP>
000100  <EMP EMPNO="000100"></EMP>
000110  <EMP EMPNO="000110"></EMP>
000120  <EMP EMPNO="000120"></EMP>
```

XMLNAMESPACES

Figure B-4 shows the Syntax command to use XMLNAMESPACES.

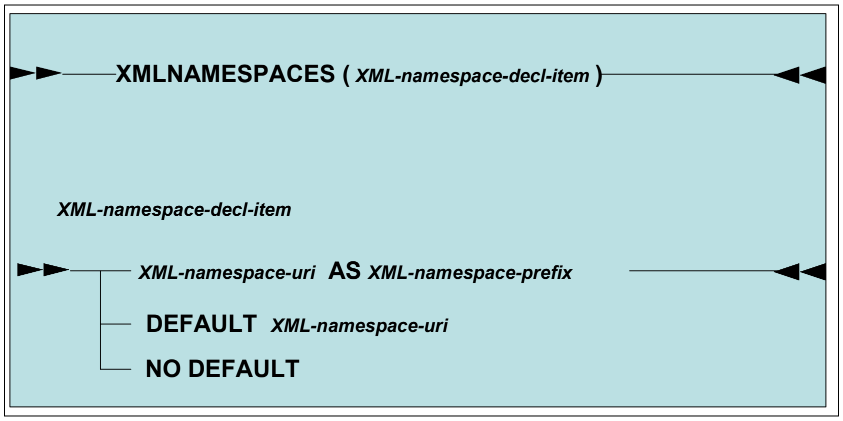

XMLNAMESPACES (XML-namespace-decl-item)
XML-namespace-decl-item
XML-namespace-uri AS XML-namespace-prefix
DEFAULT XML-namespace-uri
NO DEFAULT

Figure B-4 Syntax command to use XMLNAMESPACES

The XMLNAMESPACES function declares one or more XML namespaces.

XML-namespace-uri

A character string literal that is the namespace name. It cannot be a UX, GX, or graphic string literal. XML-namespace-uri can be an empty string constant only if it is being specified for DEFAULT.

To Generate an “employee” element for each employee. The employee element is associated with XML namespace “urn:bo”, which is bound to prefix “bo”. The element contains attributes for names and a hiredate subelement. See Example B-5.

Example: B-5 Using XMLNAMESPACES

```
SELECT E.EMPNO, XML2CLOB(XMLELEMENT(NAME "BO:EMPLOYEE",
XMLNAMESPACES('URN:BO' AS "BO"), XMLATTRIBUTES(E.LASTNAME, E.FIRSTNME),
XMLELEMENT(NAME "BO:HIREDATE", E.HIREDATE))) FROM DSN8910.EMP E WHERE
E.EDLEVEL = 12;
EMPNO
---------+---------+---------+---------+---------+---------+---------+---------+
000290  <BO:EMPLOYEE xmlns:BO="URN:BO" LASTNAME="PARKER"
FIRSTNME="JOHN"><BO:HIREDATE>19
```

To generate two elements for each employee using XMLFOREST. The first “lastname” element is associated with the default namespace “http://hr.org”, and the second “job” element is associated with XML namespace “http://fed.gov”, which is bound to prefix “d”?. See Example B-6.

Example: B-6 Selecting namespaces

```
SELECT EMPNO, XML2CLOB(XMLFOREST( XMLNAMESPACES(DEFAULT 'HTTP://HR.ORG',
'HTTP://FED.GOV' AS "D"), LASTNAME, JOB AS "D:JOB")) FROM DSN8910.EMP
WHERE EDLEVEL = 12;

EMPNO
---------+---------+---------+---------+---------+---------+---------+---------+
000290  <LASTNAME xmlns="HTTP://HR.ORG"
xmlns:D="HTTP://FED.GOV">PARKER</LASTNAME><D:JOB
```

```
000310  <LASTNAME xmlns="HTTP://HR.ORG"
xmlns:D="HTTP://FED.GOV">SETRIGHT</LASTNAME><D:JOB
```

XMLFOREST

Figure B-5 shows the Syntax command to use XMLFOREST.

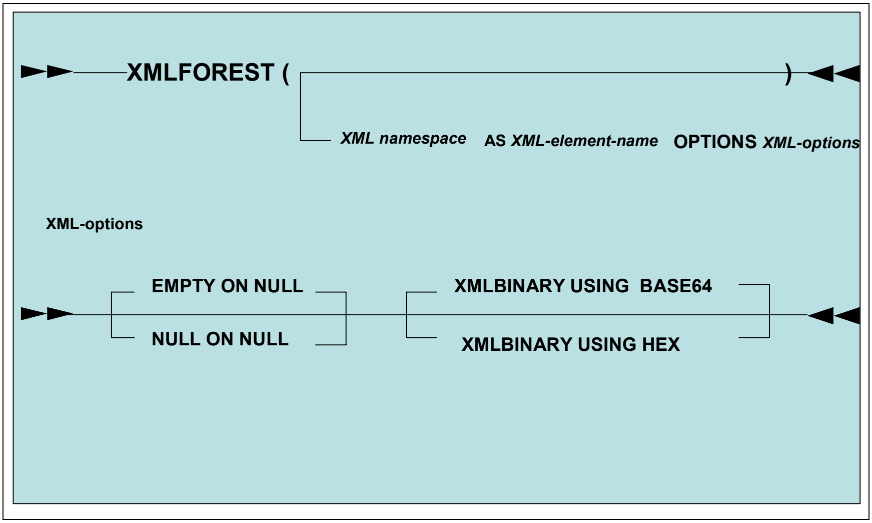


Figure B-5 Syntax command to use XMLFOREST

The XMLFOREST function returns a bunch of XML elements that all share a specific pattern from a list of expressions, one element for each argument.

content-expression:

Specifies an expression that is used as an XML element content. The result of the expression is mapped to an XML value according to the mapping rules from an SQL value to an XML value.

See on Example B-7, nested elements instead of XMLFOREST.

Example: B-7 Nested elements

```
SELECT E.EMPNO,XML2CLOB(
XMLELEMENT ( NAME "EMP",
XMLATTRIBUTES(E.FIRSTNME||' '||E.LASTNAME AS "NAME"),
XMLELEMENT(NAME "HIREDATE", E.HIREDATE),
XMLELEMENT(NAME "PROFESSION", E.JOB)))
AS "RESULT"
FROM DSN8810.EMP E;
```

Note: The generated element names are folded to uppercase. If you want them to be lowercase or mixed, you must use quotes (“department”). In the examples used in this section, there would be a difference between XMLFOREST and XMLELEMENT if there were NULL values in HIREDATE and JOB.

XMLFOREST ignores the .NULL value (not included in the result) and XMLELEMENT results in an empty element.

Generate an “Emp” element for each employee. Use employee name as its attribute and two subelements generated from columns HIRE and DEPT by using XMLFOREST as its content. The element names for the two subelements are “HIRE” and “department”. See Example B-8.

Example: B-8 Using XMLFOREST

```
SELECT e.id, XML2CLOB ( XMLELEMENT ( NAME "Emp", XMLATTRIBUTES ( e.fname || ' ' ||
e.lname AS "name" ), XMLFOREST ( e.hire, e.dept AS "department" ) ) ) AS "result"
FROM employees e;
ID result
---------------------------------------------
1001 <Emp name="John Smith"> <HIRE>2000-05-24</HIRE>
<department>Accounting</department> </Emp>
1001 <Emp name="Mary Martin"> <HIRE>1996-02-01</HIRE>
<department>Shipping</department> </Emp>
```

XMLCONCAT

Figure B-6 shows the Syntax command to use XMLCONCAT.

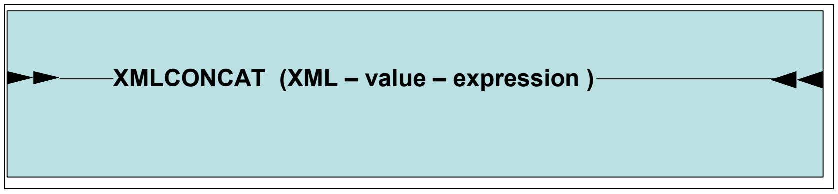


Figure B-6 Syntax command to use XMLCONCAT

The XMLCONCAT function returns a forest of XML elements that are generated from a concatenation of two or more arguments.

XML-value-expression:

Specifies an expression whose value is the XML data type. If the value of XML-value-expression is null, it is not included in the concatenation.

The result type of XMLCONCAT is the transient XML data type. If all of the arguments are null, then the null value is returned.

See Example B-9 XMLFOREST instead of XMLCONCAT.

Example: B-9 Using XMLFOREST

```
SELECT E.EMPNO,XML2CLOB(
XMLFOREST(E.FIRSTNME AS "FIRST", E.LASTNAME AS "LAST"))
AS "RESULT"
FROM DSN8810.EMP E WHERE LASTNAME = 'HAAS';
```

See Example B-10 using XMLCONCAT.

Example: B-10 XMLCONCAT

```
SELECT E.EMPNO,XML2CLOB(XMLCONCAT (XMLELEMENT(NAME "FIRST",E.FIRSTNME),
XMLELEMENT(NAME "LAST",E.LASTNAME)))
AS "RESULT" FROM DSN8910.EMP E;
EMPNO   RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
000010  <FIRST>CHRISTINE</FIRST><LAST>HAAS</LAST>
```

```
000020  <FIRST>MICHAEL</FIRST><LAST>THOMPSON</LAST>
000030  <FIRST>SALLY</FIRST><LAST>KWAN</LAST>
000050  <FIRST>JOHN</FIRST><LAST>GEYER</LAST>
000060  <FIRST>IRVING</FIRST><LAST>STERN</LAST>
```

Note: One reason for using XMLCONCAT instead of XMLFOREST is that XMLFOREST cannot generate XML elements with attributes. For this purpose, use XMLELEMENT.

To Concatenate first name and last name elements by using “first” and “last” element names for each employee. See Example B-11.

Example: B-11 Concatenating names

```
SELECT XML2CLOB( XMLCONCAT ( XMLELEMENT ( NAME "first", e.fname), XMLELEMENT (
NAME "last", e.lname) ) ) AS "result" FROM employees e;
The result of the query would look similar to the following result, where the
.result.column is a CLOB:
result
---------------------------------------------
<first>John</first><last>Smith</last>
<first>Mary</first><last>Smith</last>
```

XMLAGG

Figure B-7 shows the Syntax command to use XMLAGG.

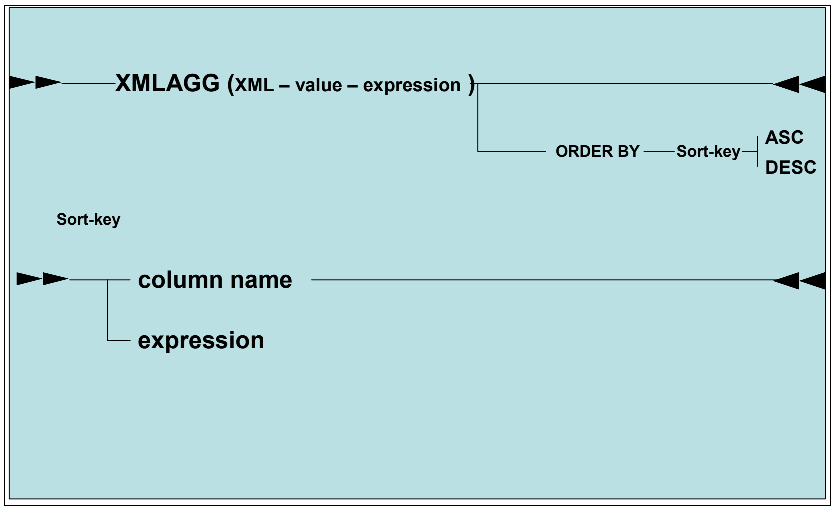


Figure B-7 Syntax command to use XMLAGG

The XMLAGG function has one argument with an optional ORDER BY clause. The ORDER BY clause specifies the ordering of the rows from the same grouping set to be processed in the aggregation. If the ORDER BY clause is not specified, or the ORDER BY clause cannot differentiate the order of the sort key value, the order of rows from the same group to be processed in the aggregation is arbitrary.

XML-value-expression

Specifies an expression whose value is the transient XML data type. sort-key:

To Group employees by their department, generate a “Department” element for each department with its name as the attribute, nest all the “emp” elements for employees in each department, and order the “emp” elements by “lname.”. See Example B-12.

Example: B-12 Using XMLAGG

```
SELECT XML2CLOB (XMLELEMENT ( NAME "Department", XMLATTRIBUTES ( e.dept AS "name"
), XMLAGG ( XMLELEMENT ( NAME "emp", e.lname) ORDER BY e.lname) ) ) AS "dept_list"
FROM employees e GROUP BY dept ;

dept_list
---------------------------------------------
<Department name="Accounting">
<emp>SMITH</emp>
<emp>Yates</emp>
</Department>
<Department name="Shipping">
<emp>Martin</emp>
<emp>Oppenheimer</emp>
</Department>
------------------------------------------------
```

B.2 What DB2 Version 9.1 for z/OS brings to XML support

Version 9.1 brings to DB2 several features about XML:

- Native (hierarchical) Storage
- XML column
- Sophisticated XML Indexes
- XPATH functions
- XMLEXISTS predicate

B.2.1 Native XML storage

Important: DB2 V9.1 for z/OS does not support XQUERY (SQL/XML) like DB2 for V9.1 for Linux, UNIX and Windows.

- DB2 stores XML in parsed hierarchical format.
- Relational columns are stored in relational format (tables).
- XML columns are stored natively in the trees models.

Figure B-8 on page 586 shows the tree structure that stores the XML.

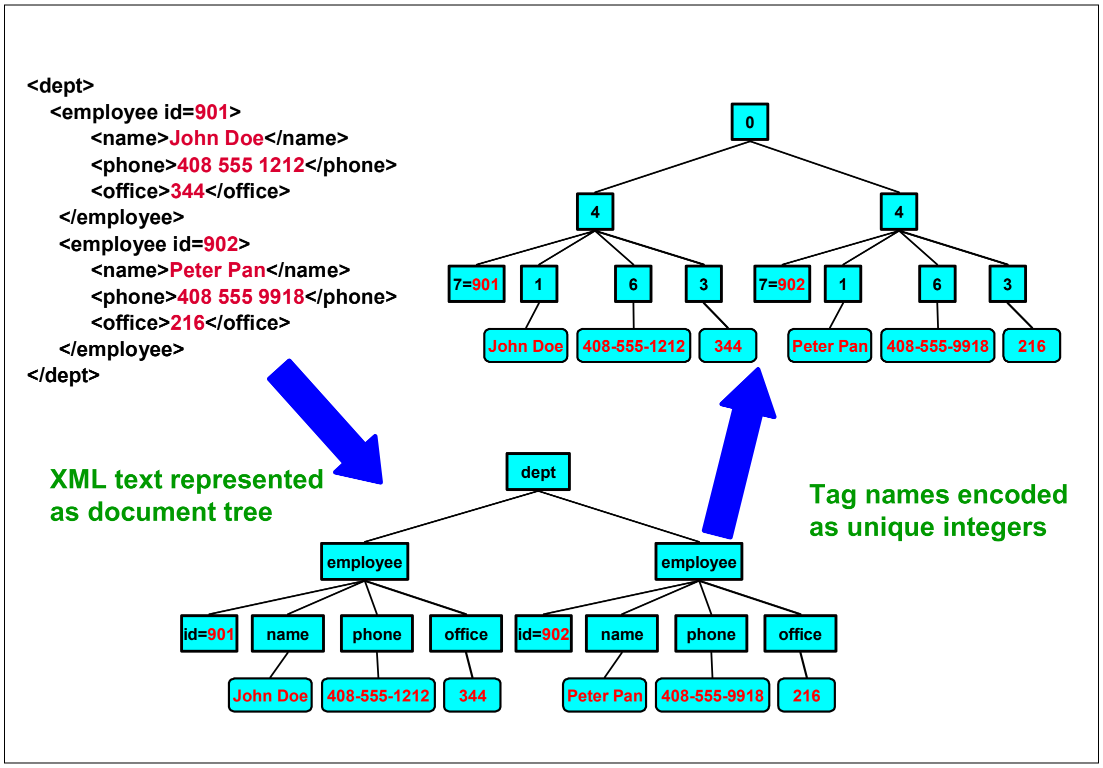


Figure B-8 Trees model to store XML

B.2.2 Using XMLEXISTS to select XML data

Note: To use this predicate you need create a XML column. To improve performance is possible as well you create a XML index, these feature ate common with DB2 for LUW, see about it in Appendix A, "XML and DB2" on page 565.

XMLEXISTS is a predicate that evaluates an XPath expression and checks whether the result is an empty sequence. If the result of the XPath expression is an empty sequence, XMLEXISTS returns false. If the result is not empty, it returns true. If the evaluation of the XPath expression returns an error, XMLEXISTS returns an error.

Restriction: XMLEXISTS cannot be used in the ON clause of outer joins.

Suppose that you want to find a purchase order that has a billing address (billTo). You can use the SELECT statement with XMLEXISTS in the predicate showed in Example B-13.

Example: B-13 Using XMLEXISTS

```
SELECT DESC_SOA FROM PAOLOR1.TBSOA WHERE XMLEXISTS ('declare namespace
ipo="http://www.example.com/IPO"; /ipo:purchaseOrder[billto]' PASSING
XML_COLUMN);
DESC_SOA
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE610I NUMBER OF ROWS DISPLAYED IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

B.2.3 Query performance using XML

XML data, by its nature, degrades the performance of most SQL statements. This degradation occurs because each row of an XML column contains an XML document, requires more processing and more space in DB2. When you use the XPath expression to search or extract the XML data, you can lessen the performance impact by avoiding the descendant or descendant-or-self axis in the expression. The XMLEXISTS predicate is always stage 2. However, you can use the XML value index to reduce the number of rows, that is, the number of XML documents, to be searched at the second stage. for how to design your XML value index so that the XPath predicates in the XMLEXISTS predicate can be used as the matching predicate in the matching index scan. Note that creating and maintaining the XML value index is more costly than the non-XML value index. You should, if possible, design your query to utilize the non-XML value indexes to filter as many rows as possible before the second stage.

Suppose that you issue the following SELECT statement that uses the value indexes to evaluate the XMLEXISTS predicates. See Example B-14.

Example: B-14 Select statement

```
SELECT * FROM T WHERE (C1 = 1 OR C2 = 1) AND XMLEXISTS('/a/b[c = 1]' PASSING
XML_COL1) AND XMLEXISTS('/a/b[(e = 2 or /f[g] = 3) and /h/i[j] = 4]' PASSING
XML_COL2);
```

B.2.4 The XPath functions reference

You can select specific values from XML data that is stored in DB2 by using an XPath expression in an SQL SELECT statement. XPath expressions address specific nodes in an XML document, much as SQL predicates can address specific values in a relational database.

Note: For general information about XPath, refer to:

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

You can use these XMLQUERY and other new functions to handle XML, we describe a brief informations about each one.

Note: This a quick reference to the SQL commands, for more informations use SQL reference.

XMLQUERY

Figure B-9 on page 588 shows the Syntax command to use XMLQUERY.

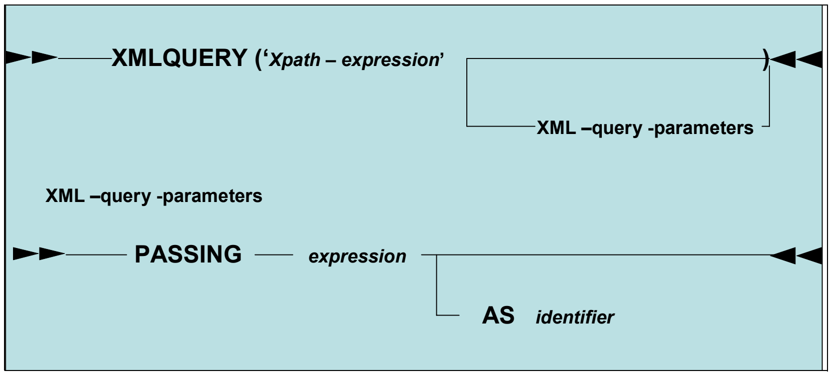


Figure B-9 Syntax command to use XMLQUERY

The XMLQUERY function returns an XML value from evaluation of an XPath expression against a set of arguments.

XPath-expression

Specifies an XPath expression to evaluate. See the XML **GETTING STARTED GUIDE** for more information about XPath and DB2.

XML-query-parameters

Specifies parameters on which the XPath expression is evaluated.

expression

Specifies an expression whose result is used as an argument to the XPath expression.

AS identifier

Specifies an identifier that is used as an XPath variable name. The name must be an XML NCName. If the AS clause is not specified, the expression has no name.

An XPath variable is created for each argument in the PASSING clause that is assigned an identifier as a name. The value of the variable is the result of the expression when cast to an XML type. If the result of the expression is a null, the empty sequence is assigned to the XPath variable.

An expression that is not named using the AS clause becomes the XPath expression's initial context item. Only one unnamed expression can exist. This expression is called context argument. If the data type of the context argument is not the XML type, its result is cast to the XML type. If the result of the context argument is the NULL value or the empty sequence, the XMLQUERY returns the NULL value of the XML type.

Example B-15 returns an XML value from evaluation of the specified XPath expression.

Example: B-15 Using XMLQUERY

```
SELECT XMLQUERY('//item[productName=$n]' PASSING PO.POrder, :hv AS "n") AS
"Result" FROM PurchaseOrders PO;
Assume that the value of the host variable (:hv) is 'Baby Monitor', the result is
similar to the following results:
Result
-----------------------------------------------------------------------
<item partNum="926-AA"><productName>Baby Monitor</productName><quantity>1
</quantity><USPrice>39.98</USPrice><shipDate>1999-05-21</shipDate></item>
```

XMLPI

Figure B-10 shows Syntax command to use XMLPI.

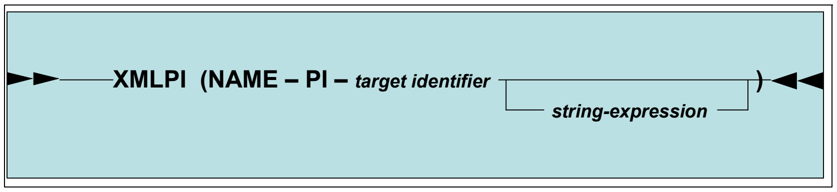


Figure B-10 Syntax command to use XMLPI

The XMLPI function returns an XML value with a single processing instruction node. The target of the processing instruction node is the input identifier NAME, which is an NCName, Its content is the character string value that is trimmed of leading blanks and mapped to Unicode (UTF-8). The result type is XML.

To Generate an XML processing instruction node, seeExample B-16.

Example: B-16 Using XMLPI

```
SELECT XMLPI(NAME "SUBMIT", 'XPUSH THE BUTTON') FROM
SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+------
<?xml version="1.0" encoding="IBM037"?><?SUBMIT XPUSH THE BUTTON?>
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

MLSERIALIZE

Figure B-11 shows the Syntax command to use XMLSERIALIZE.

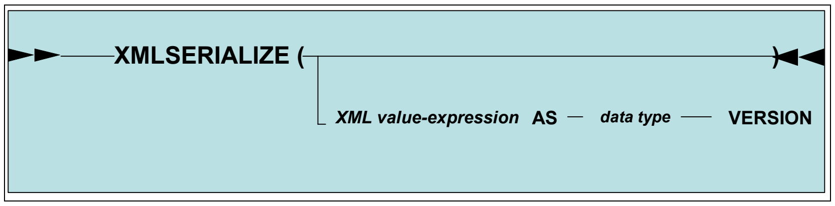


Figure B-11 Syntax command to use XMLSERIALIZE

The XMLSERIALIZE function returns a SQL character string or a BLOB value from an XML value. The character string or BLOB contains the serialized form of the XML value. The result type is the specified data type (CLOB, DBCLOB or BLOB). The maximum length of the result is also specified with the data type.

The result can be null; if the argument is null, the null value is returned.

To Serialize into CLOB of UTF-8, the XML value that is returned by the XMLELEMENT function, which is a simple XML element with “Emp” as the element name, and employee name as the element content. See Example B-17.

Example: B-17 Using XMLSERIALIZE

```
SELECT XMLSERIALIZE(XMLDOCUMENT( XMLELEMENT(NAME "EMP", E.FIRSTNME || ''
|| E.LASTNAME)) AS CLOB(100)) AS RESULT
FROM DSN8910.EMP      E WHERE E.EMPNO < '000050';
---------+---------+---------+---------+---------+---------+---------+----
RESULT
---------+---------+---------+---------+---------+---------+---------+----
```

```
<EMP>CHRISTINEHAAS</EMP>
<EMP>MICHAELTHOMPSON</EMP>
<EMP>SALLYKWAN</EMP>
DSNE610I NUMBER OF ROWS DISPLAYED IS 3
```

To Serialize into a string of BLOB type using character set US-ASCII, the XML value that is returned by the XMLELEMENT function. See Example B-18.

Example: B-18 Using BLOB

```
SELECT XMLSERIALIZE(XMLDOCUMENT( XMLELEMENT(NAME "emp", e.fname || ' ' ||
e.lname)) AS BLOB(1K) ENCODING "US-ASCII" VERSION '1.0') as result FROM employee e
WHERE e.id = '1001';
result :
---------------------------------------------------------------
<?xml version="1.0" encoding="US-ASCII"?><emp>John Smith</emp>
```

XMLDOCUMENT

Figure B-12 shows the Syntax command to use XMLDOCUMENT

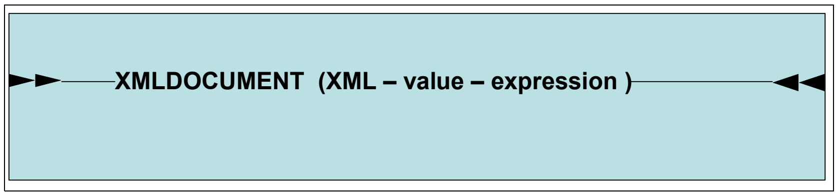


Figure B-12 Syntax command to use XMLDOCUMENT

The XMLDOCUMENT function returns an XML value with a single document node and zero or more nodes as its children. The result type is XML.

The result can be null; if the argument is null, the null value is returned.

XML-value-expression

An expression whose value is of the XML type.

Example B-19 shows an Insert a constructed document into an XML column.

Example: B-19 Using XMLDOCUMENT

```
INSERT INTO T1 VALUES(123, (SELECT XMLDOCUMENT(XMLELEMENT(NAME "Emp", e.fname || '
' || e.lname), XMLCOMMENT('This is just a simple example')) FROM EMPLOYEE e WHERE
e.empid = 123));
```

XMLCOMMENT

Figure B-13 on page 590 shows the Syntax command to use XMLCOMMENT.

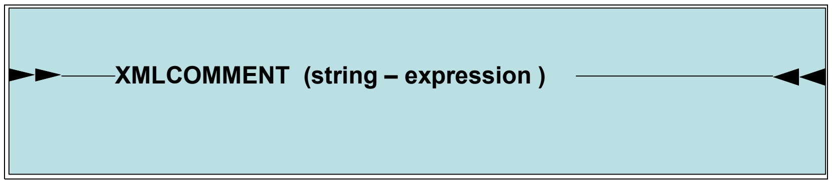


Figure B-13 Syntax command to use XMLCOMMENT

The XMLCOMMENT function returns an XML value with a single comment node from a string expression. The contents of the comment node are the value of the input string expression mapped to Unicode (UTF-8). The result type is XML. The result can be null; if the argument is null, the null value is returned.

string-expression:

An expression whose value has one of the character string types: CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC. FOR BIT DATA is not supported.

Example B-20 shows how to Generate an XML comment:

Example: B-20 Using XMLCOMMENT

```
SELECT XMLCOMMENT('THIS IS AN XML COMMENT') FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+---------
<?xml version="1.0" encoding="IBM037"?><!--THIS IS AN XML COMMENT-->
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

XMLTEXT

Figure B-14 shows the Syntax command to use XMLTEXT.

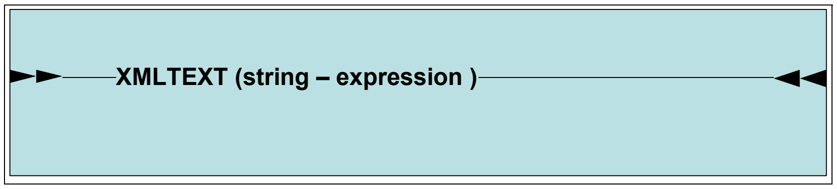


Figure B-14 Syntax command to use XMLTEXT

The XMLTEXT function returns an XML value with a single text node that contains the value of the string-expression. The result type is XML. The result can be null; if the argument is null, the null value is returned.

string-expression

An expression whose value has one of the character string types: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC, CLOB, and DBCLOB. FOR BIT DATA is not supported.

Example B-21 returns an XML value with a single text node that contains the specified value:

Example: B-21 Using XMLTEXT

```
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT XMLTEXT('THE STOCK SYMBOL FOR JOHNSON&JOHNSON.') AS
"RESULT" FROM SYSIBM.SYSDUMMY1;
---------+---------+---------+---------+---------+---------+---------+---------+
RESULT
---------+---------+---------+---------+---------+---------+---------+---------+
<?xml version="1.0" encoding="IBM037"?>THE STOCK SYMBOL FOR JOHNSON&amp;JOHNSON.
DSNE610I NUMBER OF ROWS DISPLAYED IS 1
```

The XMLTEXT function enables the XMLAGG function to construct mixed content, as in Example B-22.

Example: B-22 XMLAGG with XMLTEXT

```
SELECT XMLELEMENT(NAME "para", XMLAGG(XMLCONCAT( XMLTEXT( plaintext), XMLELEMENT(
NAME "emphasis", emphtext )) ORDER BY seqno ), '.' ) as "result" FROM T;
Suppose that the content of the table T is as the following:
seqno plaintext emphtext
----- ------------------------------------------------ ----------------
1 This query shows how to construct mixed content
2 using XMLAGG and XMLTEXT. Without XMLTEXT
3 XMLAGG cannot group text nodes with other nodes, mixed content therefore, cannot
generate
The result looks like the following result: result
--------------------------------------------------------------------------
<para>This query shows how to construct <emphasis>mixed content</emphasis> using
XMLAGG and XMLTEXT. Without <emphasis>XMLTEXT</emphasis>, XMLAGG cannot group text
nodes with other nodes, therefore, cannot generate <emphasis>mixed
content</emphasis>.</para>
```

C

XML and DB2 for Linux, UNIX and Windows

In this appendix we further discuss how DB2 for Linux, UNIX and Windows is being extended to provide support for XML. SQL/XML and XQuery examples specific to DB2 for Linux, UNIX and Windows are provided, and DB2 tools only available to these platforms will be introduced.

This appendix contains these topics:

- New features of native XML data store in DB2 V9.1 for Linux, UNIX and Windows
- Using the native XML data store in DB2 V9.1 for Linux, UNIX and Windows
- XML schema support
- SQL/XML examples
- Comparison of XML data access methods
- Annotated XML schema decomposition
- XML APIs and application support
- Create and register an XML schema using Developer Workbench
- Restrictions on native XML store
- XML schema for the DADX file
- Syntax of the DADX file
- Dynamic query service operations in the Web services provider

C.1 New features of native XML data store in DB2 V9.1 for Linux, UNIX and Windows

You have already learned about the new native XML store in DB2 and SQL/XML functions common to both DB2 on z/OS and DB2 for Linux, UNIX and Windows in Appendix A, "XML and DB2" on page 565. Now we introduce other new features for native XML data store specific to DB2 V9.1 for Linux, UNIX and Windows:

- XML support in SQL statements and SQL/XML functions

  Many SQL statements support the new XML data type. This enables you to perform many common database operations with XML data, such as creating tables with XML columns, adding XML columns to existing tables, creating indexes over XML columns, creating triggers on tables with XML columns, and inserting, updating, or deleting XML documents.

- XQuery Support

  DB2 allows XQuery to be invoked directly, obtaining data by calling functions that extract XML data from DB2 tables and views. XQuery can also be combined with SQL.

- XML schema repository (XSR)

  The XML schema repository (XSR) is a repository for all XML artifacts required to validate and process XML instance documents stored in XML columns. It stores copies of XML schemas, DTDs, and external entities referenced in your XML documents.

  The XSR allows you to manage the dependencies XML documents have transparently, without requiring changes to the XML document content.

- Enhancements to the DB2 Command line processor (CLP) and command line tool

  Several DB2 commands have been updated or added to support the native storage of XML data. These updates allow you to work with XML data alongside relational data from the DB2 command line processor (CLP). Examples of tasks that you can perform on XML data from the CLP include:

  - Issuing XQuery statements by prefixing them with the XQUERY keyword
  - Importing and export XML data
  - Collecting statistics on XML columns
  - Calling stored procedures with IN, OUT, or INOUT parameters of XML data type
  - Working with the XML schemas, DTDs, and external entities required to validate and process XML documents
  - Reorganizing indexes over XML data and tables containing XML columns
  - Decomposing XML documents

- Import and export utility support for the native XML data store

  The import and export utilities have been updated to support the native XML data type. When exported, XML data is stored in a new auxiliary storage object, similar to the LOB storage object. Application development support for importing and exporting XML data is also provided by updated db2Import and db2Export APIs. These updated utilities permit data movement of XML documents stored in XML columns that is similar to the data movement support for relational data.

- db2batch command changes for native XML data store

  The db2batch command has been updated to process both SQL and XQuery statements. Users may issue XQuery statements by prefixing them with the XQUERY keyword.

- db2look command changes for native XML data store

The db2look command has been updated to allow you to reproduce the database objects required to validate and process XML documents. These include the XML schemas, DTDs, and external entities registered with the XML schema repository (XSR). The db2look command can export all the XSR objects required to validate and process XML documents, along with the DDL statements needed to register them at the target database.

- Explain and Visual Explain support for SQL/XML and XQuery statements

  The Explain facility and the Visual Explain GUI tool have been updated to support SQL enhancements for querying XML data and to support XQuery statements. These updates to the Explain facility and to the Visual Explain GUI tool allow you to see quickly how DB2 evaluates query statements against XML data. Several operators are provided to explain statements issued against XML data stored in XML columns. Query cost estimates are provided, along with optimizer output that shows how statements issued against XML data are evaluated, including optimizer use of indexes over XML data.

- Control Center support for native XML data store

  The Control Center has been updated to support the native XML data type for many of its administrative functions. This allows database administrators to work with XML data alongside relational data from within a single GUI tool. Examples of supported administrative tasks are:

  – Creating tables with XML columns

  – Creating indexes over XML columns using the new Create Index wizard

  – Viewing the contents of XML documents stored in XML columns

  – Working with the XML schemas, DTDs, and external entities required to validate and process XML documents

  – Collecting statistics on tables containing XML columns

- XML support in Developer Workbench

  In DB2 V9.1 for Linux, UNIX and Windows, Developer Workbench replaces Development Center in DB2 V8. The XQuery builder is also available to help create queries against XML data. The XQuery builder is part of the DB2 Developer Workbench. We will go into further detail in Appendix C.9, "Create and register an XML schema using Developer Workbench" on page 629.

- RUNSTATS command support for the native XML data store

  The RUNSTATS command has been updated to support the collection of statistics on tables containing XML columns and on indexes over XML data.These statistics are used by the optimizer to determine the optimal access path to XML data stored in XML columns. Up-to-date statistics are required for the most efficient access.

- Indexes over XML data

  Native XML data store provides support for indexing the XML data stored in XML columns. The use of indexes over XML data can improve the efficiency of queries issued against XML documents. Index entries in an index over XML data provide access to the XML documents stored in the rows of a table that satisfy specific XML patterns.

- Optimizer support for the native XML data store

  The optimizer has been updated to support the evaluation of SQL and XQuery statements against XML documents stored in XML columns. This includes the evaluation of indexes over XML data when modeling the execution cost of alternative access plans. By providing support for XML columns and for associated indexes over XML data, the optimizer can minimize the execution cost of working with XML documents.

C.2 Using the native XML data store in DB2 V9.1 for Linux, UNIX and Windows

You already learned about the new XML data type for DB2 in Appendix A.4, “The XML Data Type” on page 569. There is no architectural limit on the size of an XML value in a database. However, note that serialized XML data that is exchanged with a DB2 database is limited to 2 GB. XML documents can be inserted, updated and deleted using SQL data manipulation statements. Validation of an XML document against an XML schema, typically performed during insert or update, is supported by the XML schema repository (XSR). The DB2 database system also provides mechanisms for constructing and querying XML values, as well as exporting and importing XML data. An index over XML data can be defined on an XML column, providing improved search performance of XML data. The XML data in table or view columns can be retrieved as serialized string data through various application interfaces.

Restrictions:

1. XML data can only be stored in single-partition databases defined with the UTF-8 code set. Note that using XML features prevents future use of the Database Partitioning Feature available with DB2 Enterprise Server Edition for Linux, UNIX, and Windows.
2. You do not specify a length when you define an XML column. There is no architectural limit on the size of an XML value in a database. However, serialized XML data that is exchanged with a DB2 database is limited to 2 GB, so the effective limit of an XML column is 2 GB.

For further details, refer to Appendix C.10, “Restrictions on native XML store” on page 644.

Now we will show an example that walk you through how to create a DB2 database to store XML data and to perform basic operations with the native XML data store.

We Have issued all commands from the DB2 Command Line Processor (CLP) shown in the DB2 prompt of Figure C-1.

```
db2 =>
```

Figure C-1 DB2 CLP

1. Create a database `xmldb` that stores XML data.

   `CREATE DATABASE xmldb USING CODESET UTF-8 TERRITORY US`

2. You can at the DB CFG to verify that the codeset and territory.

Example: C-1 Database Configuration (DB CFG) for the xmldb database

```
get db cfg for xmldb

       Database Configuration for Database xmldb

 Database configuration release level                    = 0x0b00
 Database release level                                  = 0x0b00

 Database territory                                      = US
 Database code page                                      = 1208
 Database code set                                       = UTF-8
 Database country/region code                            = 1
```

```
...
```

3. Connect to the database.

   `CONNECT TO xmldb`

4. Now you can create a table named Customer that contains an XML column.

   `CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML)`

   The XML Schema Definition (XSD) for the `Info XML column` is shown in Example C-2.

Example: C-2 customer.xsd

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://podemo.org"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="customerinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="1" />
        <xs:element name="addr" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="street" type="xs:string" minOccurs="1" />
              <xs:element name="city" type="xs:string" minOccurs="1" />
              <xs:element name="prov-state" type="xs:string" minOccurs="1" />
              <xs:element name="pcode-zip" type="xs:string" minOccurs="1" />
            </xs:sequence>
            <xs:attribute name="country" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="phone" nillable="true" minOccurs="0"
        maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="type" form="unqualified" type="xs:string" />
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="assistant" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string" minOccurs="0" />
              <xs:element name="phone" nillable="true" minOccurs="0"
              maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent  >
                    <xs:extension base="xs:string">
                      <xs:attribute name="type" type="xs:string" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Cid" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure C-2 shows the hierarchical representation for the `Info` XML document. For simplicity, we did not show all element attributes.

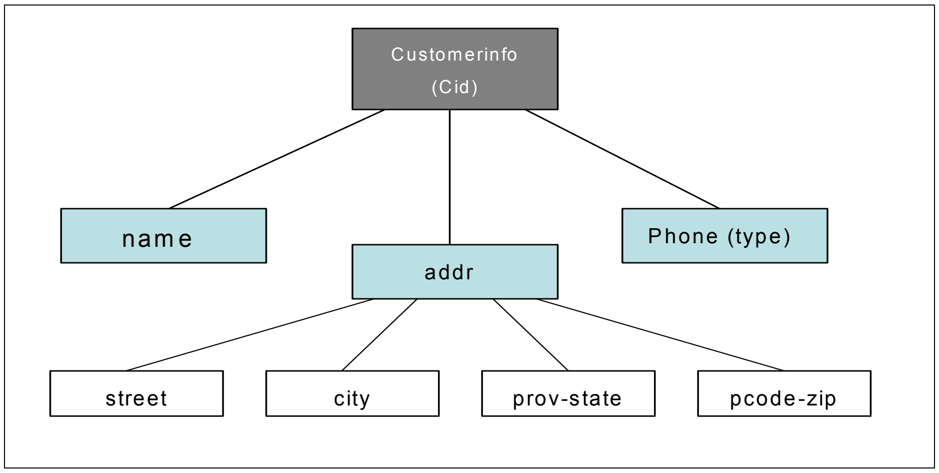


Figure C-2 Hierarchical structure of the customer info XML document

5. Creating an XML index that indexes the values of the `Cid` attribute of `<customerinfo>` elements from the Info column of the Customer table.

   `CREATE UNIQUE INDEX cust_cid_xmlidx ON Customer(Info) \`

   `GENERATE KEY USING XMLPATTERN '/customerinfo/@Cid' AS SQL DOUBLE`

6. Insert rows into the `Customer` table. Cut and paste all the lines in Example C-3 into the DB2 CLP.

Example: C-3 Insert statements to insert rows into Customer table

```
INSERT INTO Customer (Cid, Info) VALUES (1000, \
'<customerinfo Cid="1000"> \
  <name>Kathy Smith</name> \
  <addr country="Canada"> \
    <street>5 Rosewood</street> \
    <city>Toronto</city> \
    <prov-state>Ontario</prov-state> \
    <pcode-zip>M6W-1E6</pcode-zip> \
  </addr> \
  <phone type="work">416-555-1358</phone> \
</customerinfo>')

INSERT INTO Customer (Cid, Info) VALUES (1002, \
'<customerinfo Cid="1002"> \
  <name>Jim Noodle</name> \
  <addr country="Canada"> \
    <street>25 EastCreek</street> \
    <city>Markham</city> \
    <prov-state>Ontario</prov-state> \
    <pcode-zip>N9C-3T6</pcode-zip> \
  </addr> \
  <phone type="work">905-555-7258</phone> \
</customerinfo>')

INSERT INTO Customer (Cid, Info) VALUES (1003, \
'<customerinfo Cid="1003"> \
  <name>Robert Shoemaker</name> \
  <addr country="Canada"> \
    <street>1596 Baseline</street> \
    <city>Aurora</city> \
    <prov-state>Ontario</prov-state> \
    <pcode-zip>N8X-7F8</pcode-zip> \
  </addr> \
  <phone type="work">905-555-7258</phone> \
  <phone type="home">416-555-2937</phone> \
  <phone type="cell">905-555-8743</phone> \
  <phone type="cottage">613-555-3278</phone> \
</customerinfo>')
```

Typically, XML documents are inserted using application programs. While XML data can be inserted through applications using XML, binary, or character types, it is recommended that you use XML or binary types to avoid code page conversion issues. Above we show how to insert XML documents into XML typed columns manually in the DB2 CLP, where the XML document is always a character literal. In most cases, string data cannot be directly assigned to a target with an XML data type; the data must first be parsed explicitly using the XMLPARSE function. In INSERT, UPDATE, or DELETE operations, however, string data can be directly assigned to XML columns, without an explicit call to the XMLPARSE function. In these three cases, the string data is implicitly parsed.

7. Issue a SELECT statement to confirm data has been inserted successfully.

   `SELECT * from Customer`

   You will see three rows returned from the above query.

We continue using the example later in C.4, “SQL/XML examples” on page 601 to demonstrate how to use SQL/XML.

C.3 XML schema support

DB2 supports XML Schema validation of XML documents during insert, update, and query operations. XML schemas are valid XML documents that can be processed by tools such as the XSD Editor in DB2 Developer Workbench. You need to register your XML Schemas or DTD (limited support) before you can validate the XML document.

You use the **`REGISTER XMLSCHEMA`** command to registers an XML schema with the XML schema repository (XSR).

Example: C-4 Registering a XML schema. The schema document is PO.xsd

```
REGISTER XMLSCHEMA 'http://myPOschema/PO.xsd'
FROM 'file:///c:/TEMP/PO.xsd'
WITH 'file:///c:/TEMP/schemaProp.xml'
```

```
AS user1.POschema
```

The XMLVALIDATE function is used to validate XML documents. This function returns a copy of the input XML value augmented with information obtained from XML schema validation, including default values and type annotations.

Example C-5 shows how to do XML Schema validation during insert using the XML schema identified by the SQL name PODOCS.WORLDPO.

Example: C-5 XML Schema validation during insert

```
INSERT INTO T1(XMLCOL)
  VALUES (
    XMLVALIDATE(
      ? ACCORDING TO XMLSCHEMA ID PODOCS.WORLDPO
    )
 )
```

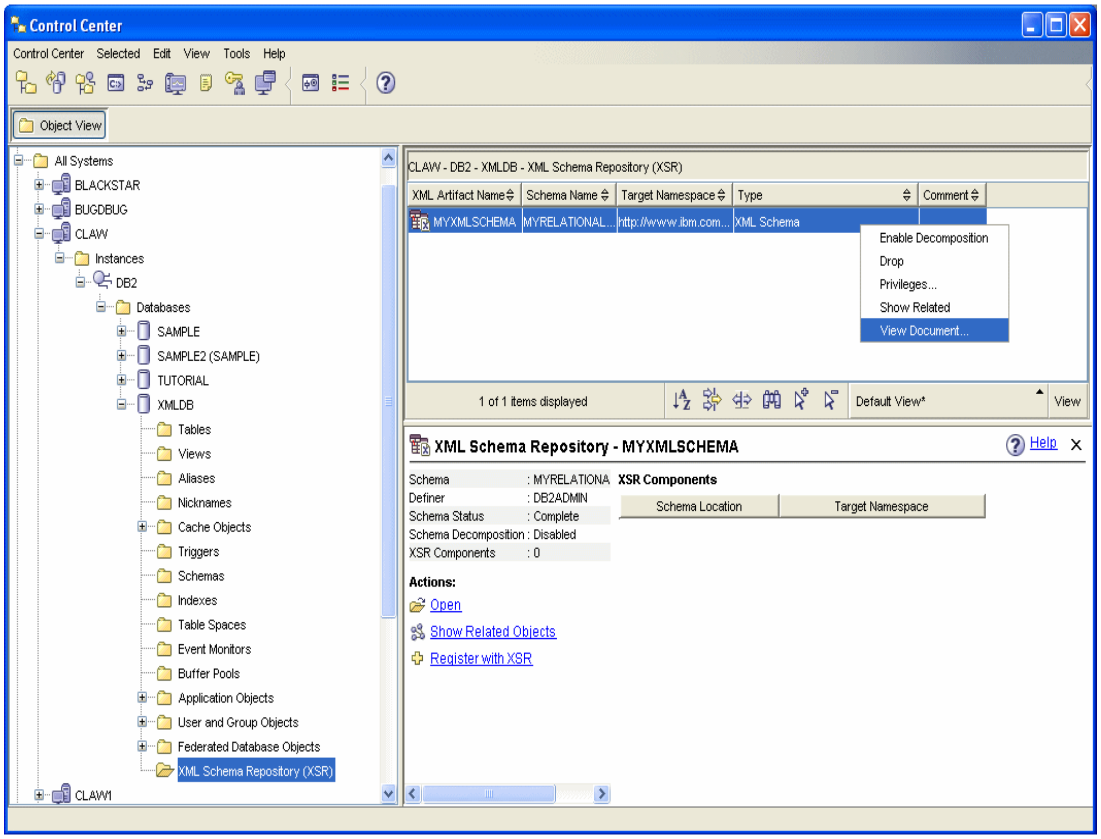
Assuming that the XML schema that is associated with SQL name FOO.WORLDPO is found in the XML repository, the input XML value will be validated and the type annotated according to that XML schema.

You can register your XML Schema and validate your XML document from the command line using the REGISTER XMLSCHEMA command, and the XMLVALIDATE function. You can also use Developer Workbench or Control Center to do the same. We will show an example of how to register an XML schema and perform validation using Developer Workbench in "Create and register an XML schema using Developer Workbench" on page 629.

C.3.1 XMl schema repository

The XML Schema Repository (XSR) is a repository for all XML artifacts required to validate and process XML instance documents stored in XML columns. It stores copies of XML schemas, DTDs, and external entities referenced in your XML documents.

The XSR allows you to manage the dependencies XML documents have transparently, without requiring changes to the XML document content. You can browse your XML Schema Document in Developer Workbench or in Control Center. Figure C-3 on page 601 shows the Control Center interface which allows you to register and view the XML schema document.



CLAW - DB2 - XMLDB - XML Schema Repository (XSR)

XML Artifact Name	Schema Name	Target Namespace	Type	Comment
MYXMLSCHEMA	MYRELATIONAL...	http://www.ibm.com...	XML Schema	

Schema	: MYRELATIONA
Definer	: DB2ADMIN
Schema Status	: Complete
Schema Decomposition	: Disabled
XSR Components	: 0

XSR Components

Schema Location	Target Namespace

Figure C-3 Control Center allows you to register and view XML Schema Document

C.4 SQL/XML examples

We have already introduced the SQL/XML functions in Appendix A.7, “SQL/XML” on page 571 when we introduced the common new XML functions available to both DB2 z/OS and DB2 for Linux, UNIX and Windows. Now we will show you some SQL/XML examples on DB2 for Linux, UNIX and Windows.

C.4.1 Update, Delete and Query using SQL/XML

For those who are familiar with SQL queries, we often restrict the rows returned from a query based on certain condition. If you want to restrict your search based on some condition that applies to data in an XML column, you can use the XMLEXISTS predicate to do this.

Continuing with the previous example we have used in Appendix C.2, “Using the native XML data store in DB2 V9.1 for Linux, UNIX and Windows” on page 596, we now show how to use some of the SQL/XML functions to update and query XML data. Our `Info` XML column currently contains the following information. (`Cid` column is omitted for simplicity and clarity.)

Example: C-6 Current XML contents in the Info column of the Customer table

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W-1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>

<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C-3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

1. We want to update the address for customer `Jim Noodle` since his address has changed. We show how we can use XMLEXISTS to specify that we only want to update customer Jim Noodle's information who has a customer ID of 1002. Cut and paste all the lines in Example C-7 into the DB2 CLP.

Example: C-7 Update customer Jim Noodle's address

```
UPDATE customer SET info = \
'<customerinfo Cid="1002"> \
  <name>Jim Noodle</name> \
  <addr country="Canada"> \
    <street>1150 Maple Drive</street> \
    <city>Newtown</city> \
    <prov-state>Ontario</prov-state> \
    <pcode-zip>Z9Z 2P2</pcode-zip> \
  </addr> \
  <phone type="work">905-555-7258</phone> \
</customerinfo>' \
WHERE XMLEXISTS ('$doc/customerinfo[@Cid = "1002"]' passing INFO as "doc")
```

2. Issue a SELECT query again to confirm Jim Noodle's address is updated.

   `SELECT * from Customer WHERE Cid=1002`

3. Next we show in Example C-8 on page 603 how to retrieve the name of the customer who has a customer ID of 1000 using XMLQUERY.

Example: C-8 Using XMLQUERY

```
SELECT XMLQUERY('$c/customerinfo/name' passing Info as "c") FROM Customer \
WHERE Cid = 1000
```

You get the result shown in Example C-9.

Example: C-9 Result for query in Example C-8

```
<name>Kathy Smith</name>
```

4. Query in Example C-10 retrieves the ID for the customer who lives in the city of Toronto.

Example: C-10 Using XMLEXISTS

```
SELECT Cid FROM Customer WHERE XMLEXISTS \
('$c/customerinfo/addr[city = "Toronto"]' passing Customer.Info as "c")
```

You get the result of Example C-11.

Example: C-11 Results for query in Example C-10

```
CID
--------------------
                1000

  1 record(s) selected.
```

Important: One very common *mistake* is to issue the statement shown in Example C-12 instead of the one in Example C-10.

Example: C-12 Incorrect statement that uses XMLEXISTS

```
SELECT Cid FROM Customer WHERE XMLEXISTS \
('$c/customerinfo/addr/city = "Toronto"' passing Customer.Info as "c")

CID
--------------------
                1000
                1002
                1003

  3 record(s) selected.
```

Notice that you get all the Cid(s) in the Customer table instead of only the Cid of the customer who lives in the city of Toronto. This query runs successfully, but it does *not* give you the results you want. This is because the semantics of the query in Example C-12 is incorrect. Remember our Customer entry for Cid = 1000 (see Example C-13).

Example: C-13 The value of the Info XML column in the Customer table for CId = 1000

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W-1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>
```

If XMLEXISTS includes an XPath expression with a value predicate `(expression)`, you want to enclose the predicate in square brackets, that is `[expression]`. When the entire expression is placed in square brackets, the meaning is 'fetch the document if [expression]', and you should always get an empty sequence if a document does not satisfy expression. Since the value comparison is always within square brackets, Example C-10 on page 603 works properly, but Example C-12 on page 603 fails to give us the result we expect. The rule also applies for queries where there is no value comparison. For example, when you want to return the documents for all EMPLOYEE which happen to have a COMMENT child element, then you will use:

```
SELECT EMPID FROM EMPLOYEE WHERE XMLEXISTS('$e[EMPLOYEE/COMMENT ]' PASSING
EMPLOYEE.xmlcol AS "e")
```

5. If we want to delete customer with customer ID 1003 we can issue the following command shown in Example C-14. **(We will NOT delete customer with ID 1003 at this time, therefore we will not issue the command below. This is so that we can continue to show the next few examples)**

Example: C-14 Delete customer with Cid=1003

```
DELETE FROM Customer \
WHERE XMLEXISTS ('$doc/customerinfo[@Cid = "1003"]' passing INFO as "doc")
```

6. Now we show how to make use of the **`XMLTABLE`** function to get a table of results from the XML information in the `Customer` table `Info` column. We want the phone information to appear in a row for each individual customer. So we can issue the command shown in Example C-15.

Example: C-15 Using XMLTABLE to return a table listing of customers phone numbers

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/phone' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(30) PATH '../name',
                           "PHONETYPE" CHAR(30) PATH '@type',
                           "PHONENUM" CHAR(15) PATH '.'
                          ) as X;
```

And the result is shown in Figure C-4.

```
CUSTNAME                       PHONETYPE                      PHONENUM
------------------------------ ------------------------------ ---------------
Kathy Smith                    work                           416-555-1358
Jim Noodle                     work                           905-555-7258
Robert Shoemaker               work                           905-555-7258
Robert Shoemaker               home                           416-555-2937
Robert Shoemaker               cell                           905-555-8743
Robert Shoemaker               cottage                        613-555-3278

  6 record(s) selected.
```

Figure C-4 Results for query in Example C-15

7. Another example, say we want to display each of the address field in its own column and generate a tabular output, then we can issue:

Example: C-16 Using XMLTABLE to display address

```
SELECT X.*
FROM CUSTOMER C, XMLTABLE ('$cust/customerinfo/addr' PASSING C.INFO as "cust"
                           COLUMNS "CUSTNAME" CHAR(18) PATH '../name',
                           "COUNTRY" CHAR(8) PATH '@country',
                           "STREET" CHAR(16) PATH './street',
                           "CITY" CHAR(10) PATH './city',
                           "PROVINCE" CHAR(10) PATH './prov-state',
                           "POSTAL CODE" CHAR(7) PATH './pcode-zip'
                          ) as X
```

And you will get the following tabular output as shown in Figure C-5.

```
CUSTNAME           COUNTRY  STREET           CITY       PROVINCE   POSTAL CODE
------------------ -------- ---------------- ---------- ---------- -----------
Kathy Smith        Canada   5 Rosewood       Toronto    Ontario    M6W-1E6
Jim Noodle         Canada   1150 Maple Drive Newtown    Ontario    Z9Z 2P2
Robert Shoemaker   Canada   1596 Baseline    Aurora     Ontario    N8X-7F8

  3 record(s) selected.
```

Figure C-5 Results for query in Example C-16

Tip: The XMLTABLE function is very useful when you want to present your native XML data in a relational table output.

C.4.2 Publishing XML as relational data

We have already learned now to query, extract, and transform data in an XML column just now. In this section we will show how to use SQL/XML functions to construct and publish XML data from relational data. At this time, we will start referencing the `EMPLOYEE` table in the DB2 `SAMPLE` database.

Note: SAMPLE database needs to be created in unicode or converted to unicode. You can follow the steps below to drop the existing non-unicode SAMPLE database and recreate a new SAMPLE unicode database

1. Create a temporary export directory where you will store the export files. For example, we call our directory /exportdata. Go to the /exportdata directory and export all the data from the existing SAMPLE database.

   ```
   cd exportdata
   ```

   ```
   db2move sample export
   ```

2. Generate a DDL script for your existing database using the db2look command: **db2look -d sample -e -o unisampl.ddl -l -x -f**

   where

   – sample is the existing database name
   – unisampl.ddl is the file name for the generated DDL script
   – The **-l** option generates DDL for user defined table spaces, database partition groups and buffer pools
   – The **-x** option generates authorization DDL
   – The **-f** option generates update command for database configuration parameters

3. Drop the existing SAMPLE database

   ```
   DROP DATABASE SAMPLE
   ```

4. Create new the SAMPLE database in Unicode:

   ```
   CREATE DATABASE SAMPLE USING CODESET UTF-8 TERRITORY US
   ```

5. Recreate your database structure by running the DDL script

   ```
   db2 -tvf unisampl.ddl
   ```

6. Import your data into the new Unicode database using the **db2move** command:

   ```
   cd exportdata
   ```

   ```
   db2move sample import
   ```

7. The STAFF table contains the following relational data as shown in Figure C-6. (Not all the rows are shown for simplicity.)

```
SELECT * FROM STAFF

ID     NAME      DEPT   JOB   YEARS  SALARY    COMM
------ --------- ------ ----- ------ --------- ---------
    10 Sanders       20 Mgr        7  18357.50         -
    20 Pernal        20 Sales      8  18171.25    612.45
    30 Marenghi      38 Mgr        5  17506.75         -
    40 O'Brien       38 Sales      6  18006.00    846.55
    50 Hanes         15 Mgr       10  20659.80         -
    60 Quigley       38 Sales      -  16808.30    650.25
    70 Rothman       15 Sales      7  16502.83   1152.00

...

   310 Graham        66 Sales     13  21000.00    200.30
   320 Gonzales      66 Sales      4  16858.20    844.00
   330 Burke         66 Clerk      1  10988.00     55.50
   340 Edwards       84 Sales      7  17844.00   1285.00
   350 Gafney        84 Clerk      5  13030.50    188.00

  35 record(s) selected.
```

Figure C-6 Staff table in DB2 SAMPLE database

We can easily construct XML data from the SQL columns by using the **XMLELEMENT** function. The **XMLELEMENT** scalar function returns an XML value that is an XML element node.

Example: C-17 Constructing XML data from SQL column

```
SELECT XMLELEMENT (name "employee",
                   XMLELEMENT (name "ID", id),
                   XMLELEMENT (name "NAME", name),
                   XMLELEMENT (name "JOB", job),
                   XMLELEMENT (name "SALARY", salary))
FROM STAFF
```

Example C-17 on page 607 generates the following XML data (extra formatting and white spaces added to enhance readability).

```
1
--------------------------------------------------------------------------------
<employee>
   <id>10</id>
   <name>Sanders</name>
   <job>Mgr  </job>
   <salary>18357.50</salary>
</employee>
<employee>
   <id>20</id>
   <name>Pernal</name>
   <job>Sales</job>
   <salary>18171.25</salary>
</employee>

...

<employee>
   <id>340</id>
   <name>Edwards</name>
   <job>Sales</job>
   <salary>17844.00</salary>
</employee>
<employee>
   <id>350</id>
   <name>Gafney</name>
   <job>Clerk</job>
   <salary>13030.50</salary>
</employee>

  35 record(s) selected.
```

Figure C-7 XML data generated from the SQL columns in Staff table

8. You can also use XMLFOREST to generate the same XML data as shown in Figure C-7 above. The XMLFOREST function produces a forest of XML elements that all share a specific pattern from a list of columns and expressions.

Example: C-18 Using XMLFOREST and XMLELEMENT together

```
SELECT XMLELEMENT (name "employee",
                   XMLFOREST (id as "id",
                              name as "name",
                              job as "job",
                              salary as "salary"))
FROM STAFF
```

This will also generate the same XML output as shown in Figure C-7.

9. The XMLAGG aggregate function returns an XML sequence containing an item for each non-null value in a set of XML values. In this example we show how to generate a list of employee information in XML format sorted by their salary in ascending order. The query is shown in Example C-19.

Example: C-19 Using XMLAGG and ORDER BY

```
SELECT XMLAGG (XMLELEMENT (name "employee",
               XMLELEMENT (name "id", id),
               XMLELEMENT (name "name", name),
               XMLELEMENT (name "job", job),
               XMLELEMENT (name "salary", salary))
       ORDER BY SALARY )
FROM STAFF
```

The result is shown in Figure C-8 on page 610.

```
1
--------------------------------------------------------------------------------
<employee>
   <id>130</id>
   <name>Yamaguchi</name>
   <job>Clerk</job>
   <salary>10505.90</salary>
</employee>
<employee>
   <id>330</id>
   <name>Burke</name>
   <job>Clerk</job>
   <salary>10988.00</salary>
</employee>
<employee>
   <id>200</id>
   <name>Scoutten</name>
   <job>Clerk</job>
   <salary>11508.60</salary>
</employee>

...

<employee>
   <id>140</id>
   <name>Fraye</name>
   <job>Mgr  </job>
   <salary>21150.00</salary>
</employee>
<employee>
   <id>260</id>
   <name>Jones</name>
   <job>Mgr  </job>
   <salary>21234.00</salary>
</employee>
<employee>
   <id>160</id>
   <name>Molinare</name>
   <job>Mgr  </job>
   <salary>22959.20</salary>
</employee>

  1 record(s) selected.
```

Figure C-8 Output for the XMLAGG query

C.5 XQuery in DB2 V9.1 for Linux, UNIX and Windows

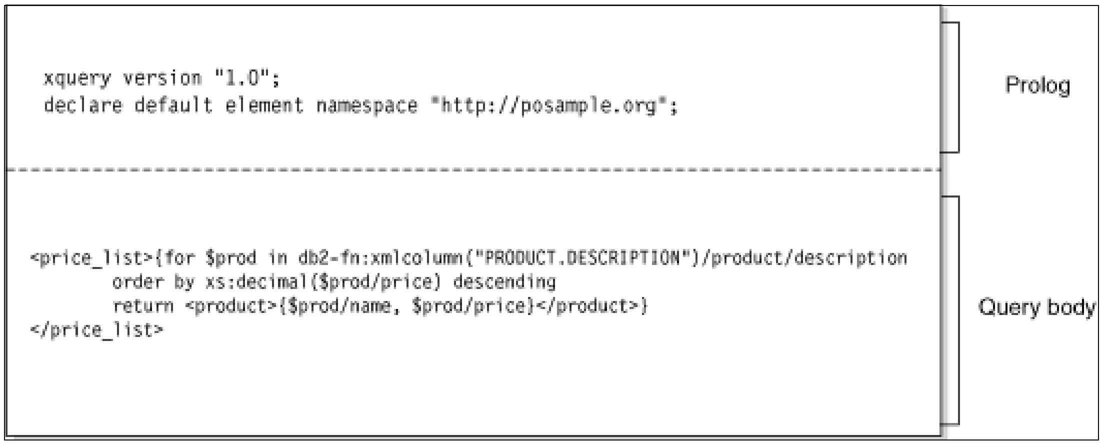
XQuery is a generalized language for querying XML data. DB2 treats XQuery as a first-class language, which allows XQuery to be invoked directly. This means that DB2 engine processes XQueries natively, and parses XQueries without translating them into SQL. XQuery can also be invoked from an SQL query. In this case, the SQL query can pass XML data to XQuery in the form of bound variables. XQuery supports various expressions for processing XML data and for constructing new XML objects such as elements and attributes. The programming interface to XQuery provides facilities similar to those of SQL to execute queries and retrieve query results.

XQuery is a functional programming language that was designed by the World Wide Web Consortium (W3C) to meet specific requirements for querying XML data. Unlike relational data, which is predictable and has a regular structure, XML data is highly variable. Because the structure of XML data is unpredictable, the queries that you need to perform on XML data often differ from typical relational queries. The XQuery language provides the flexibility required to perform these kinds of operations. For example, you might need to create XML queries that search XML data for objects that are at unknown levels of the hierarchy or that perform structural transformations on the data and return results that have mixed types. XQuery is a strongly-typed language in which the operands of various expressions, operators, and functions must conform to expected types. The type system for XQuery is based on XML Schema.

In XQuery, expressions are the basic building blocks of a query. Expressions can be nested and form the body of a query. A query consists of an optional prolog that is followed by a query body. The prolog contains a series of declarations that define the processing environment for the query. The query body consists of an expression that defines the result of the query. Expressions can be used alone or in combination with other expressions to form complex queries. DB2 supports several kinds of expressions for working with XML data, including path expressions for locating nodes within a document tree, constructors for creating XML structures within a query, and FLWOR expressions for iteration and for binding of variables to intermediate query results.

Note: FLWOR expression stands for - **F**or, **L**et, **W**here, **O**rder, **R**eturn expressions. FLWOR expressions will be explained further in Appendix C.5.1, “FLWOR expressions” on page 612. Path expression is discussed further in Appendix C.5.2, “Path expressions” on page 614.

Figure C-9 illustrates the structure of a typical query. In this example, the prolog contains two declarations: a version declaration, which specifies the version of the XQuery syntax to use to process the query, and a default namespace declaration that specifies the namespace URI to use for unprefixed element and type names. The version declaration, if present, must be first in the prolog. The query body contains an expression that constructs a price_list element. The content of the price_list element is a list of product elements that are sorted in descending order by price.



```
xquery version "1.0";
declare default element namespace "http://posample.org";
```

```
<price_list>{for $prod in db2-fn:xmlcolumn("PRODUCT.DESCRIPTION")/product/description
       order by xs:decimal($prod/price) descending
       return <product>{$prod/name, $prod/price}</product>}
</price_list>
```

Figure C-9 XQuery structure showing the optional prolog followed by a query body

XQuery uses the XQuery and XPath data model (XDM), which represents an XML document as a hierarchy (tree) of nodes that represent XML elements and attributes. The XDM allows XQuery to operate on the abstract, logical structure of an XML document or fragment, rather than its surface syntax. The inputs (if any) of an XQuery expression are instances of the XDM, and the result of an expression is also an instance of the XDM. XML documents are converted into the XDM when they are stored in an XML column.

DB2 supports XQuery built-in functions for working with XML data. The library includes the following types of functions: string functions, numeric functions, functions that operate on boolean values, functions that operate on QNames, functions that operate on nodes, functions on sequences, and functions that operate on durations, dates, and times.

A query that invokes XQuery directly begins with the keyword **`XQUERY`**. This keyword indicates that XQuery is being used and that the DB2 server must therefore use case sensitivity rules that apply to the XQuery language. After establishing the processing environment for the query, the query must retrieve input data. DB2 provides the following functions to retrieve input data from an XML column: **`db2-fn:xmlcolumn`** and **`db2-fn:sqlquery`**.

Important: SQL is *not* a case-sensitive language, but XQuery *is* a case-sensitive language.

SQL and XQuery have different conventions for case-sensitivity of names. You should be aware of these differences when using the **`db2-fn:sqlquery`** and **`db2-fn:xmlcolumn`** functions. SQL is not a case-sensitive language. By default, all ordinary identifiers, which are used in SQL statements, are automatically converted to uppercase. Therefore, the names of SQL tables and columns are customarily uppercase names. In an SQL statement, these columns can be referenced by using lowercase names, which are automatically converted to uppercase during processing of the SQL statement. (You can also create a case-sensitive name that is called a delimited identifier in SQL by enclosing the name in double quotation marks.)

XQuery is a case-sensitive language. XQuery does not convert lowercase names to uppercase. This difference can lead to some confusion when XQuery and SQL are used together. The string that is passed to **`db2-fn:sqlquery`** is interpreted as an SQL query and is parsed by the SQL parser, which converts all names to uppercase. The operand of **`db2-fn:xmlcolumn`**, however, is not an SQL query. The operand is a case-sensitive XQuery string literal that represents the name of a column.

C.5.1 FLWOR expressions

FLWOR expressions iterate over sequences and bind variables to intermediate results. These expressions are useful for computing joins between two or more documents, restructuring data, and sorting the result. A FLWOR expression is composed of the following clauses which it gets its name from: **F**or, **L**et, **W**here, **O**rder by, and **R**eturn. Figure C-10 on page 613 shows the FLWOR expressions syntax diagram.

```
    .--------------------.
    V                    |
>>---+-| for clause |-+-+--+-------------------+----------------->
     '-| let clause |-'    '-where--Expression-'

>--+---------------------------------------+------------------>
   |           .-,------------------------. |
   |           V            .-ascending--. | |
   '-order by----Expression--+------------+-+-'
                             '-descending-'

>--return--Expression------------------------------------------><

for clause

        .-,------------------------------------------------------------.
        V                                                              |
|--for----$VariableName--+----------------------------+--in--Expression-+--|
                         '-at--$PositionalVariableName-'

let clause

        .-,-----------------------------.
        V                               |
|--let----$VariableName--:=--Expression-+-----------------------|
```

Figure C-10 FLWOR expressions syntax diagram

Table C-1 explains each of the clause in the FLWOR expression.

Table C-1 FLWOR expression clauses

Clause	Explanation
for	The keyword that begins a for clause. A for clause iterates over the result of *Expression* and binds *VariableName* to each item that is returned by *Expression*.
let	The keyword that begins a let clause. A let clause binds *VariableName* to the entire result of *Expression*. ***VariableName*** The name of the variable to bind to the result of *Expression*. ***PositionalVariableName*** The name of an optional variable that is bound to the position within the input stream of the item that is bound by each iteration of the for clause. ***Expression*** Any XQuery expression. If the expression includes a top-level comma operator, then the expression must be enclosed in parentheses.
where	The keyword that begins a where clause. A where clause filters the tuples of variable bindings that are generated by the for and let clauses.

Clause	Explanation
order by	The keywords that begin an order by clause. An order by clause specifies the order in which values are processed by the return clause. **ascending** Specifies that ordering keys are processed in ascending order. **descending** Specifies that ordering keys are processed in descending order.
return	The keyword that begins a return clause. The expression in the return clause is evaluated once for each tuple of bound variables that is generated by the for, let, where, and order by clauses. The results of all of the evaluations of the return clause are concatenated into a single sequence, which is the result of the FLWOR expression.

XML data can be queried using SQL (with the SELECT statement), XQuery (with XQuery expressions), or a combination of both. When querying with SQL alone (without the use of any XQuery), you can only query at the column level. That is, you can return the entire XML document stored in the column, but you cannot query within the document or return fragments of a document. To query values within an XML document or return fragments of a document, you must use XQuery.

C.5.2 Path expressions

Path expressions identify nodes within an XML tree. Path expressions in DB2 XQuery are based on the syntax of XPath 2.0.

A path expression consists of one or more steps that are separated by slash (/) or double-slash (//) characters. A path expression can begin with a step or with a slash or double-slash character. Each step before the final step generates a sequence of nodes that are used as context nodes for the step that follows.

The first step specifies the starting point of the path, often by using a function call or variable reference that returns a node or sequence of nodes. An initial "/" indicates that the path begins at the root node of the tree that contains the context node. An initial "//" indicates that the path begins with an initial node sequence that consists of the root node of the tree that contains the context node, plus all of the descendants of the root node.

Each step is executed repeatedly, once for each context node that is generated by the previous step. The results of these repeated executions are then combined to form the sequence of context nodes for the step that follows. The value of the path expression is the combined sequence of items that results from the final step in the path. This value can be either a sequence of nodes or a sequence of atomic values. Because each step in a path provides context nodes for the step that follows, the final step in a path is the only step that can return a sequence of atomic values which satisfied the specified condition.

Figure C-11 on page 615 shows the syntax of path expressions.

```
           .-/ or //-------------------.
           V                           |
>>-+----+----+-| axis step |---------+-+-----------------------><
   +-/--+    '-| filter expression |-'
   '-//-'

axis step

                            .---------------------------.
                            V                           |
|----+--------+--node-test----+----------------------+-+-------|
     '-axis::-'                '-[PredicateExpression]-'

filter expression

                      .---------------------------.
                      V                           |
|--PrimaryExpression----+----------------------+-+-------------|
                        '-[PredicateExpression]-'
```

Figure C-11 Syntax of path expressions

Each step of a path expression is either an axis step or a filter expression. An axis step returns a sequence of nodes that are reachable from the context node via a specified axis. A filter expression consists of a primary expression that is followed by zero or more predicates.

An axis step consists of three parts: an optional axis, which specifies a direction of movement; a node test, which specifies the criteria that is used to select nodes; and zero or more predicates, which filter the sequence that is returned by the step. The result of an axis step is always a sequence of zero or more nodes.

XQuery provides an abbreviated syntax for expressing axes in path expressions. For Abbreviated syntax for path expressions, you can refer to DB2 V9.1 for Linux, UNIX and Windows Information Center at this URL:

`http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.xquery.doc/xqrabbrsyn.html`

In most cases, you can write a query using either a FLWOR expression or a path expression to get the results you want. Table C-2 provides a full listing of various DB2 XQuery expressions.

Table C-2 DB2 XQuery expressions

Type	**Explanation**
Primary expressions	Primary expressions are the basic primitives of the language. They include literals, variable references, parenthesized expressions, context item expressions, constructors, and function calls.
Path expressions	*Path expressions* identify nodes within an XML tree. Path expressions in DB2 XQuery are based on the syntax of XPath 2.0.

Predicates	A *predicate* filters a sequence by retaining the qualifying items. A predicate consists of an expression, called a predicate expression, that is enclosed in square brackets ([]). The predicate expression is evaluated once for each item in the sequence, with the selected item as the context item. Each evaluation of the predicate expression returns an xs:boolean value called the *predicate truth value*. Those items for which the predicate truth value is true are retained, and those for which the predicate truth value is false are discarded.
Sequence expressions	Sequence expressions construct, filter, and combine sequences of items. Sequences are never nested. For example, combining the values 1, (2, 3), and ( ) into a single sequence results in the sequence (1, 2, 3).
Arithmetic expressions	Arithmetic expressions perform operations that involve addition, subtraction, multiplication, division, and modulus.
Comparison expressions	Comparison expressions compare two values. XQuery provides three kinds of comparison expressions: value comparisons, general comparisons, and node comparisons.
Logical expressions	Logical expressions use the operators and or to compute a Boolean value (true or false).
Constructors	Constructors create XML structures within a query. XQuery provides constructors for creating element nodes, attribute nodes, document nodes, text nodes, processing instruction nodes, and comment nodes. XQuery provides two kinds of constructors: direct constructors and computed constructors.
FLWOR expressions	FLWOR expressions iterate over sequences and bind variables to intermediate results. FLWOR expressions are useful for computing joins between two or more documents, restructuring data, and sorting the result.
Conditional expressions	Conditional expressions use the keywords if, then, and else to evaluate one of two expressions based on whether the value of a test expression is true or false.
Quantified expressions	Quantified expressions return true if some or every item in one or more sequences satisfies a specific condition. The value of a quantified expression is always true or false.
Cast expressions	A cast expression creates a new value of a specific type based on an existing value.

C.5.3 XQuery examples for DB2 for Linux, UNIX and Windows

We now look at some XQuery examples to show you what can be done with XQuery. These are very simple examples to get you familiar with XQuery. Again we are using the same example we used earlier in Appendix C.2, “Using the native XML data store in DB2 V9.1 for Linux, UNIX and Windows” on page 596 and Appendix C.4, “SQL/XML examples” on page 601. Remember our `Customer` table is defined as follow:

```
CREATE TABLE Customer (Cid BIGINT NOT NULL PRIMARY KEY, Info XML)
```

Our `Customer` table currently looks like Example C-20. For simplicity we have omitted the `Cid` column and only show the `Info` column that contains XML data.

Example: C-20 Current content of the Info XML column in the Customer table

```
<customerinfo Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W-1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>

<customerinfo Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C-3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>

<customerinfo Cid="1003">
  <name>Robert Shoemaker</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Aurora</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N8X-7F8</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
  <phone type="home">416-555-2937</phone>
  <phone type="cell">905-555-8743</phone>
  <phone type="cottage">613-555-3278</phone>
</customerinfo>
```

Important: XQuery is case-sensitive, while SQL is case-insensitive. Names in XQuery, such as table and SQL schema names (which are both uppercase by default), must be carefully specified because of the language's case-sensitivity. This is particularly important when using XQuery with SQL. When invoking XQuery within SQL, be mindful that the XQuery expression remains case-sensitive, even though it is placed within the SQL context.

We have already shown how to retrieve data using only SQL:

```
SELECT * from Customer
```

Invoking the above SQL will retrieve the entire XML document stored in the column named `Info` and values from the `Cid` primary key column.

From within the contexts of both SQL and XQuery, you can invoke the other. In SQL, you can invoke XQuery using the **`XMLQUERY`** function. The **`XMLQUERY`** function enables you to invoke XQuery from the SQL context. In XQuery, you can issue a fullselect using the **`db2-fn:sqlquery`** function. To get the sequence of XML documents that is stored in the given column, use the **`db2-fn:xmlcolumn`** function.

1. For example, the following query shown in Example C-21 on page 618 returns the XML stored in the `Info` column of our `Customer` table as shown in Example C-20 on page 616.

Example: C-21 Retrieve XML from the Info column

```
xquery db2-fn:xmlcolumn ('CUSTOMER.INFO')
```

Important: Specify the column and table names in upper case, otherwise you may see a SQL0204N error indicating that database object is undefined. This is because table and column names are stored in DB2's catalog in upper case. XQuery is case-sensitive so if you use lower case table and column names it will not find the table and column names in the DB2 catalog.

2. This query only retrieves the names for all customers using path expressions (see Example C-22).

Example: C-22 Path expression to retrieve names for all customers

```
xquery  \
db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo/name
```

Figure C-12 shows the result:

```
<name>Kathy Smith</name>
<name>Jim Noodle</name>
<name>Robert Shoemaker</name>
```

Figure C-12 All names of our customers

3. This query returns the same result as in Example C-22, but this time we use the FLWOR expression and utilizes the **`for`** and **`return`** clause. We iterate the `<name>` elements of `<customerinfo>` in the CUSTOMER.INFO XML column. Each name is bound to the variable $n and the value is returned in $n for each iteration. See Example C-23.

Example: C-23 FLWOR expression to retrieve names for all customer

```
xquery  \
for $n in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo/name \
return $n
```

4. But if you only want the text string for the customer's name stored in the `Info` XML column and do not care about the XML tag, use the text() function. See Example C-24.

Example: C-24 Using the text() function to retrieve customer's name

```
xquery  \
db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo/name/text()
```

Then you will get the text:

```
Kathy Smith
Jim Noodle
Robert Shoemaker
```

5. The following illustrates the additional use of a **`where`** clause in the FLWOR expression to get the address for customers who are in the city of Toronto only. The **`FOR`** clause specifies iteration through the `<customerinfo>` elements of documents in the `Info` column. The **`WHERE`** clause filters to yield only items that have a `<city>` element (along the path specified) with a value of `"Toronto"`. Finally the **`RETURN`** clause constructs the returned XML value, which is an element that contains the `<addr>` element for all documents that satisfy the condition specified in the **`WHERE`** clause. See Example C-25.

Example: C-25 Adding the where clause to filter the results in a FLWOR expression

```
xquery \
for $n in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo/addr \
where $n/city ="Toronto" \
return $n
```

And we get the fragment of the XML document shown in Figure C-13.

```
 <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W-1E6</pcode-zip>
  </addr>
```

Figure C-13 Resulting XML fragment for the query in Example C-25

6. You can also get the same result using the path expressions of Example C-26.

Example: C-26 Path expressions equivalent to the FLWOR expression in Example C-25

```
xquery \
db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo/addr[city ="Toronto"]
```

7. Now we illustrate how to build and transform existing phone information into a phone list sort by customer's name.

Example: C-27 Transforming the XML output

```
xquery \
for $n in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo \
order by $n/name \
return <phonelist> {$n/phone} </phonelist>
```

See Figure C-14.

```
<phonelist>
   <phone type="work">905-555-7258</phone>
</phonelist>

<phonelist>
   <phone type="work">416-555-1358</phone>
</phonelist>

<phonelist>
   <phone type="work">905-555-7258</phone>
   <phone type="home">416-555-2937</phone>
   <phone type="cell">905-555-8743</phone>
   <phone type="cottage">613-555-3278</phone>
</phonelist>
```

Figure C-14 Transforming into a phone list sorted by customer's first name

For more information about XML, SQL/XML, and XQuery, refer to the XML Guide, available at:

ftp://ftp.software.ibm.com/software/data/pubs/papers/db2xmlguide.pdf

C.6 Comparison of XML data access methods

As you have learned in the previous sections, XML data can be queried in a number of ways: using plain SQL only, using SQL/XML functions, using XQuery or a combination of all of these methods. So which method should you use so that it will be most advantageous for your particular situation. Let us explore our options:

- SQL only

  When retrieving XML data using SQL alone, you can only query at the XML column level. Therefore, you can only retrieve the entire XML documents from the query. This usage is suitable if you want to retrieve entire XML documents. It is also suitable in the case where you do not need to query based on values within the stored XML documents, and the predicates of your query are on other non-XML columns of the table. SQL also works well for SQL programmers or DBAs who are already very familiar with the language.

- XQuery only

  If you only have to access XML data, and do not need to access any relational data, then XQuery alone will do the work. XQuery is also suitable when you need to use the query result to construct other XML documents. XQuery might also be the preference for those who are familiar with XML and XQuery, but do not use SQL very often.

- XQuery that invokes SQL

  Combining XQuery and SQL will be a suitable choice when you need to access both XML and relational data. This way you cannot only access your XML data, but also leverage SQL predicates and indexes on relational columns.

- SQL/XML functions that execute XQuery expressions

  The SQL/XML functions like XMLQUERY, XMLTABLE, and the XMLEXISTS predicate, enable XQuery expressions to be executed from within the SQL context. This will allow you to access both relational and XML data in a single query. This also means that you join XML with relational data, and further, you can aggregate the data by using the GROUP BY and ORDER BY SQL clauses. Combining these methods will be a suitable choice when you want to enable existing SQL applications to query within XML documents

Table C-3 on page 621 summarizes when it is most advantageous to use a particular query method:

Table C-3 Comparison of XML data access methods

SQL only	- Good choice when you are very familiar with SQL - Can only retrieve entire XML document but not parts of it - Predicates of your query are on relational columns of the table
XQuery only	- Good choice when you are very familiar with XML and XQuery - Can only access XML data - You can use query result to construct new XML documents
XQuery and SQL	- Can access both XML and relational data - Allows you to leverage SQL predicates and indexes on relational column
SQL/XML and XQuery	- Allows you to access both relational and XML data in a single query- - Allows you to join XML with relational data, and aggregate the data using the GROUP BY and ORDER BY SQL clauses - Good choice when you want to enable existing SQL applications to query within XML documents

C.7 Annotated XML schema decomposition

DB2 native XML store allows you to store and access XML data natively, but there may be cases where accessing XML data as relational data is required. An example of such a requirement is an existing application that expects and treats XML in a relational form. For such cases, annotated XML schema decomposition can be used to store content from XML documents in columns of relational tables.

Annotated XML schema decomposition is a new feature that decomposes documents based on annotations specified in an XML schema. The annotations added to XML schema documents specify details such as the name of the target table and column the XML data is to be stored in, the default SQL schema for when a target table's SQL schema is not identified, as well as any transformation of the content before it is stored.

C.7.1 XML Extender shredding versus annotated XML schema decomposition

You might ask why we want to use annotated XML Schema Decomposition when current functionalities are also offered by DB2 XML Extender. While DB2 XML Extender also supports the ability to shred documents into relational schema, it only has very limited functionality. The XML Extender decomposition stored procedures dxxInsertXML() and dxxShredXML() are used to break down or shred incoming XML documents and to store data in new or existing database tables. The dxxInsertXML() stored procedure takes an enabled XML collection name as input; while the dxxShredXML() stored procedure takes a DAD file as input. Therefore, XML Extender is constrained by the proprietary DAD mapping format.

The annotated XML schema decomposition in DB2 V9.1 for Linux, UNIX and Windows is faster and more efficient than the DB2 XML Extender. Further, Annotated XML schema decomposition provides an XML schema-based flexible mapping language that provides granular control to the users over the entire process of decomposition.

C.7.2 DB2 V9.1 for Linux, UNIX, and Windows and its annotated XML schema decomposition

Annotated XML schema decomposition is a type of decomposition that operates based on annotations specified in an XML schema. Decomposition, sometimes referred to as "shredding", is the process of storing content from an XML document in columns of relational tables. After it is decomposed, the data then has the SQL type of the column it was inserted into.

An XML schema consists of one or more XML schema documents. In annotated XML schema decomposition, or schema-based decomposition, you control decomposition by annotating a document's XML schema with decomposition annotations. These annotations specify details such as the name of the target table and column the XML data is to be stored in, the default SQL schema for when a target table's SQL schema is not identified, as well as any transformation of the content before it is stored.

The XML decomposition annotations belong to the http://www.ibm.com/xmlns/prod/db2/xdb1 namespace and are identified by the "db2-xdb" prefix throughout the documentation. You can select your own prefix; however, if you do, you must bind your prefix to the following namespace: http://www.ibm.com/xmlns/prod/db2/xdb1. The decomposition process recognizes only annotations that are under this namespace at the time the XML schema is enabled for decomposition.

The decomposition annotations are only recognized by the decomposition process if they are added to element and attribute declarations, or as global annotations, in the schema document. They are either specified as attributes or as part of an <xs:annotation> child element of the element or attribute declaration. Annotations added to complex types, references, or other XML schema constructs are ignored. Although these annotations exist in the XML schema documents, they do not affect the original structure of the schema document, nor do they participate in the validation of XML documents. They are only referred to by the XML decomposition process.

The annotated schema documents must be stored in and registered with the XML schema repository (XSR). The schema must then be enabled for decomposition. If a schema document is not successfully registered, or if the XML schema is not enabled for decomposition with the XSR, then the decomposition process cannot infer the mapping and therefore cannot determine how elements and attributes should be decomposed; schema-based decomposition cannot occur in this case.

After the successful registration of the annotated schema, decomposition can be performed either by calling one of the decomposition stored procedures or by executing the DECOMPOSE XML DOCUMENT command.

DB2 supports a set of annotations used by the annotated XML schema decomposition process to map elements and attributes from an XML document to target database tables. Table C-4 on page 623 summarized some of the XML decomposition annotations grouped by the tasks and actions you use the annotations to perform. Refer to the DB2 V9.1 for Linux, UNIX and Windows Information Center and search for the specific annotation to obtain further details about a particular annotation.

Table C-4 XML decomposition annotations grouped by tasks

Task	XML decomposition annotation	Action
Map multiple elements or attributes to column and table pair	db2-xdb:defaultSQLSchema	Specifies the default SQL schema for all table names referenced in the XML schema that are not explicitly qualified using the db2-xdb:table annotation.
	db2-xdb:rowSet	Specifies an XML element or attribute mapping to a target base table
	db2-xdb:column	Specifies a column name of the table to which an XML element or attribute has been mapped
	db2-xdb:rowSetMapping	Maps a single XML element or attribute to one or more column and table pairs
	db2-xdb:table	Maps multiple XML elements or attributes to the same target column; or enables you to specify a target table that has an SQL schema different from the default SQL schema specified by <db2-xdb:defaultSQLSchema>
Control data to be decomposed by specifying the type of content to be inserted for an element of complex type (text, string, or markup)	db2-xdb:contentHandling	Specifies the type of content that will be decomposed into a table for an element of complex type or simple type.
Specify any content transformation to be applied before insertion	db2-xdb:normalization	Specifies the normalization of whitespace in the XML data to be inserted or to be substituted
	db2-xdb:expression	Specifies a customized expression, the result of which is inserted into the table this element is mapped to
	db2-xdb:truncate	Specifies whether truncation is permitted when an XML value is inserted into a character target column
Conditional decomposition on filtered data based on the item's content or the context in which it appears	db2-xdb:condition	Specifies a condition that determines if a row will be inserted into a table. A row that satisfies the condition might be inserted (depending on other conditions for the rowSet, if any); a row that does not satisfy the condition will not be inserted
	db2-xdb:locationPath	Filters the data to be decomposed based on the item's content or the context in which it appears

Please see the following article *From DAD to annotated XML schema decomposition* for further detail on how to convert from DAD to annotated XML schema decomposition.

http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604pradhan/

C.8 XML APIs and application support

Application development support of the new native XML data store enables applications to combine XML and relational data access and storage. Both internally encoded and externally encoded XML data are supported.

The following programming languages support the new XML data type:

- C or C++ (embedded SQL or DB2 CLI)
- COBOL
- Java (JDBC or SQLJ)
- C# and Visual Basic (DB2 .NET Data Provider)
- PHP

Java, DB2 CLI, or DB2 .NET Data Provider applications can use XML, binary, or character application data types to store XML data in or fetch XML data from XML columns. Embedded SQL applications can use XML, LOB or LOB_FILE application data types.

XML data can be passed to SQL and external procedures by including parameters of data type XML in CREATE PROCEDURE parameter signatures. Existing procedure features support the implementation of procedural logic flow around SQL statements that produce or make use of XML values as well as the temporary storage of XML data values in variables.

The encoding of XML data can be derived from the data itself, which is known as *internally encoded data*, or from external sources, which is known as *externally encoded data*. The application data type that you use to exchange the XML data between the application and the XML column determines how the encoding is derived.

XML data that is in character or graphic application data types is considered to be externally encoded. Like character and graphic data, XML data that is in these data types is considered to be encoded in the application code page. XML data that is in a binary application data type or binary data that is in a character data type is considered to be internally encoded.

When you send externally encoded data to a DB2 database, the database manager checks for internal encoding. The effective CCSID that is associated with the internal encoding must match the external encoding. Otherwise, an error occurs. Therefore, it is important to avoid unnecessary code page conversion in order to use XML efficiently and effectively.

C.8.1 Embedded SQL

To transmit XML data between the database server and an embedded SQL application, you need to declare host variables in your application source code. In the declaration section of the application, you can declare host variables as LOB data types. Table C-5 lists the declarations you can use and the encoding implications.

Table C-5 XML host variables in embedded SQL applications

Declaration	Base SQL Types	Encoding
SQL TYPE IS XML AS CLOB(n) <hostvar_name>	SQL_TYP_CLOB	Where <hostvar_name> is a CLOB host variable that contains XML data encoded in the mixed codepage of the application
SQL TYPE IS XML AS DBCLOB(n) <hostvar_name>	SQL_TYP_DBCLOB	Where <hostvar_name> is a DBCLOB host variable that contains XML data encoded in the application graphic codepage

SQL TYPE IS XML AS BLOB(n) <hostvar_name>	SQL_TYP_BLOB	Where <hostvar_name> is a BLOB host variable that contains XML data internally encoded
SQL TYPE IS XML AS CLOB_FILE <hostvar_name>	SQL_TYP_CLOB_FILE	Where <hostvar_name> is a CLOB file that contains XML data encoded in the application mixed codepage
SQL TYPE IS XML AS DBCLOB_FILE <hostvar_name>	SQL_TYP_DBCLOB_FILE	Where <hostvar_name> is a DBCLOB file that contains XML data encoded in the application graphic codepage
SQL TYPE IS XML AS BLOB_FILE <hostvar_name>	SQL_TYP_BLOB_FILE	Where <hostvar_name> is a BLOB file that contains XML data internally encoded

Example C-28 shows a sample application that demonstrates how to reference XML host variables in C.

Example: C-28 Sample embedded C application

```
EXEC SQL BEGIN DECLARE;
  SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
  SQL TYPE IS XML AS BLOB( 10K ) xmlblob;
  SQL TYPE IS CLOB( 10K ) clobBuf;
EXEC SQL END DECLARE SECTION;

// as XML AS CLOB
// The XML value written to xmlBuf will be prefixed by an XML declaration similar
// to: <?xml version = "1.0" encoding = "ISO-8859-1"?>
// Note: The encoding name will depend upon the application codepage
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001';

// as XML AS BLOB
// The XML value written to xmlblob will be prefixed by an XML declaration similar
// to: <?xml version = "1.0" encoding = "UTF-8"?>
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001';

// as CLOB
// The output will be encoded in the application character codepage,
// but will not contain an XML declaration
EXEC SQL SELECT XMLSERIALIZE (xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id = '001';
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE (:clobBuf PRESERVE WHITESPACE)
   WHERE id = '001';
```

Example C-29 shows a COBOL sample application which uses the XML data type.

Example: C-29 Sample embedded Cobol application

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 xmlBuf USAGE IS SQL TYPE IS XML as CLOB(5K).
  01 clobBuf USAGE IS SQL TYPE IS CLOB(5K).
  01 xmlblob  USAGE IS SQL TYPE IS BLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.

* as XML
EXEC SQL SELECT xmlCol INTO :xmlBuf
   FROM myTable
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlBuf
   WHERE id = '001'.

* as BLOB
EXEC SQL SELECT xmlCol INTO :xmlblob
   FROM myTable
   WHERE id = '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = :xmlblob
   WHERE id = '001'.

* as CLOB
EXEC SQL SELECT XMLSERIALIZE(xmlCol AS CLOB(10K)) INTO :clobBuf
   FROM myTable
   WHERE id= '001'.
EXEC SQL UPDATE myTable
   SET xmlCol = XMLPARSE(:clobBuf) PRESERVE WHITESPACE
   WHERE id = '001'.
```

C.8.2 JDBC or SQLJ

Currently there is no standardized XML data type in JDBC. For Java applications, you can use the com.ibm.db2.jcc.DB2Xml DB2 proprietary data type for XML columns and parameters, and invoke one of the `getDB2Xmlxxx` methods from the `com.ibm.db2.jcc.DB2Xml` interface to retrieve the XML data. Table C-6 shows the list of `getDB2Xmlxxx` methods.

Table C-6 DB2Xml methods, data types, and added encoding specifications

Method	Output data type	Type of XML internal encoding declaration added
DB2Xml.getDB2AsciiStream	InputStream	None
DB2Xml.getDB2BinaryStream	InputStream	None
DB2Xml.getDB2Bytes	byte[]	None

DB2Xml.getDB2CharacterStream	Reader	None
DB2Xml.getDB2String	String	None
DB2Xml.getDB2XmlAsciiStream	InputStream	US-ASCII
DB2Xml.getDB2XmlBinaryStream	InputStream	Specified by getDB2XmlBinaryStream targetEncoding parameter
DB2Xml.getDB2XmlBytes	byte[]	Specified by getDB2XmlBytes targetEncoding parameter
DB2Xml.getDB2XmlCharacterStream	Reader	ISO-10646-UCS-2
DB2Xml.getDB2XmlString	String	ISO-10646-UCS-2

Example C-30 shows a JDBC application that reads XML data from file `c6.xml` as binary data, and insert the data into an XML column.

Example: C-30 Sample JDBC code that reads and insert XML data

```
PreparedStatement insertStmt = null;
String sqls = null;
int cid = 1015;
sqls = "INSERT INTO MyCustomer (Cid, Info) VALUES (?, ?)";
insertStmt = conn.prepareStatement(sqls);
insertStmt.setInt(1, cid);
File file = new File("c6.xml");
insertStmt.setBinaryStream(2, new FileInputStream(file), (int)file.length());
insertStmt.executeUpdate();
```

C.8.3 ODBC/CLI

A new XML type SQL_XML is now available. This data type corresponds to the native XML data type of the DB2 database, which is used to define columns that store well-formed XML documents. DB2 will avoid unnecessary code page conversion for data that is defined as XML data type. The SQL_XML type can be bound to the following C types: SQL_C_BINARY, SQL_C_CHAR, SQL_C_WCHAR, and SQL_C_DBCHAR. Using the default SQL_C_BINARY type, however, instead of character types, is recommended to avoid possible data loss or corruption resulting from code page conversion when character types are used. Example C-31 shows how to update XML data in an XML column using the recommended SQL_C_BINARY type.

Example: C-31 Updating XML data in an XML column

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
```

```
SQLExecute (hStmt);
```

C.8.4 .NET

New DB2Xml class is added to DB2 .NET provider to encapsulate the XML column type. DB2DataReader.Getxxx methods can be used to retrieve the Xml data from the column. See the list of `Getxxx` methods in Table C-7.

Table C-7 DB2 .NET provider Getxxx methods to retrieve XML data

Method	Description
GetXml	Return an internal representation of an XML column (DB2Xml). The resulting type cannot be processed directly by an application, it can only be passed back to DB2 in places where an input variable of type XML is required (for example, an insert statement, procedure CALL).
GetString	Return a string representing the contents of an XML column. There will be no encoding attribute in the document, it will be in Unicode (UTF-16).
GetBytes	Return a byte array representing the contents of an XML column. There will be no encoding attribute in the document, it will be in Unicode.
GetStream	Return a Stream object representing the contents of an XML column. There will be no encoding attribute in the document, it will be in Unicode
GetXmlReader	Return an XmlReader for the specified column (column must be of character, binary or XML type). Note: For a character column, there must be no encoding attribute, or it must match the server codepage, and it will be removed during codepage conversion on the client.

Example C-32 shows an example of an application using the .NET interface to retrieve XML data.

Example: C-32 Retrieving XML data with .NET

```
DB2Command cmd = DB2Connection.CreateCommand();
cmd.CommandText = "select deptdoc from dept";
cmd.CommandType = CommandType.Text;
DB2DataReader dr = cmd.Execute();
dr.Read( );
// retrieve XML column as an XML reader
XmlReader xml0 = dr.GetXmlReader( 0 );
```

C.8.5 PHP

PHP stands for Hypertext Preprocessor (PHP). See also Chapter 16, “PHP client design” on page 507. The ibm_db2 is an extension written, maintained, and supported by IBM for accesses to DB2 databases. You can also connect to IBM data servers via the following PHP drivers (including the ibm_db2 extension):

- Unified ODBC (ext/odbc): Support all IBM data servers
- Extension for DB2 (ibm_db2): support IBM DB2 on all platforms

- PHP Data Objects(PDO): supports all IBM data servers

In this redbook we discussed PHP in detail in Chapter 16, “PHP client design” on page 507. Please read Chapter 16 for detailed information about how to implement Web services with PHP.

C.9 Create and register an XML schema using Developer Workbench

You have already learned that in DB2 V9 for Linux, UNIX and Windows, Developer Workbench replaces DB2 V8’s Development Center in 5.1, “DB2 Developer Workbench” on page 104. You have also seen how to create a stored procedure in Developer Workbench just like in Development Center in DB2 V8.

Now, we will show you how to create and register an XML schema on DB2 with Developer Workbench. This is part of the new XML support provided by DB2 V9.1 for Linux, UNIX and Windows. We will also show how to perform XML document validation in Developer Workbench. You should have the following ready before you proceed:

You must install and configure the a UTF-8 database. If you have already created the `XMLDB database` in Appendix C.2, “Using the native XML data store in DB2 V9.1 for Linux, UNIX and Windows” on page 596, then you may skip this step. Otherwise, from the DB2 Command Line Processor, run the following command:

```
CREATE DATABASE xmldb USING CODESET UTF-8 TERRITORY US
```

The name of the database is `XMLDB`. The territory identifier shows the code that is used by the database manager internally to provide region-specific support. All territory codes are supported. In the command above, the territory identifier is the United States. In order to create the database, you must have SYSADM or SYSCTRL authorization.

1. Creating a connection to the XMLDB database

   The workbench provides wizards that make it easy for you to connect to both DB2 and non-DB2 databases and to display the status of your connections. The New Connection wizard creates a connection to a database that is displayed in the Database Explorer view. Using the view, you have the option to create a DB2 database connection. For the purposes of this tutorial you will connect to the XMLDB database.

   To create a connection to the XMLDB database:

   a. In the Database Explorer view, right-click the **Connections** → **New Connection**. The New Connection wizard opens.

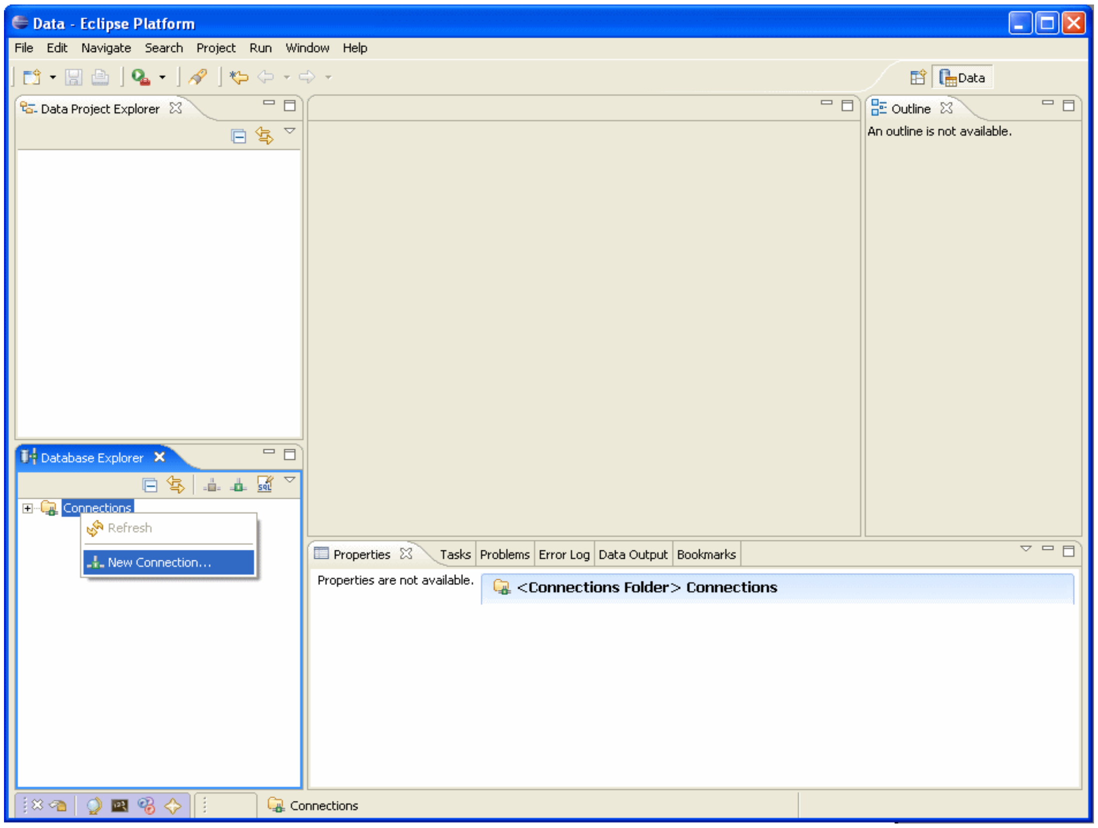


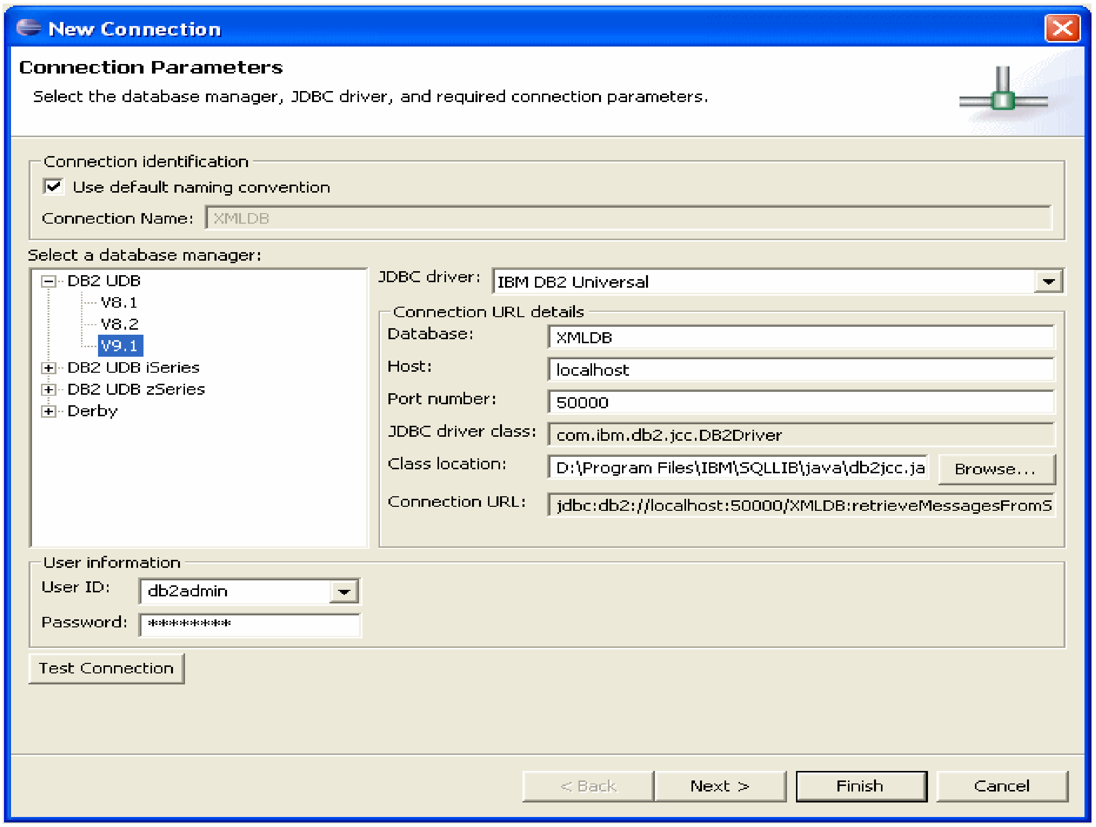
Figure C-15 Creating a new database connection in Developer Workbench

b. In the Database field, type `XMLDB`.

c. Select the DB2 UDB database manager that matches the DB2 server that manages the XMLDB database.

d. You must connect via a JCC driver 3.0 or greater.

e. Specify the user ID and password that you want to connect with, and then click **Finish.**


New Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: XMLDB

Select a database manager:

- DB2 UDB
  - V8.1
  - V8.2
  - V9.1
- DB2 UDB iSeries
- DB2 UDB zSeries
- Derby

JDBC driver: IBM DB2 Universal

Connection URL details

Database:	XMLDB
Host:	localhost
Port number:	50000
JDBC driver class:	com.ibm.db2.jcc.DB2Driver
Class location:	D:\Program Files\IBM\SQLLIB\java\db2jcc.ja
Connection URL:	jdbc:db2://localhost:50000/XMLDB:retrieveMessagesFromS

User information

User ID: db2admin

Password: *********

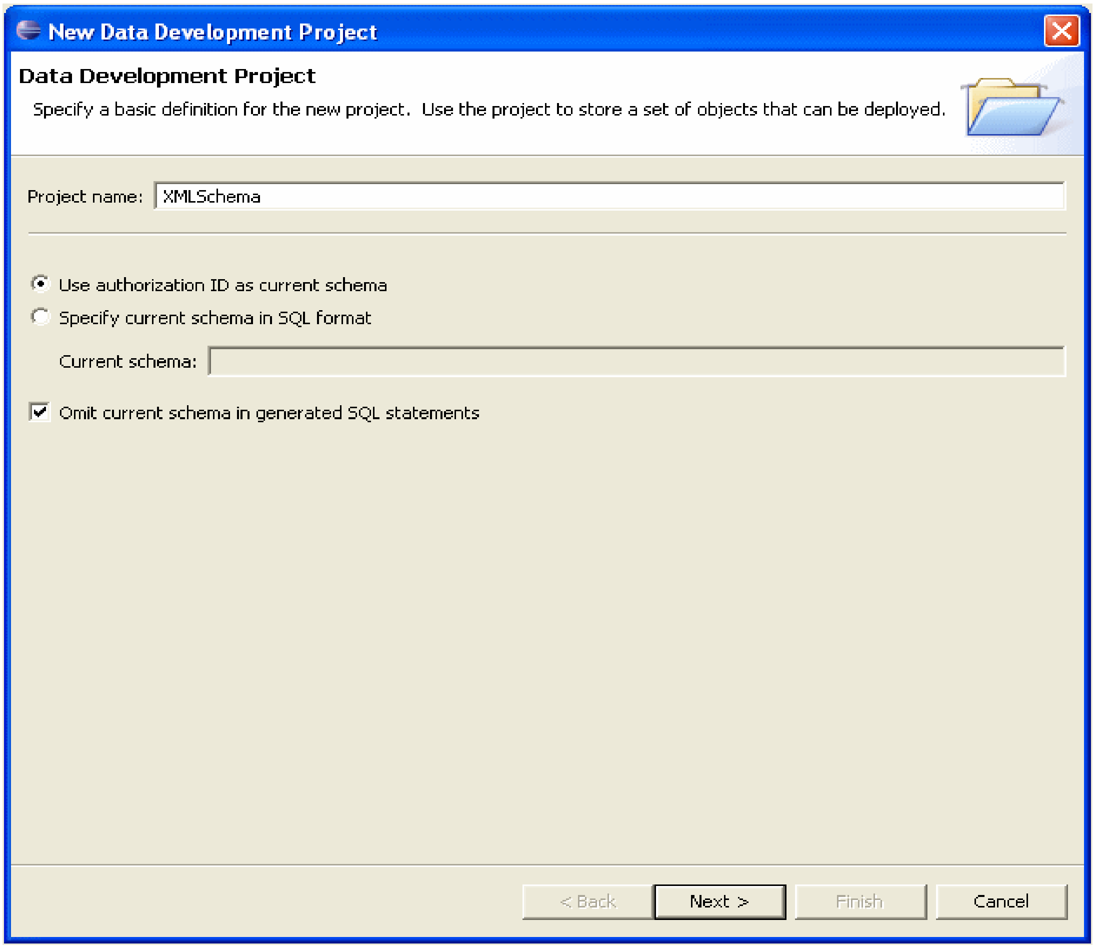
Figure C-16 New Connection wizard

2. Creating a project for XML schema development

   Before you create database objects, you must create a Data Development project.

   a. From the menu bar, select **File** → **New** → **Data Development** Project. The New Data Development Project wizard opens.

   b. In the **Project name** field, type `XMLSchema`.


New Data Development Project

Data Development Project

Specify a basic definition for the new project. Use the project to store a set of objects that can be deployed.

Project name: XMLSchema

(●) Use authorization ID as current schema

( ) Specify current schema in SQL format

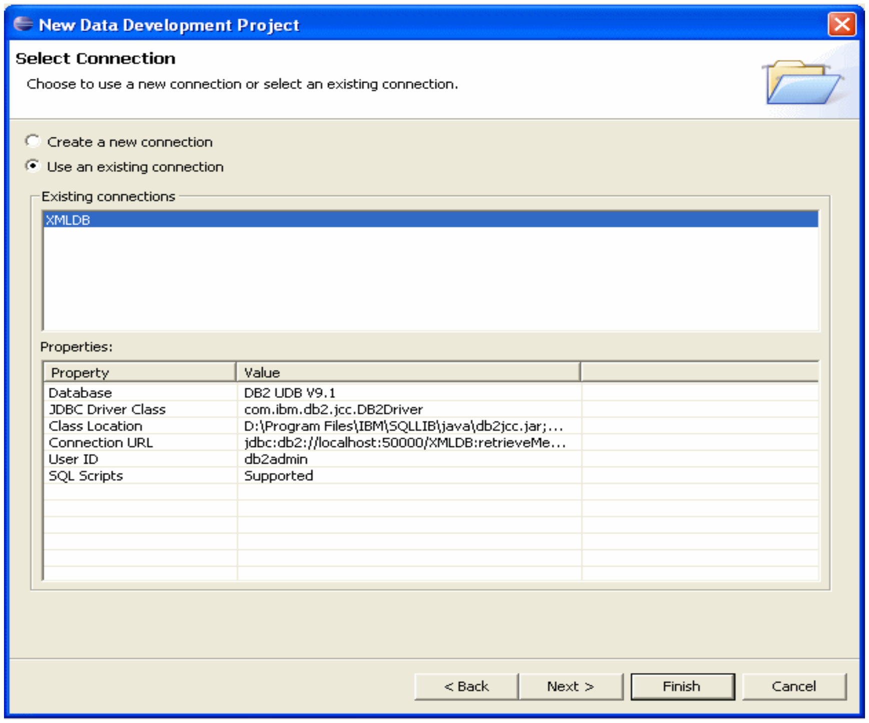
Current schema:

[✓] Omit current schema in generated SQL statements

Figure C-17 New Data Development Project

c. Click Next.

d. Under **Use an existing connection**, select **XMLDB** and browse its properties to confirm that this is the correct database.


New Data Development Project

Select Connection

Choose to use a new connection or select an existing connection.

( ) Create a new connection
[✓ Use an existing connection]

Existing connections

XMLDB

Properties:

Property	Value
Database	DB2 UDB V9.1
JDBC Driver Class	com.ibm.db2.jcc.DB2Driver
Class Location	D:\Program Files\IBM\SQLLIB\java\db2jcc.jar;...
Connection URL	jdbc:db2://localhost:50000/XMLDB:retrieveMe...
User ID	db2admin
SQL Scripts	Supported

Figure C-18   New XMLDB connection properties

e. Click Finish. The new project is displayed in the Data Project Explorer view.

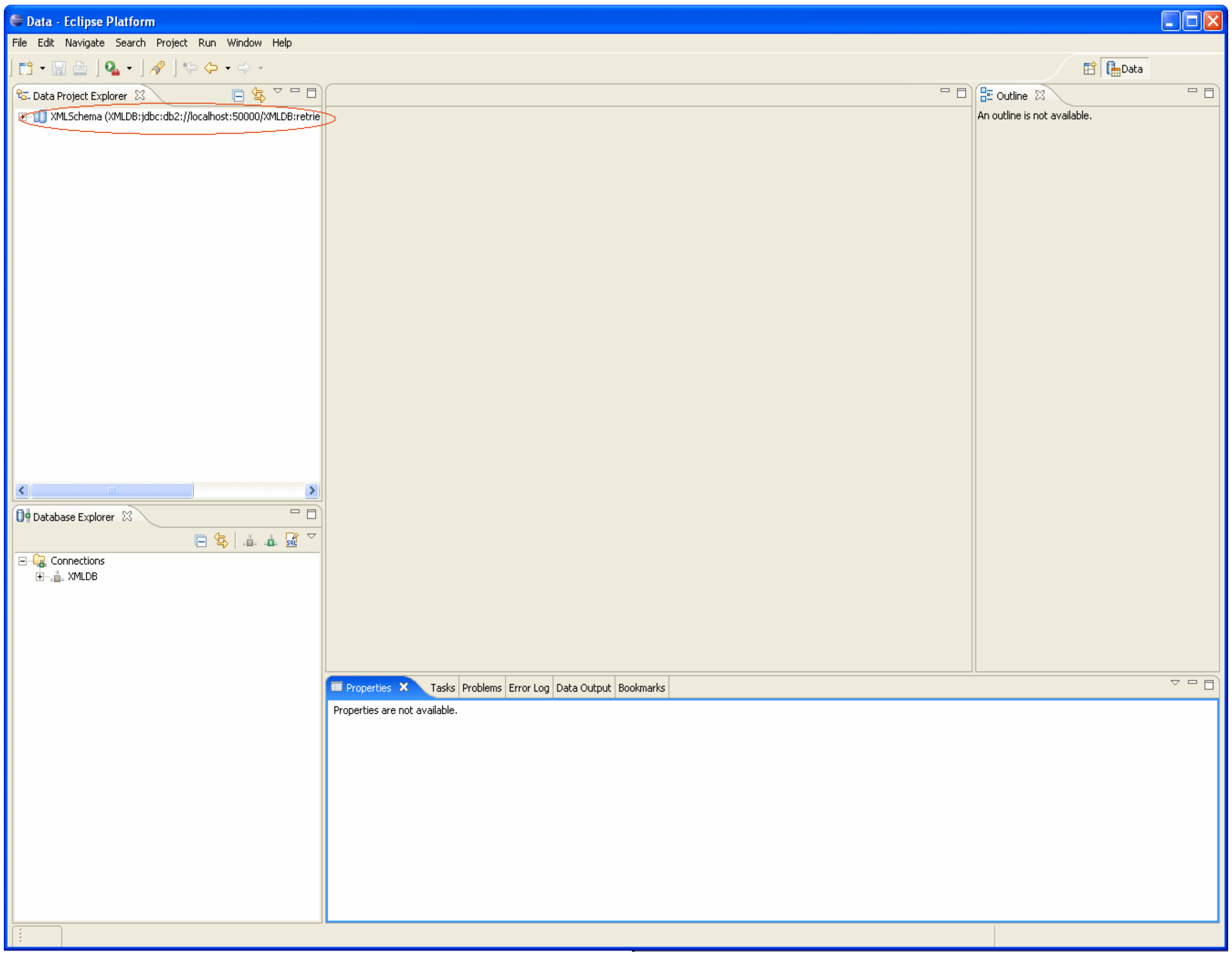

Data - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Data
Data Project Explorer
XMLSchema (XMLDB:jdbc:db2://localhost:50000/XMLDB:retrie
Outline
An outline is not available.
Database Explorer
Connections
XMLDB
Properties
Tasks
Problems
Error Log
Data Output
Bookmarks
Properties are not available.

Figure C-19 XMLSchema project in the Data Project Explorer

3. Creating an XML schema document

   Now you will use the XSD editor to create an XML document for your XML schema. The XML document is used to define the XML schema. XML schemas must contain one or more XML schema documents. To create an XML schema document:

   a. In the File menu, select **New** → **Other**. The New wizard opens.

   b. Expand the **XML** folder. and select **XML Schema**.

   c. Click Next. The Create XML Schema wizard opens.

   d. Select the **XMLSchema** project.

   e. In the **File name** field, type `employee.xsd`.

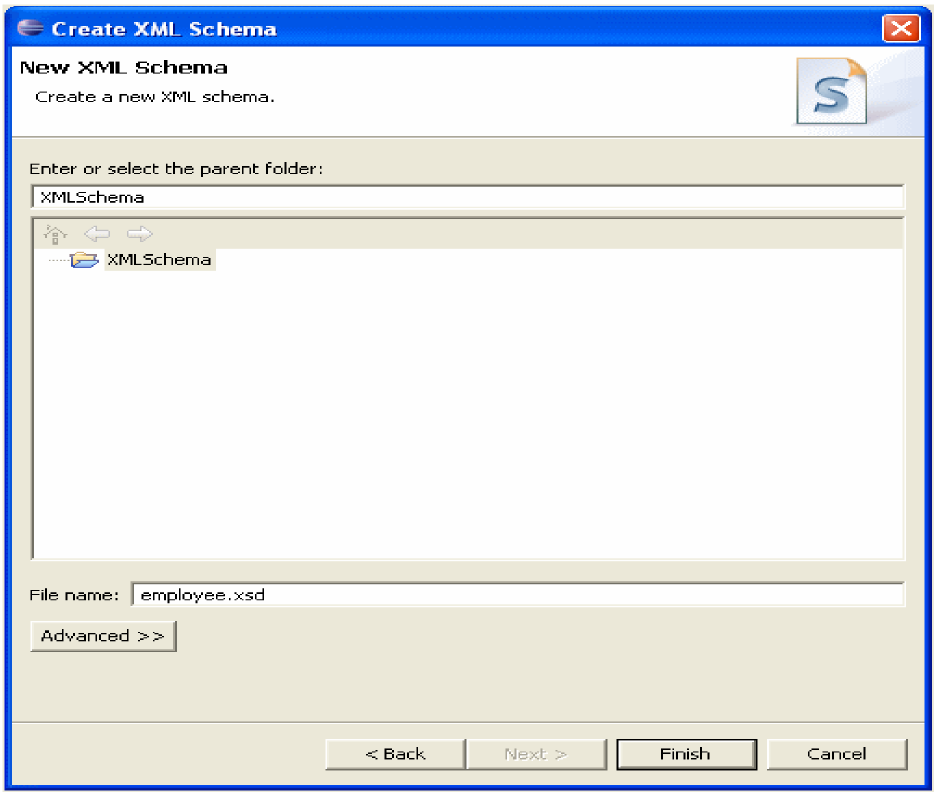


Figure C-20 XML Schema wizard

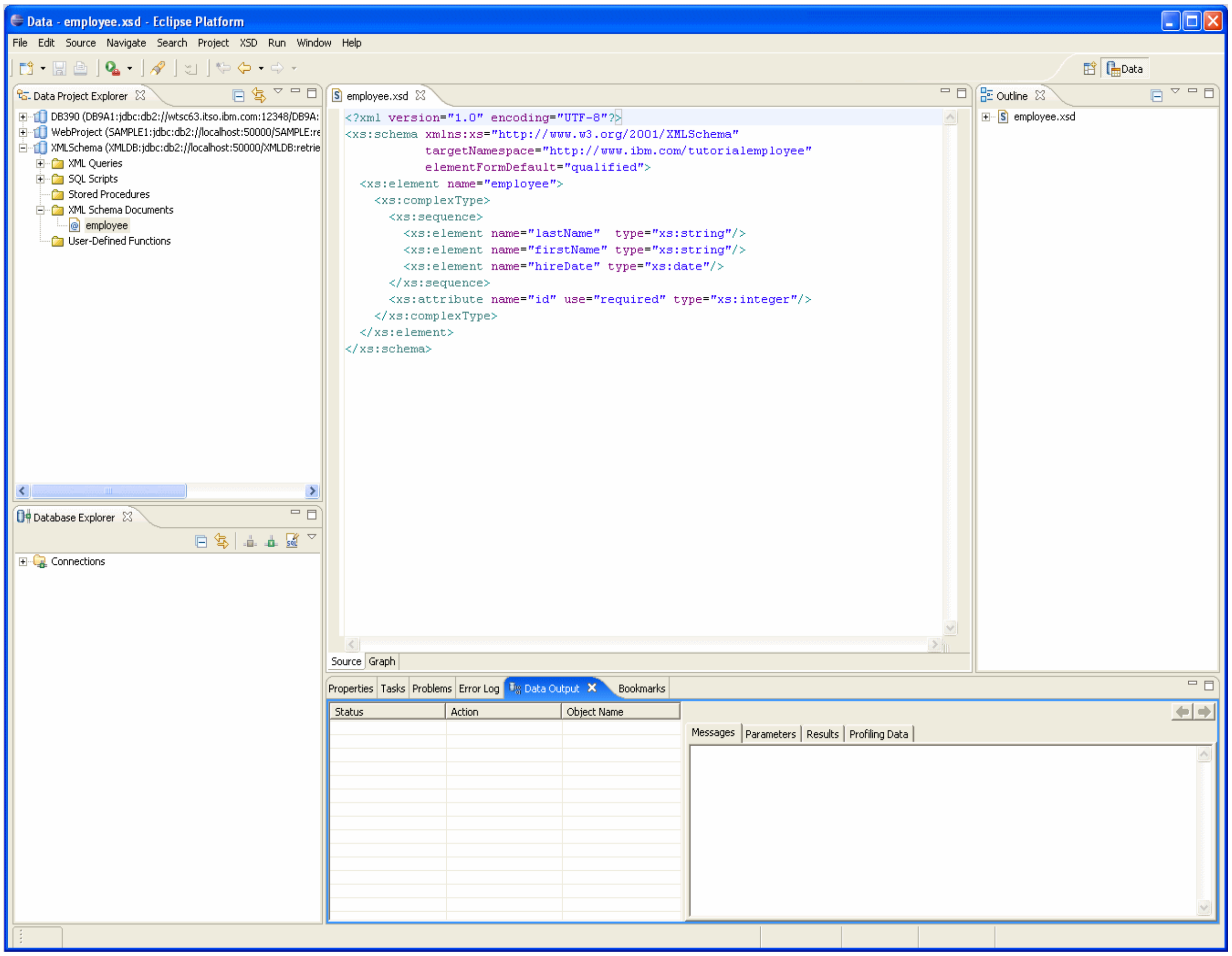
f. Click Finish to create the XML schema. The XSD editor opens.

g. In the XSD editor, replace any existing XML with the following XML as shown in Example C-33.

Example: C-33 Creating an XML document for XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.ibm.com/tutorialemployee"
           elementFormDefault="qualified">
  <xs:element name="employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="lastName"  type="xs:string"/>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="hireDate" type="xs:date"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

h. In the File menu, select **Save**.


Data - employee.xsd - Eclipse Platform
File Edit Source Navigate Search Project XSD Run Window Help
Data Project Explorer
DB390 (DB9A1:jdbc:db2://wtsc63.itso.ibm.com:12348/DB9A:
WebProject (SAMPLE1:jdbc:db2://localhost:50000/SAMPLE:re
XMLSchema (XMLDB:jdbc:db2://localhost:50000/XMLDB:retrie
XML Queries
SQL Scripts
Stored Procedures
XML Schema Documents
employee
User-Defined Functions
Database Explorer
Connections
employee.xsd
Source Graph
Outline
employee.xsd
Properties Tasks Problems Error Log Data Output Bookmarks
Status Action Object Name
Messages Parameters Results Profiling Data

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.ibm.com/tutorialemployee"
           elementFormDefault="qualified">
  <xs:element name="employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="lastName"  type="xs:string"/>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="hireDate" type="xs:date"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure C-21 XSD Editor

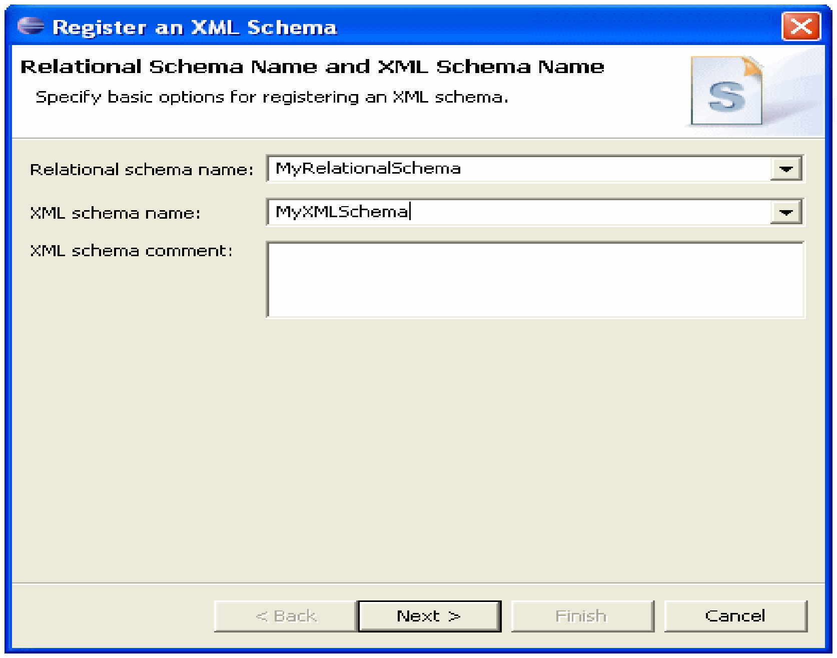
4. Registering the XML schema

   By using the Register an XML Schema wizard, you can register an XML schema to the DB2 database server. The XML schema will be used to validate XML column data by using the Table editor. To register the XML schema:

   a. In the Data Project Explorer view, right-click the `employee` XML schema document, and select **Register an XML Schema**. The Register an XML Schema wizard opens.

   b. In the **Relational schema name** field, type `MyRelationalSchema`.

   c. In the XML schema name field, type `MyXMLSchema`.



Register an XML Schema

Relational Schema Name and XML Schema Name

Specify basic options for registering an XML schema.

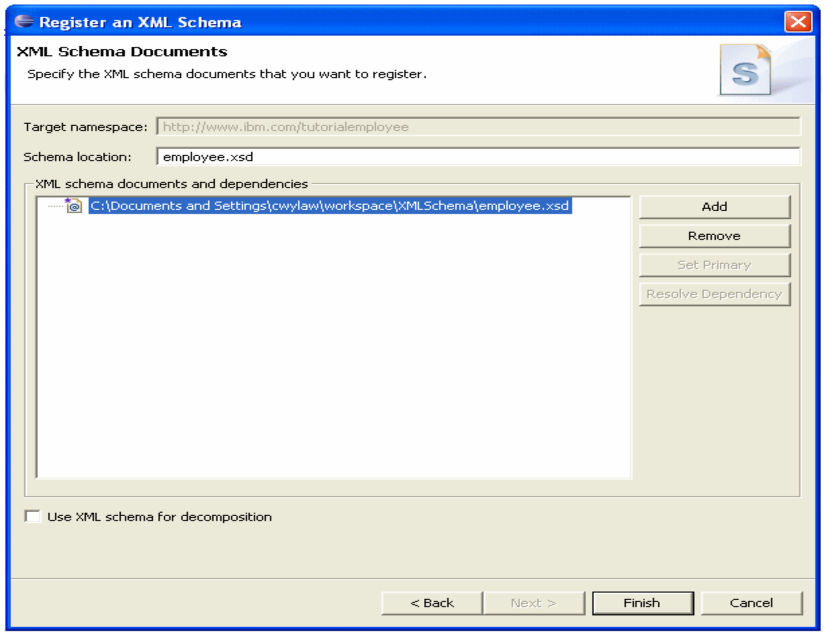
Relational schema name: MyRelationalSchema

XML schema name: MyXMLSchema

XML schema comment:

Figure C-22 Register an XML Schema wizard

d. Click Next. You should see the employee XML schema document in the **XML schema documents and dependencies** list.



Register an XML Schema

XML Schema Documents

Specify the XML schema documents that you want to register.

Target namespace: http://www.ibm.com/tutorialemployee

Schema location: employee.xsd

XML schema documents and dependencies

C:\Documents and Settings\cwylaw\workspace\XMLSchema\employee.xsd

Use XML schema for decomposition

Figure C-23 Continuing with Registering the employee XML schema

e. Click **Finish** to register the XML schema.

5. Using the XML schema to validate XML column data

You can use the XML schema to validate XML column data. To use the XML schema for validation:

a. In the Data Project Explorer, right-click the **SQL Scripts** folder, and select **New** → **SQL Statement**. The New SQL Statement wizard opens.

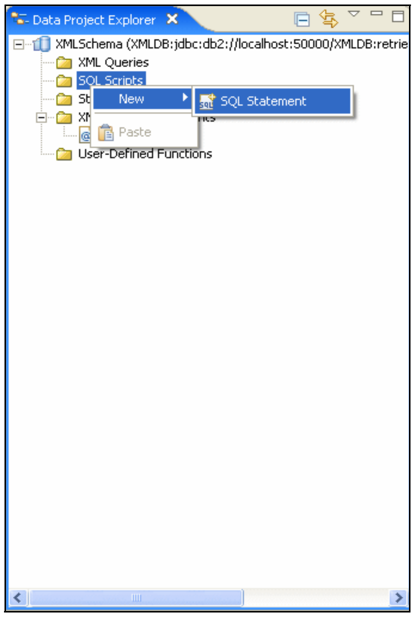


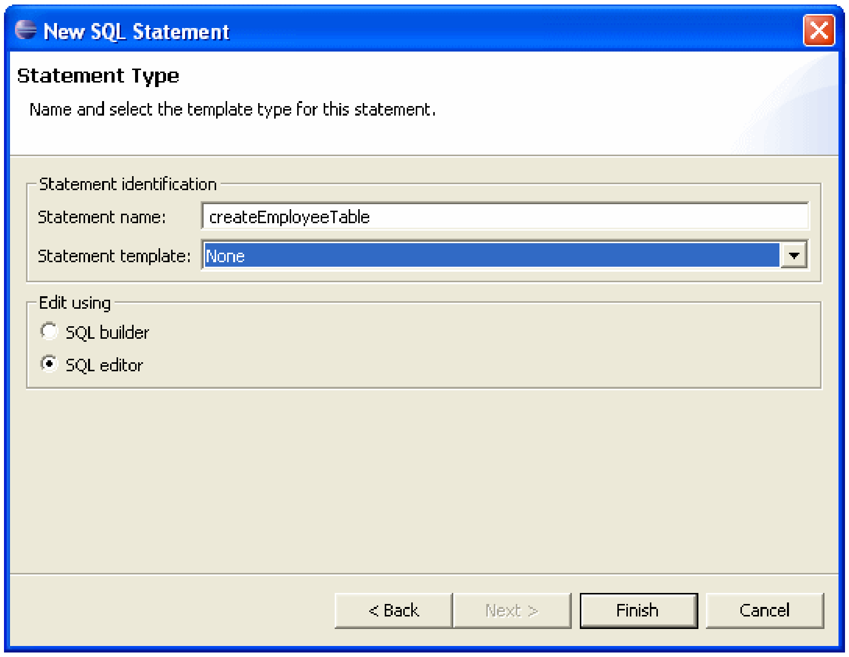
Figure C-24   Accessing the New SQL Statement wizard

b. Click **Next**.

c. In the Statement name field, type `createEmployeeTable`.

d. Select SQL editor.

e. In the **Statement template** field, select **None**.



New SQL Statement

Statement Type

Name and select the template type for this statement.

Statement identification

Statement name: createEmployeeTable

Statement template: None

Edit using

( ) SQL builder

(•) SQL editor

Figure C-25   New SQL Statement wizard

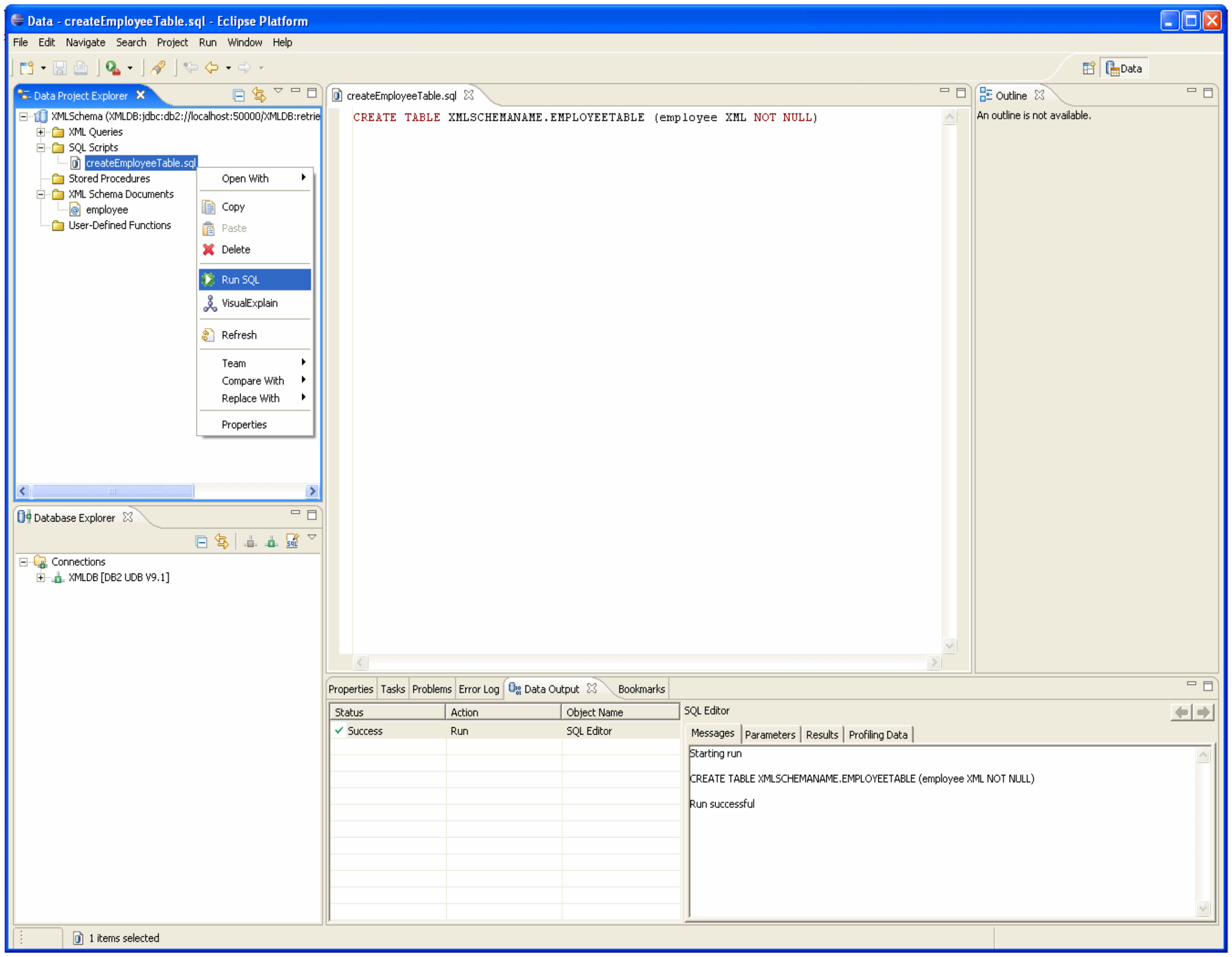
f. Click **Finish**. The SQL editor opens.

g. In the SQL editor, create a table that contains employee information as an XML value. Copy and paste the following SQL statement into the editor:

```
CREATE TABLE XMLSCHEMANAME.EMPLOYEETABLE (employee XML NOT NULL)
```

h. From the menu bar, select **File** → **Save**.

i. In the SQL editor, right-click `createEmployeeTable`.sql and select **Run SQL**. You can view the results in the Data Output view.



```
CREATE TABLE XMLSCHEMANAME.EMPLOYEETABLE (employee XML NOT NULL)
```

Status	Action	Object Name
✓ Success	Run	SQL Editor

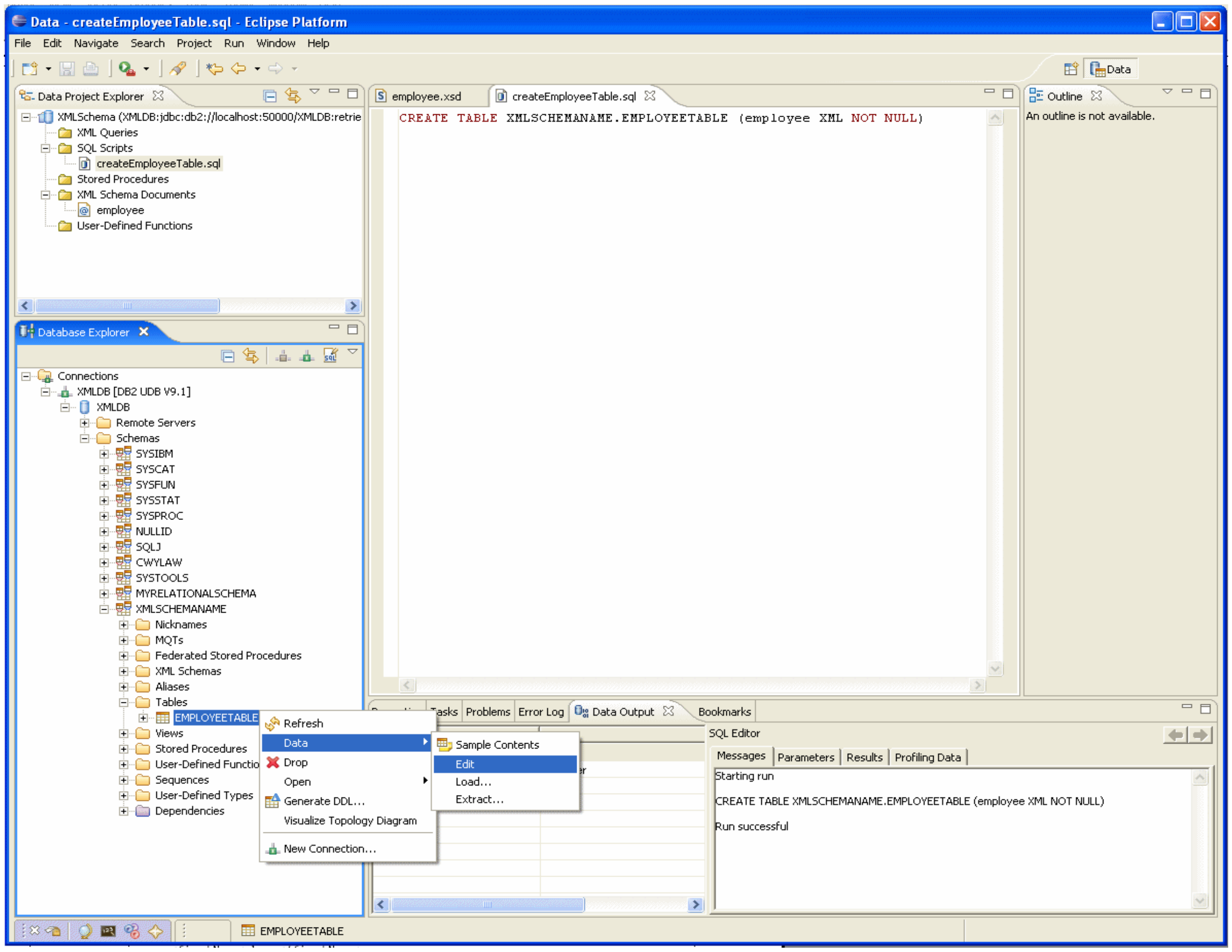
```
Starting run

CREATE TABLE XMLSCHEMANAME.EMPLOYEETABLE (employee XML NOT NULL)

Run successful
```

Figure C-26 Running a SQL in Developer Workbench

j. In the Database Explorer view, expand the XMLDB connection and navigate to the `XMLSCHEMANAME.EMPLOYEETABLE` table.

Tip: If you do not see the `XMLSCHEMANAME.EMPLOYEETABLE` table, right-click the `XMLSCHEMANAME` → **Refresh**.

k. Right-click the `XMLSCHEMANAME.EMPLOYEETABLE` table, and select **Data** → **Edit**. The Table editor opens.



```
CREATE TABLE XMLSCHEMANAME.EMPLOYEETABLE (employee XML NOT NULL)
```

```
Starting run

CREATE TABLE XMLSCHEMANAME.EMPLOYEETABLE (employee XML NOT NULL)

Run successful
```

Figure C-27 Accessing Table Editor

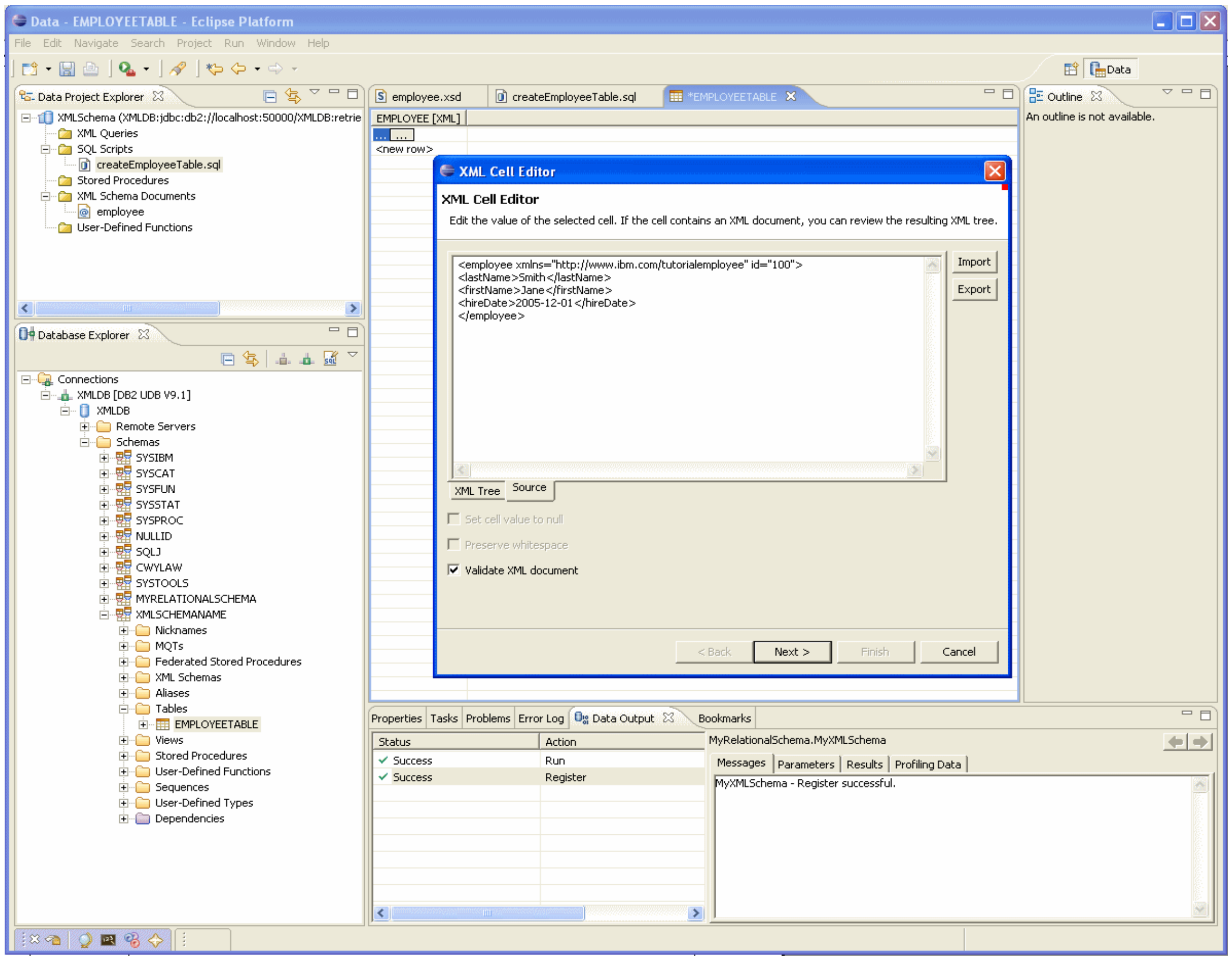
l. Click the ellipsis **[...]** button. The XML Cell editor opens.

m. Copy and paste the following XML:

```
<employee xmlns="http://www.ibm.com/tutorialemployee" id="100">
<lastName>Smith</lastName>
<firstName>Jane</firstName>
<hireDate>2005-12-01</hireDate>
</employee>
```

n. Select **Validate XML document**.

o. Click **Next**.



XML Cell Editor

Edit the value of the selected cell. If the cell contains an XML document, you can review the resulting XML tree.

```
<employee xmlns="http://www.ibm.com/tutorialemployee" id="100">
<lastName>Smith</lastName>
<firstName>Jane</firstName>
<hireDate>2005-12-01</hireDate>
</employee>
```

- [ ] Set cell value to null
- [ ] Preserve whitespace
- [x] Validate XML document

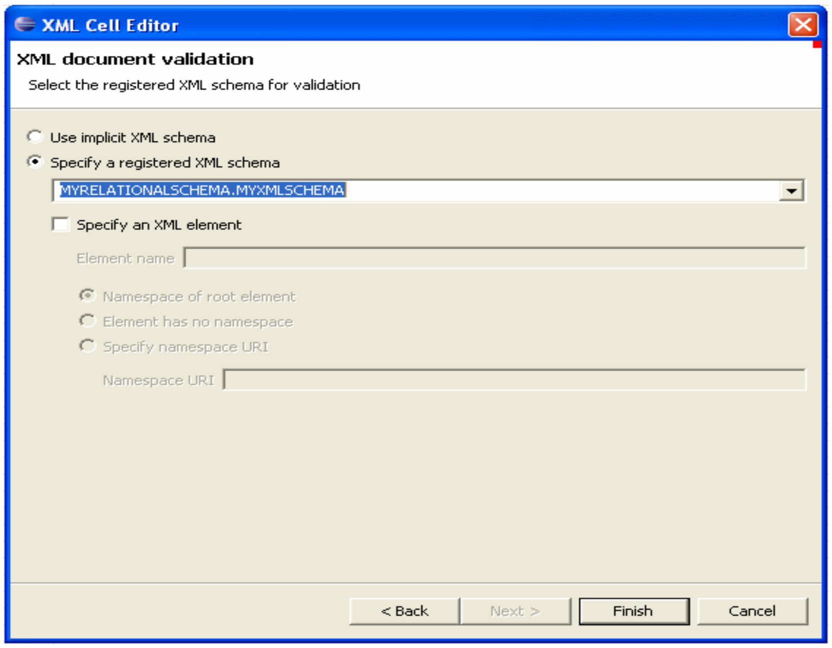
Status	Action
✓ Success	Run
✓ Success	Register

MyRelationalSchema.MyXMLSchema

MyXMLSchema - Register successful.

Figure C-28 XML Cell Editor

p. Select **Specify a registered XML schema**, and select `MyRelationalSchema.MyXMLSchema`.


XML Cell Editor

XML document validation

Select the registered XML schema for validation

( ) Use implicit XML schema
(●) Specify a registered XML schema

MYRELATIONALSCHEMA.MYXMLSCHEMA

[ ] Specify an XML element

Element name

(●) Namespace of root element
( ) Element has no namespace
( ) Specify namespace URI

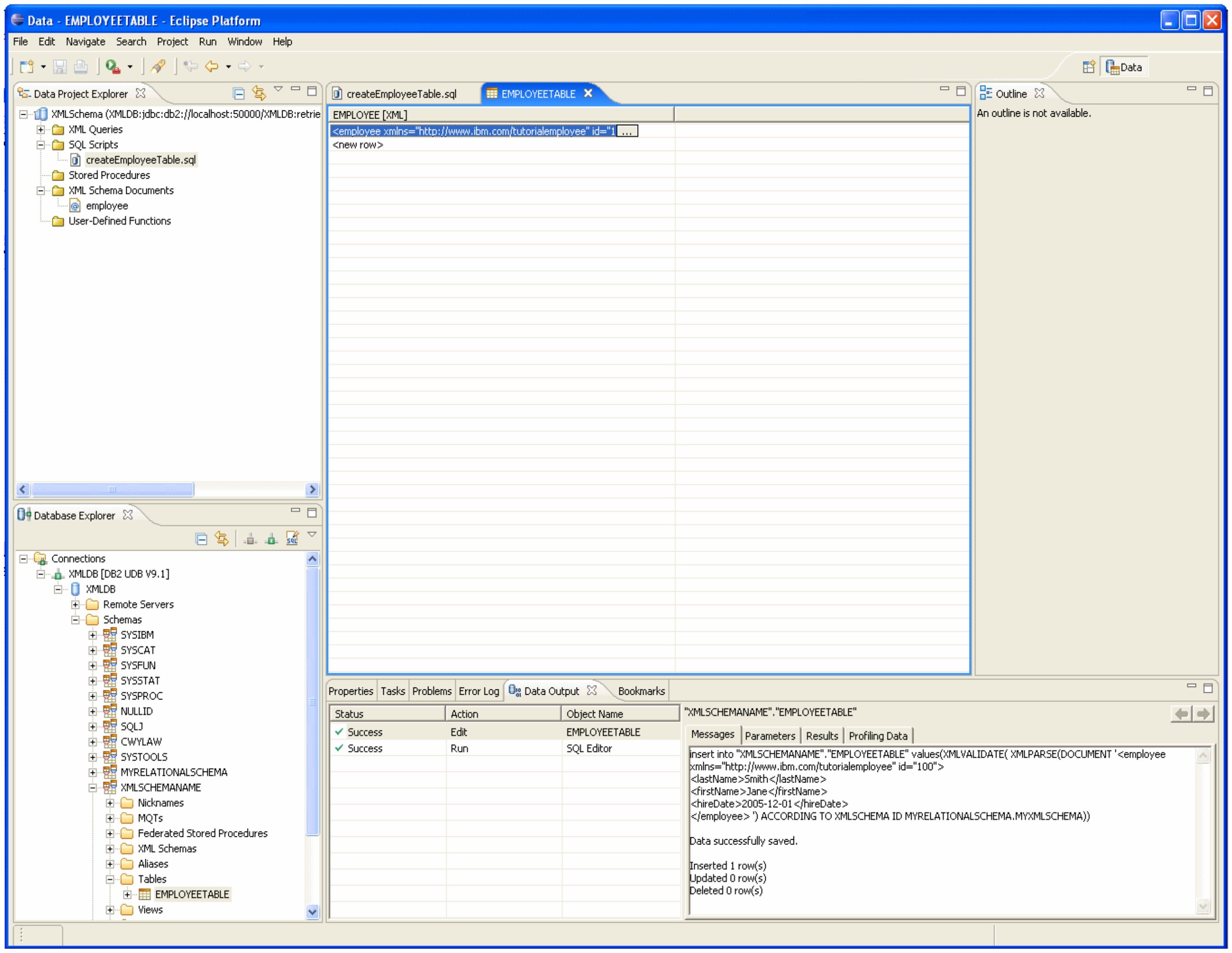
Namespace URI

Figure C-29 Setting up validation for XML document

q. Click **Finish**.

r. From the menu bar, select **File** → **Save**.

s. You can view the messages and results in the Data Output view to confirm that data has been inserted successfully.


Data - EMPLOYEETABLE - Eclipse Platform
Data Project Explorer
Database Explorer
createEmployeeTable.sql
EMPLOYEETABLE
EMPLOYEE [XML]
<employee xmlns="http://www.ibm.com/tutorialemployee" id="1
<new row>
Outline
An outline is not available.

Status	Action	Object Name
✓ Success	Edit	EMPLOYEETABLE
✓ Success	Run	SQL Editor

"XMLSCHEMANAME"."EMPLOYEETABLE"

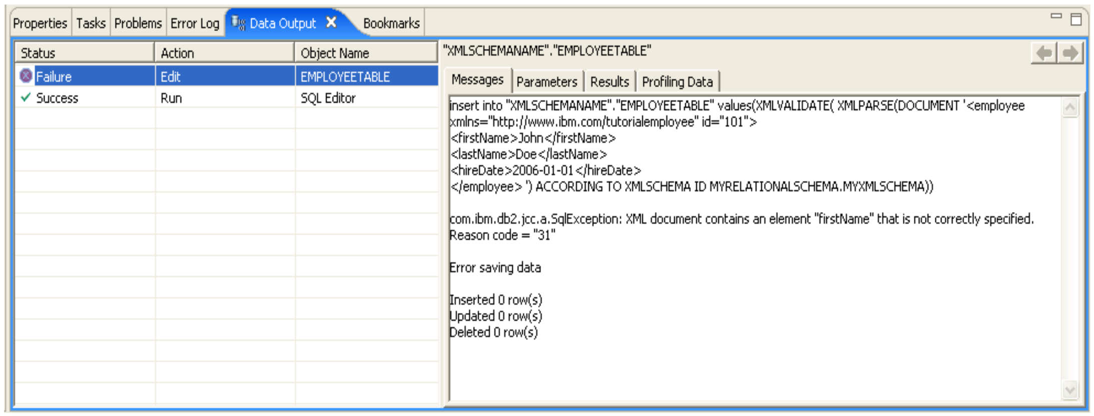
```
insert into "XMLSCHEMANAME"."EMPLOYEETABLE" values(XMLVALIDATE( XMLPARSE(DOCUMENT '<employee
xmlns="http://www.ibm.com/tutorialemployee" id="100">
<lastName>Smith</lastName>
<firstName>Jane</firstName>
<hireDate>2005-12-01</hireDate>
</employee> ') ACCORDING TO XMLSCHEMA ID MYRELATIONALSCHEMA.MYXMLSCHEMA))

Data successfully saved.

Inserted 1 row(s)
Updated 0 row(s)
Deleted 0 row(s)
```

Figure C-30 Developer Workbench showing one row of XML data

t. Repeat step m to r, this time insert a second row of XML data into the `XMLSCHEMANAME.EMPLOYEETABLE`:

```
<employee xmlns="http://www.ibm.com/tutorialemployee" id="101">
<firstName>John</firstName>
<lastName>Doe</lastName>
<hireDate>2006-01-01</hireDate>
</employee>
```

Notice how we reversed the order of firstName and lastName element in this case (compare with step m). This resulted in an error as shown at the bottom right hand corner of the Developer Workbench. This is because we have previously defined `lastName`, `firstName` and `hireDate` elements in a sequence, thus the sequential order of the elements cannot be changed. Since the XML document is not valid, an error occurred and is displayed in the Data Output view.



Status	Action	Object Name
Failure	Edit	EMPLOYEETABLE
✓ Success	Run	SQL Editor

"XMLSCHEMANAME"."EMPLOYEETABLE"

```
insert into "XMLSCHEMANAME"."EMPLOYEETABLE" values(XMLVALIDATE( XMLPARSE(DOCUMENT '<employee
xmlns="http://www.ibm.com/tutorialemployee" id="101">
<firstName>John</firstName>
<lastName>Doe</lastName>
<hireDate>2006-01-01</hireDate>
</employee> ') ACCORDING TO XMLSCHEMA ID MYRELATIONALSCHEMA.MYXMLSCHEMA))

com.ibm.db2.jcc.a.SqlException: XML document contains an element "firstName" that is not correctly specified.
Reason code = "31"

Error saving data

Inserted 0 row(s)
Updated 0 row(s)
Deleted 0 row(s)
```

Figure C-31   Error due to an invalid XML document

To summarize what we have done in this section C.9, “Create and register an XML schema using Developer Workbench”:

- You have just connected to the XMLDB database and created a Data Development project to work with the database.
- You use the XSD editor to create an XML document for your XML schema
- You registered the XML schema, and you have also used the XML schema to validate XML column data in Developer Workbench.

C.10 Restrictions on native XML store

The native XML data store adheres to some key restrictions. For detailed restrictions, refer to the documentation for a specific feature.

Restrictions on XML column definitions

XML columns can only be defined in a table of a database defined with the UTF-8 code set. The XML document stored in an XML column must be well-formed. While there is no architectural limit on the size of an XML value stored in the database, serialized XML data that is exchanged with the database is limited to 2 GB.

XML columns:

- Cannot be included as columns of keys, including primary, foreign, and unique keys, dimension keys of multi-dimensional clustering (MDC) tables, sequence keys of range-clustered tables, distribution keys, and data partitioning keys.
- Cannot be part of any index except an index over XML data
- Cannot have a default value specified by the WITH DEFAULT clause; if the column is nullable, the default for the column is NULL
- Cannot be used in a range-clustered table
- Cannot be used in a multi-dimensional clustering (MDC) table
- Cannot be used in a table with a distribution key
- Cannot be used in a table partitioned by range
- Cannot be included in typed tables and typed views

- Cannot be added to tables that have type-1 indexes defined on them (note that type-1 indexes are deprecated indexes; new indexes since DB2 UDB Version 8.1 are created as type-2 indexes)
- Cannot be referenced in CHECK constraints (except for a VALIDATED predicate)
- Cannot be referenced in generated columns
- Cannot be referenced in the triggered-action of a CREATE TRIGGER statement
- Cannot be specified in the select-list of scrollable cursors
- Cause data blocking to be disabled when retrieving XML data

Restrictions on database partitions

- The use of any features of the native XML data store will prevent future use of the Database Partitioning Feature available with DB2 Enterprise Server Edition.
- An XML column or XML schema repository (XSR) object cannot be defined in a table of a database with more than one database partition defined.
- If a database is defined with a single database partition and includes XML columns or XSR objects, then a new database partition cannot be added.

Restrictions on utilities

Loading data into tables containing XML columns using the load utility is not supported. Data movement of XML data should be performed using the import and export utilities.

C.11 XML schema for the DADX file

Here is the complete dadx.xsd file that describes the DADX schema.

Example: C-34 dadx.xsd describes the DADX schema

```
<?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>
------------------------------------------------------------------------------
<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>
------------------------------------------------------------------------------
<element name="DQS">
<annotation>
<documentation>
Defines the DQS tag.
</documentation>
</annotation>
<complexType/>
</element>
------------------------------------------------------------------------------
<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>
------------------------------------------------------------------------------
<element name="implements">
<annotation>
<documentation>
Defines the namespace and location of a set of WSDL bindings
defined elsewhere. This information is imported into the
WSDL document generated for this Web Service.
</documentation>
</annotation>
<complexType>
<attribute name="namespace"
type="anyURI" use="required"/>
<attribute name="location"
type="anyURI" use="required"/>
</complexType>
</element>
------------------------------------------------------------------------------
<element name="result_set_metadata">
<annotation>
<documentation>
Defines the metadata for a result set returned
by a stored procedure.
Each metadata element defines a global element
in the WSDL for the Web Service.
The metatdata name defines the name of its global element.
The metadata rowName defines the element name of each row.
The metadata contains one or more column definitions.
</documentation>
</annotation>
<complexType>
<sequence>
<element ref="dadx:column"
maxOccurs="unbounded"/>
</sequence>
<attribute name="name"
type="NCName"
use="required"/>
<attribute name="rowName"
type="NCName"
use="required"/>
</complexType>
</element>
------------------------------------------------------------------------------
<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>
-------------------------------------------------------------------------------
<simpleType name="columnType">
<restriction base="string">
<enumeration value="BIGINT"/>
<enumeration value="CHAR"/>
<enumeration value="CLOB"/>
<enumeration value="DATE"/>
<enumeration value="DECIMAL"/>
<enumeration value="DOUBLE"/>
<enumeration value="FLOAT"/>
<enumeration value="INTEGER"/>
<enumeration value="NUMERIC"/>
<enumeration value="REAL"/>
<enumeration value="SMALLINT"/>
<enumeration value="TIME"/>
<enumeration value="TIMESTAMP"/>
<enumeration value="TINYINT"/>
<enumeration value="VARCHAR"/>
</restriction>
</simpleType>
-------------------------------------------------------------------------------
<element name="operation">
<annotation>
<documentation>
Defines an operation of the Web Service.
Each operation has a unique name and is
optionally described
by WSDL documentation.
An operation is defined using one of the
```

```
supported operators.
</documentation>
</annotation>
<complexType>
<sequence>
<element ref="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>
-------------------------------------------------------------------------------
<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>
-----------------------------------------------------------------------------
<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>
-----------------------------------------------------------------------------
<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>
------------------------------------------------------------------------------
<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>
------------------------------------------------------------------------------
<element name="call">
<annotation>
<documentation>
```

```
Calls a stored procedure.
The call statement contains in, out, and
in/out parameters using host variable syntax.
The parameters are defined by a list of parameter
elements that are uniquely named
within the operation.
</documentation>
</annotation>
<complexType>
<sequence>
<element name="SQL_call"
type="string"/>
<element ref="dadx:parameter"
minOccurs="0"
maxOccurs="unbounded"/>
<element ref="dadx:result_set"
minOccurs="0"
maxOccurs="unbounded"/>
</sequence>
</complexType>
<unique name="callParameterNames">
<selector xpath="dadx:parameter"/>
<field xpath="@name"/>
</unique>
<unique name="callResultSetNames">
<selector xpath="dadx:result_set"/>
<field xpath="@name"/>
</unique>
</element>
------------------------------------------------------------------------------
<element name="result_set">
<annotation>
<documentation>
Defines a result set.
The name defines the element name of the result
set and becomes part of the output message.
The metatdata name refers to a result set metadata
element defined in the same document.
</documentation>
</annotation>
<complexType>
<attribute name="name"
type="NCName" use="required"/>
<attribute name="metadata"
type="NCName" use="required"/>
</complexType>
</element>
<element name="DAD_ref"
type="string"/>
<element name="collection_name"
type="string"/>
------------------------------------------------------------------------------
<element name="parameter">
<annotation>
<documentation>
```

```
Defines a named parameter. A parameter
must have it contents defined either by
an XML Schema element or type, but not both.
The parameter kind in one of in,
out, or in/out, with in being the default.
</documentation>
</annotation>
<complexType>
<attribute name="name"
type="NCName"
use="required"/>
<attribute name="element"
type="QName"/>
<attribute name="type"
type="QName"/>
<attribute name="kind"
type="dadx:parameterKindType"
default="in"/>
</complexType>
</element>
<simpleType name="parameterKindType">
<restriction base="string">
<enumeration value="in"/>
<enumeration value="out"/>
<enumeration value="in/out"/>
</restriction>
</simpleType>
-------------------------------------------------------------------------------
<element name="XML_result">
<annotation>
<documentation>
Defines a named column that contains XML documents.
The document type
must be defined by the XML Schema element
of its root.
</documentation>
</annotation>
<complexType>
<attribute name="name"
type="NCName"
use="required"/>
<attribute name="element"
type="QName"
use="required"/>
</complexType>
</element>
</schema>
```

C.12 Syntax of the DADX file

In Figure C-32, the numbers next to the nodes and elements in DADX syntax definitions 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.

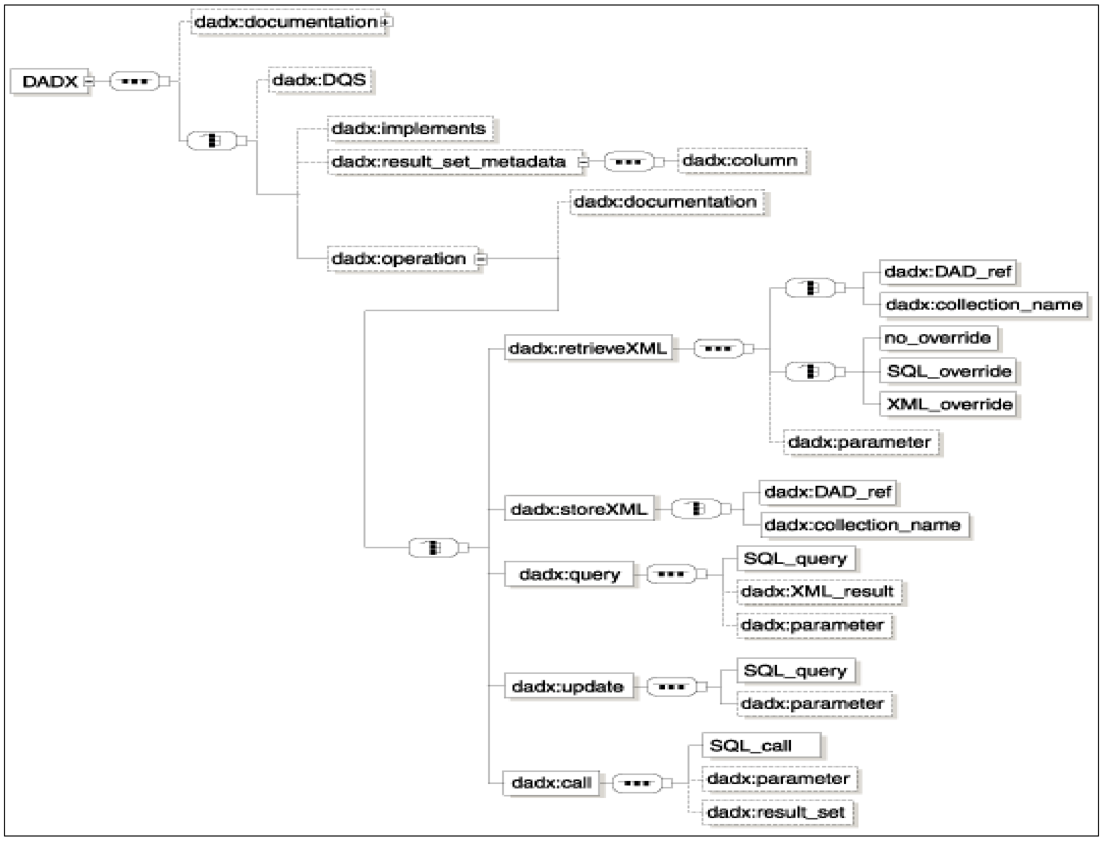


Figure C-32   Syntax of the DADX file

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.

C.13 Dynamic query service operations in the Web services provider

Table C-8 describes supported dynamic query operations in the DB2 Web services provider.

Table C-8 Operations for metadata retrieval

Web service operation	Description
getTables **tablesInputParameter** input; type = tablesInputData (See Table C-12) **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 C-13) **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 C-9 shows the operations to run queries.

Table C-9 Operations to run queries and stored procedures

Operations	Description
executeQuery **queryInputParameter** required input; type = string **extendedInputParameter** required input; type = properties (See Table C-10) **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 C-10) **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 C-10) **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 C-10) **executeOutputParameter** output; type = executeOutputData	Issues a single SQL statement on the database server and returns a completion code and a sequence of result sets.

Table C-10 shows the input data types for the extended parameters.

Table C-10 Input data types for the extended parameters

Properties type	Description
loginInfo (userid and 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.
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 C-11 on page 661 shows the input data types for callInputParameter.

Table C-11 Input data types for 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.

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 C-8 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 C-12 shows the input data types for the tablesInputData type.

Table C-12 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 C-13 shows the input data types for columnsInputData types.

Table C-13 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 C-14 shows the output data types for the callOutputData types.

Table C-14 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"/>

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 C-15 shows the output data types for the executeOutputData types.

Table C-15 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>
```

D

Setting up IMS services

In Chapter 8, “IMS and SOA” on page 201 we use a simple application “customer query”. This application uses one DLI database (HDAM), composed of one root segment. The wizards exploited during the generation of the SOA elements, use “c includes” and “cobol copybooks” representing the layouts of the input/output messages.

The message program (MPP) is written in Java and has to run in a Java Message Processing (JMP).

In this appendix we describe the different steps and objects involved in setting up this application on the z/OS system.

© Copyright IBM Corp. 2006. All rights reserved.

D.1 Database

We list the definitions of the database objects.

D.1.1 Database descriptor

Example D-1 contains the DBD source, the database description of DLI HDAM database SJCUSTDB with only a root segment.

Example: D-1 DBD for HDAM database SJCUSTDB

```
DBD     NAME=SJCUSTDB,ACCESS=HDAM,RMNAME=(DFSHDC40,40,100)
DATASET DD1=SJCUSDB,DEVICE=3380,SIZE=4096
SEGM    NAME=CUSTINFO,PARENT=0,BYTES=300,RULES=(LLL,LAST)
FIELD   NAME=(CUSTNUM,SEQ,U),BYTES=010,START=00001,TYPE=C
FIELD   NAME=SSN,BYTES=011,START=00011,TYPE=C
FIELD   NAME=FIRSTNME,BYTES=020,START=022,TYPE=C
FIELD   NAME=MI,BYTES=002,START=042,TYPE=C
FIELD   NAME=LASTNAME,BYTES=040,START=044,TYPE=C
FIELD   NAME=SALUTAT,BYTES=010,START=84,TYPE=C
FIELD   NAME=ADDRESS1,BYTES=040,START=94,TYPE=C
FIELD   NAME=ADDRESS2,BYTES=040,START=134,TYPE=C
FIELD   NAME=CITY,BYTES=030,START=174,TYPE=C
FIELD   NAME=STATE,BYTES=004,START=204,TYPE=C
FIELD   NAME=ZIPCD,BYTES=010,START=208,TYPE=C
FIELD   NAME=PHONE,BYTES=020,START=218,TYPE=C
FIELD   NAME=FAX,BYTES=020,START=238,TYPE=C
FIELD   NAME=EMLADDR,BYTES=040,START=258,TYPE=C
DBDGEN
FINISH
END
```

Example D-2 contains the DBD Generation JCL.

Example: D-2 Asm/Lked for DBD generation

```
//VANAERS1 JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//*******************************************************************
//* FUNCTION: DBDGEN FOR THE HDAM/VSAM DATA BASE
//*******************************************************************
//CUSDB2   EXEC PROC=DBDGEN,MBR=SJCUSTDB,SOUT='*'
//C.SYSIN  DD DISP=SHR,
//            DSN=VANAERS.IMS.DATA(SJCUSTDB)
//*
```

D.1.2 Database load

Example D-3 shows the allocation of VSAM file for the database.

Example: D-3   Allocation of VSAM cluster for Database

```
//VANAERS2 JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//*******************************************************************
//* FUNCTION: ALLOCATE DATA SETS NEEDED FOR THE DB SJSAMPLE
//**********************************************************âSCPYRT**
//* SCRATCH DATA SETS
//*
//SCRATCH EXEC PGM=IDCAMS,DYNAMNBR=200
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DELETE IMS910H.SJCUSDB CLUSTER
//*-----------------------------------------------
//* ALLOCATE DATA SETS
//*
//ALLOCATE EXEC PGM=IDCAMS,DYNAMNBR=200
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER(                               -
                NAME(IMS910H.SJCUSDB)           -
                NONINDEXED                      -
                FREESPACE(10 10)                -
                RECORDSIZE(4089 4089)           -
                SHAREOPTIONS(3 3)               -
                UNIQUE                          -
                VOLUMES(SBOXI4)                 -
                CYLINDERS(20)                   -
                CONTROLINTERVALSIZE(4096)       -
               )                                -
          DATA(                                 -
                NAME(IMS910H.SJCUSDB.DATA)      -
               )
//*
```

Example D-4 shows the generation of Dynamic Allocation member.

Example: D-4   IMSDALLOC for database

```
//VANAERS3 JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//*******************************************************************
//* FUNCTION: CREATE DYNAMIC ALLOCATION MEMBERS
//**********************************************************âECPYRT**
//*
//STEP01   EXEC PROC=IMSDALOC,SOUT='*'
//ASSEM.SYSIN DD *
```

```
*
* START
*
  DFSMDA TYPE=INITIAL
*
* SJSAMPLE DATABASE
*
* HDAM/VSAM
*
  DFSMDA TYPE=DATABASE,DBNAME=SJCUSTDB
  DFSMDA TYPE=DATASET,DDNAME=SJCUSDB,                                  X
               DSNAME=IMS91OH.SJCUSTDB,                                X
               DISP=SHR
*
* END
*
  DFSMDA TYPE=FINAL
          END
/*
//*
```

Example D-5 lists the input data to DFSDDLT0.

Example: D-5   Short sample(3 records) input data used by DFSDDLT0 utility

```
L        ISRT  CUSTINFO                                                 00000000
L        DATA  0000000000*123456789**123456789012345678****123456789012X00000001
               345678901234567890123456 78**12345678**123456789012345678X00000002
               901234567890123456 78**1234567890123456789012345678901234X00000003
               5678**12345678901234567890123456 78**12**12345678**123456X00000004
               789012345678**1234567890123456789**1234567890123456 78901X00000005
               23456789012345678*
L        ISRT  CUSTINFO                                                 00000010
L        DATA  F006668   PSSC       EGIDE              LPVAN AERSCHOT X00000011
                                           MR        IBM FRANCE         X00000012
                                    RUE DE LA VIELLE POSTE              X00000013
                    MONTPELIER                    ----34006     33/[0]4X00000014
               67344978     33/[0]467341221      van_aerschot_egide@be.X00000015
               ibm.com                                                  00000016
L        ISRT  CUSTINFO                                                 00000020
L        DATA  F010000   PSSC       PIERRE             XXGAL           X00000021
                                           MR        IBM FRANCE         X00000022
                                    RUE DE LA VIELLE POSTE              X00000023
                    MONTPELIER                    ----34006     33/[0]4X00000024
               67344960     33/[0]467341221      pierre_gal@fr.ibm.com X00000025
                                                                        00000026
```

Example D-6 on page 668 shows the Load/List Execution of DFSDDLT0.

Example: D-6   Load with DFSDDLT0

```
//VANAERSL JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//* THE FOLLOWING STEP LOADS THE HDAM/VSAM DATA BASE
//LOAD     EXEC PROC=DLIBATCH,SOUT='*',
//         MBR=DFSDDLTO,PSB=SJCUSPSL,
//         DBRC=N,IRLM=N
//G.IEFRDER  DD DUMMY,UNIT=3390,DCB=BLKSIZE=6144
//G.IEFRDER2 DD DUMMY,UNIT=3390,DCB=BLKSIZE=6144
//G.SYSIN    DD DSN=VANAERS.IMS.DATA(SJLODATA),DISP=SHR
//G.SJCUSDB  DD DISP=OLD,DSN=IMS910H.SJCUSDB
//G.PRINTDD  DD SYSOUT=*
//G.DFSVSAMP DD *
VSRBF=4096,5
//READ     EXEC PROC=DLIBATCH,SOUT='*',
//         MBR=DFSDDLTO,PSB=SJCUSPSG,
//         DBRC=N,IRLM=N
//G.IEFRDER  DD DUMMY,UNIT=3390,DCB=BLKSIZE=6144
//G.IEFRDER2 DD DUMMY,UNIT=3390,DCB=BLKSIZE=6144
//G.SYSIN    DD DSN=VANAERS.IMS.DATA(SJRDDATA),DISP=SHR
//G.SJCUSDB  DD DISP=OLD,DSN=IMS910H.SJCUSDB
//G.PRINTDD  DD SYSOUT=*
//G.DFSVSAMP DD *
VSRBF=4096,5
//*
```

D.2 Programs and PSBs

We list the definitions of the application objects.

D.2.1 PSBs

Example D-7 shows the Load PSB.

Example: D-7 Load PSB for Java

```
***********************************************************************
*          NAME:  SJCUSPSL                                            *
*                                                                     *
*   DESCRIPTION:  PSB FOR HDAM/VSAM DATA BASE LOAD                    *
*****************************************************************äSCPYRT**
SJCUSTL  PCB   TYPE=DB,DBDNAME=SJCUSTDB,PROCOPT=L,KEYLEN=10
         SENSEG NAME=CUSTINFO,PARENT=0
         PSBGEN LANG=ASSEM,PSBNAME=SJCUSPSL
         END
```

Example D-8 shows the PSB for JAVA Message Processing Program.

Example: D-8 PSB for Java MPP(JMP)

```
***********************************************************************
*         NAME:  SJCUSPSJ                                             *
*                                                                     *
*   DESCRIPTION:  PSB FOR JMP PROGRAM (JAVA PROGRAM)                  *
**************************************************************äSCPYRT**
SJCUSTJ  PCB   TYPE=DB,DBDNAME=SJCUSTDB,PROCOPT=A,KEYLEN=10
         SENSEG NAME=CUSTINFO,PARENT=0
         PSBGEN LANG=JAVA,PSBNAME=SJCUSPSJ,COMPAT=YES
         END
```

The following two examples show the HDAM PSBs.

Example D-9 shows the HDAM retrieve PSB.

Example: D-9 PSB for HDAM retrieve

```
***********************************************************************
*         NAME:  SJCUSPSG                                             *
*                                                                     *
*   DESCRIPTION:  PSB FOR HDAM/VSAM DATA BASE RETRIEVE                *
**************************************************************äSCPYRT**
SJCUSTG  PCB   TYPE=DB,DBDNAME=SJCUSTDB,PROCOPT=G,KEYLEN=10
         SENSEG NAME=CUSTINFO,PARENT=0
         PSBGEN LANG=ASSEM,PSBNAME=SJCUSPSG
         END
```

Example D-10 shows the HDAM update PSB.

Example: D-10 PSB for HDAM update

```
***********************************************************************
*         NAME:  SJCUSPSA                                             *
*                                                                     *
*   DESCRIPTION:  PSB FOR HDAM/VSAM DATA BASE UPDATE                  *
**************************************************************äSCPYRT**
SJCUSTA  PCB   TYPE=DB,DBDNAME=SJCUSTDB,PROCOPT=A,KEYLEN=10
         SENSEG NAME=CUSTINFO,PARENT=0
         PSBGEN LANG=ASSEM,PSBNAME=SJCUSPSA
         END
```

Example D-11 shows the PSB Generation JCL.

Example: D-11 PSBgen

```
//VANAERS4 JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//*********************************************************************
//* FUNCTION: PERFORM PSBGEN'S
//*********************************************************************
```

```
//*  LICENSED MATERIALS - PROPERTY OF IBM                           *
//*  5655-J38                                                       *
//*  (C) COPYRIGHT IBM CORP. 1989,1998 ALL RIGHTS RESERVED          *
//*  US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR    *
//*  DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH        *
//*  IBM CORP.                                                      *
//***********************************************************âECPYRT**
//* PSBGEN FOR LOADING SJSAMPLE DATABASE
//*
//SJPSBGNL EXEC PROC=PSBGEN,MBR=SJCUSPSL,SOUT='*'
//C.SYSIN  DD DISP=SHR,
//            DSN=VANAERS.IMS.DATA(SJCUSPSL)
//*---------------------------------------------
//* PSBGEN FOR LISTING SJSAMPLE DATABASE
//*
//SJPSBGNG EXEC PROC=PSBGEN,MBR=SJCUSPSG,SOUT='*'
//C.SYSIN  DD DISP=SHR,
//            DSN=VANAERS.IMS.DATA(SJCUSPSG)
//*---------------------------------------------
//* PSBGEN FOR UPDATING SJSAMPLE DATABASE
//*
//SJPSBGNA EXEC PROC=PSBGEN,MBR=SJCUSPSA,SOUT='*'
//C.SYSIN  DD DISP=SHR,
//            DSN=VANAERS.IMS.DATA(SJCUSPSA)
//*---------------------------------------------
//* PSBGEN FOR JAVA JMP PROGRAM
//*
//SJPSBGNA EXEC PROC=PSBGEN,MBR=SJCUSPSJ,SOUT='*'
//C.SYSIN  DD DISP=SHR,
//            DSN=VANAERS.IMS.DATA(SJCUSPSJ)
//*
```

Example D-12 shows the ACBgen for IMS online.

Example: D-12 ACBgen

```
//VANAERSA JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=64M
//*
/*JOBPARM L=9999,SYSAFF=*
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*
//***********************************************************âECPYRT**
//*
//ACBGEN   EXEC PROC=ACBGEN,SOUT='*',COMP='POSTCOMP'
//G.SYSIN  DD *
  BUILD PSB=SJCUSPSL
  BUILD PSB=SJCUSPSA
  BUILD PSB=SJCUSPSJ
  BUILD PSB=SJCUSPSG
//*
```

D.2.2 JDBC access to DLI data

Here we list the definitions of objects related to the JDBC access to DLI data.

Example D-13 shows the dDliModel Utility JCL.

Example: D-13 SJDLIMOD

```
//VANAERSD JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=OM
//*
/*JOBPARM L=9999,SYSAFF=*
//*
//*****************************************************************
//DLIMODEL PROC ABSPATH=,DSNAME=,SOUT=*
//*****************************************************************
//* THIS PROC RUNS THE IMS JAVA UTILITY IN BATCH MODE
//*****************************************************************
//STEP1 EXEC PGM=BPXBATCH,PARM='SH cd &ABSPATH;go "&DSNAME" PDS'
//STDENV DD DUMMY
//STDOUT DD PATH='/tmp/&SYSUID..out',
//  PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//  PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/&SYSUID..err',
//  PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//  PATHMODE=SIRWXU
//*---------------------------------------------
//* Redirect stdout and stderr output to SYSOUT:
//STEP2 EXEC PGM=IKJEFT01,DYNAMNBR=300,COND=EVEN
//SYSTSPRT DD SYSOUT=&SOUT
//HFSOUT DD PATH='/tmp/&SYSUID..out'
//HFSERR DD PATH='/tmp/&SYSUID..err'
//STDOUTL DD SYSOUT=&SOUT,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//STDERRL DD SYSOUT=&SOUT,DCB=(RECFM=VB,LRECL=133,BLKSIZE=137)
//SYSPRINT DD SYSOUT=&SOUT
// PEND
//SJDLI EXEC DLIMODEL,DSNAME='VANAERS.IMS.DATA(SJDLMOCN)',
//   ABSPATH='/usr/lpp/imsv9/imsjava91/dlimodel'
//STEP2.SYSTSIN DD *
  OCOPY INDD(HFSOUT) OUTDD(STDOUTL)
  OCOPY INDD(HFSERR) OUTDD(STDERRL)
```

Example D-14 shows the DliModel Utility input.

Example: D-14 SJDLMOCN, input for DLIModel Utility

```
OPTIONS PSBds=VANAERS.IMS.DATA
        DBDds=VANAERS.IMS.DATA
        GenJavaSource
        GenTrace
        OutPath=/u/vanaers/sjcusJMP
        Package=dlicust
PSB     PSBName=SJCUSPSJ JavaName=SJCUSTDatabaseView
PCB     PCBName=SJCUSTJ JavaName=CustInfo
SEGM    DBDName=SJCUSTDB SegmentName=CUSTINFO JavaName=Customer
```

```
FIELD   Name=CUSTNR Start=1 Bytes=10 JavaName=custnr JavaType=CHAR
FIELD   Name=SSN Start=11 Bytes=11 JavaName=ssn JavaType=CHAR
FIELD   Name=FIRSTNME Start=22 Bytes=20 JavaName=firstnme JavaType=CHAR
FIELD   Name=MI Start=42 Bytes=02 JavaName=mi JavaType=CHAR
FIELD   Name=LASTNAME Start=44 Bytes=40 JavaName=lastname JavaType=CHAR
FIELD   Name=SALUTAT Start=84 Bytes=10 JavaName=salutat JavaType=CHAR
FIELD   Name=ADDRESS1 Start=94 Bytes=40 JavaName=address1 JavaType=CHAR
FIELD   Name=ADDRESS2 Start=134 Bytes=40 JavaName=address2 JavaType=CHAR
FIELD   Name=CITY Start=174 Bytes=30 JavaName=city JavaType=CHAR
FIELD   Name=STATE Start=204 Bytes=04 JavaName=state JavaType=CHAR
FIELD   Name=ZIPCD Start=208 Bytes=10 JavaName=zipcd JavaType=CHAR
FIELD   Name=PHONE Start=218 Bytes=20 JavaName=phone JavaType=CHAR
FIELD   Name=FAX Start=238 Bytes=20 JavaName=fax JavaType=CHAR
FIELD   Name=EMLADDR Start=258 Bytes=40 JavaName=emladdr JavaType=CHAR
```

Example D-15 shows the DatabaseView class generated by the DliModel utility for the PSB SJCUSPSJ used with the JMP Java program.

Example: D-15 java class dlicust.SJCUSTDLIDatabaseView

```
package dlicust;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class SJCUSTDatabaseView extends DLIDatabaseView {

    // This class describes the data view of PSB: SJCUSPSJ
    // PSB SJCUSPSJ has database PCBs with 8-char PCBNAME or label:
    //       SJCUSTJ

    // The following describes Segment: CUSTINFO ("Customer") in PCB: SJCUSTJ
("CustInfo")
    static DLITypeInfo[] SJCUSTJCUSTINFOArray= {
        new DLITypeInfo("custnr", DLITypeInfo.CHAR, 1, 10, "CUSTNR",
DLITypeInfo.UNIQUE_KEY),
        new DLITypeInfo("ssn", DLITypeInfo.CHAR, 11, 11, "SSN"),
        new DLITypeInfo("firstnme", DLITypeInfo.CHAR, 22, 20, "FIRSTNME"),
        new DLITypeInfo("mi", DLITypeInfo.CHAR, 42, 2, "MI"),
        new DLITypeInfo("lastname", DLITypeInfo.CHAR, 44, 40, "LASTNAME"),
        new DLITypeInfo("salutat", DLITypeInfo.CHAR, 84, 10, "SALUTAT"),
        new DLITypeInfo("address1", DLITypeInfo.CHAR, 94, 40, "ADDRESS1"),
        new DLITypeInfo("address2", DLITypeInfo.CHAR, 134, 40, "ADDRESS2"),
        new DLITypeInfo("city", DLITypeInfo.CHAR, 174, 30, "CITY"),
        new DLITypeInfo("state", DLITypeInfo.CHAR, 204, 4, "STATE"),
        new DLITypeInfo("zipcd", DLITypeInfo.CHAR, 208, 10, "ZIPCD"),
        new DLITypeInfo("phone", DLITypeInfo.CHAR, 218, 20, "PHONE"),
        new DLITypeInfo("fax", DLITypeInfo.CHAR, 238, 20, "FAX"),
        new DLITypeInfo("emladdr", DLITypeInfo.CHAR, 258, 40, "EMLADDR")
    };
    static DLISegment SJCUSTJCUSTINFOSegment= new DLISegment
        ("Customer","CUSTINFO",SJCUSTJCUSTINFOArray,300);

    // An array of DLISegmentInfo objects follows to describe the view for PCB:
SJCUSTJ ("CustInfo")
```

```
    static DLISegmentInfo[] SJCUSTJarray = {
        new DLISegmentInfo(SJCUSTJCUSTINFOSegment,DLIDatabaseView.ROOT)
    };


    // Constructor
    public SJCUSTDatabaseView() {
        super("2.0","SJCUSPSJ", "CustInfo", "SJCUSTJ", SJCUSTJarray);
    } // end SJCUSTDatabaseView constructor

} // end SJCUSTDatabaseView class definition
```

Example D-16 shows the Report generated by the DliModel utility.

Example: D-16 SJCUSTDatabaseViewJavaReport

```
DLIModel IMS Java Report
========================
Class: SJCUSTDatabaseView  in package: dlicust  generated for PSB: SJCUSPSJ

==================================================
PCB: CustInfo
==================================================
Segment: Customer
Field: custnr    Type=CHARLength=10++ Primary Key Field ++
Field: ssn       Type=CHARLength=11     (Search Field)
Field: firstnme  Type=CHARLength=20     (Search Field)
Field: mi        Type=CHARLength=2     (Search Field)
Field: lastname  Type=CHARLength=40     (Search Field)
Field: salutat   Type=CHARLength=10     (Search Field)
Field: address1  Type=CHARLength=40     (Search Field)
Field: address2  Type=CHARLength=40     (Search Field)
Field: city      Type=CHARLength=30     (Search Field)
Field: state     Type=CHARLength=4     (Search Field)
Field: zipcd     Type=CHARLength=10     (Search Field)
Field: phone     Type=CHARLength=20     (Search Field)
Field: fax       Type=CHARLength=20     (Search Field)
Field: emladdr   Type=CHARLength=40     (Search Field)
```

D.2.3 Java Message Processing Program preparation

Here we list the definitions for JMPP preparation.

Example D-17 shows the layout of input/output message for C language, it includes file “inoutcust4rad.h” for C language.

Example: D-17 inoutcust4rad.h

```
struct INDATA {
   short     llin;
   char      z1;
   char      z2;
   char      tran[8];       // transaction code
   char      custnr[10];    // customernr
   };
struct OUTDATA {
   short     llout;
   char      z1;
   char      z2;
   char      custnr[10];    // customernr
   char      ssn[11];       //
   char      firstnme[20];  // first name
   char      mi;            // middle name
   char      lastname[40];  // last name
   char      salutat[10];   // salutation
   char      address1[40];  // address 1
   char      address2[40];  // address 2
   char      city[30];      // city
   char      state[4];      // state/department/kreiz
   char      zipcd[10];     // zip code/ postcode
   char      phone[20];     // phone
   char      fax[20];       // fax
   char      emladdr[40];   // email adddress
   char      message[200];  // message
   };
```

Example D-18 shows the Layout input/output message for Cobol language. It includes the Cobol copybook file “inoutcust4rad.cbl” for Cobol language.

Example: D-18   inoutcust4rad.cb

```
IDENTIFICATION DIVISION.
        program-id. pgm1.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        DATA DIVISION.
      *
      *    IMS Connector for Java, COBOL Transaction Message Source
      *
LINKAGE SECTION.

        01  INDATA.
            02  IN-LL          PICTURE S9(3) COMP.
            02  IN-ZZ          PICTURE S9(3) COMP.
            02  IN-TRCD        PICTURE X(8).
            02  CUSTOMERNR     PICTURE X(10).
            02  TRACEX         PICTURE X(1).

        01  OUTDATA.
            02  OUT-LL         PICTURE S9(3) COMP VALUE +0.
            02  OUT-ZZ         PICTURE S9(3) COMP VALUE +0.
            02  CUSTOMERNR     PICTURE X(10).
            02  SSN            PICTURE X(11) VALUE SPACES.
            02  FIRSTNME       PICTURE X(20) VALUE SPACES.
            02  MI             PICTURE X(2) VALUE SPACES.
            02  LASTNAME       PICTURE X(40) VALUE SPACES.
```

```
           02  SALUTAT        PICTURE X(10) VALUE SPACES.
           02  ADDRESS1       PICTURE X(40) VALUE SPACES.
           02  ADDRESS2       PICTURE X(40) VALUE SPACES.
           02  CITY           PICTURE X(30) VALUE SPACES.
           02  STATE          PICTURE X(4) VALUE SPACES.
           02  ZIPCD          PICTURE X(10) VALUE SPACES.
           02  PHONE          PICTURE X(20) VALUE SPACES.
           02  FAX            PICTURE X(20) VALUE SPACES.
           02  EMLADDR        PICTURE X(40) VALUE SPACES.
           02  MESSAGEX       PICTURE X(200) VALUE SPACES.

       PROCEDURE DIVISION.
```

Example D-19 shows the Input Message.

Example: D-19   java class imstmcust.INDATA

```
package imstmcust;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/**
 * @author vanaersc
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class INDATA extends IMSFieldMessage {
   /* Creates array of DLITypeInfo objects for the fields in message */
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo( "custnr", DLITypeInfo.CHAR, 1, 10),
      new DLITypeInfo( "trace", DLITypeInfo.CHAR,11, 1)
   };
   public INDATA() {
      super(fieldInfo, 10, false);
   }
}
```

Example D-20 shows the Output Message.

Example: D-20   java class imstmcust.OUTDATA

```
package imstmcust;

import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/**
 * @author vanaersc
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class OUTDATA extends IMSFieldMessage {
   /* Creates array of DLITypeInfo objects for the fields in message */
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo( "custnr", DLITypeInfo.CHAR, 1, 10) ,
      new DLITypeInfo( "ssn", DLITypeInfo.CHAR, 11, 11) ,
      new DLITypeInfo( "firstnme", DLITypeInfo.CHAR,22,20),
      new DLITypeInfo( "mi", DLITypeInfo.CHAR,42,2),
      new DLITypeInfo( "lastname", DLITypeInfo.CHAR,44,40),
      new DLITypeInfo( "salutat", DLITypeInfo.CHAR,84,10),
      new DLITypeInfo( "address1", DLITypeInfo.CHAR,94,40),
      new DLITypeInfo( "address2", DLITypeInfo.CHAR,134,40),
      new DLITypeInfo( "city", DLITypeInfo.CHAR,174,30),
      new DLITypeInfo( "state", DLITypeInfo.CHAR,204,4),
      new DLITypeInfo( "zipcd", DLITypeInfo.CHAR,208, 10),
      new DLITypeInfo( "phone", DLITypeInfo.CHAR,218, 20),
      new DLITypeInfo( "fax", DLITypeInfo.CHAR,238, 20),
      new DLITypeInfo( "emladdr", DLITypeInfo.CHAR,258,40),
      new DLITypeInfo( "message", DLITypeInfo.CHAR,298,200)
   };
   public OUTDATA() {
      super(fieldInfo, 646, false);
   }
}
```

Example D-21 shows the Java program for JMP.

Example: D-21 JMP program imstmcust.CQuery

```
package imstmcust;

import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Connection;

import com.ibm.ims.application.*;
import com.ibm.ims.base.DLITypeInfo;
import com.ibm.ims.base.IMSException;

import com.ibm.ims.db.*;

/**
 * @author vanaersc
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class CQuery {
   public static void main(String args[]) {
      IMSMessageQueue messageQueue = new IMSMessageQueue();
      INDATA inputMessage = new INDATA();
      OUTDATA outputMessage = new OUTDATA();
```

```
        Connection connection;
        boolean debugOn = false;
        try {
           Class.forName("com.ibm.ims.db.DLIDriver");
           System.out.print("CQuery_main making connection with DLI");
           connection = DriverManager.getConnection("jdbc:dli:"
                 + "dlicust.SJCUSTDatabaseView");
           System.out.print("CQuery_main connection OK");
           while (messageQueue.getUniqueMessage(inputMessage)) {
              if (inputMessage.getString("trace").equals("Y"))
                 debugOn = true;
              else
                 debugOn = false;
              /* for testing */
              debugOn = true;
              if (debugOn) {
                 System.out.print("CQuery_main custnr(" + inputMessage.getString("custnr") +
")");
                 System.out.print("            trace(" + inputMessage.getString("trace") +
")");
              }
              String custnr = inputMessage.getString("custnr").trim();
              if (!custnr.equals("")) {
                 if (getCustomerInfo(inputMessage, outputMessage,
                       connection, debugOn)) {
                    //
                 } else {
                    //
                 }
              } else {
                 outputMessage.setString("message",
                       " --wrong input for custnr --");
              }
              outputMessage.setString("custnr", custnr);
              if (debugOn) {
                 System.out.print("CQuery_main insert in queue");
              }
              messageQueue.insertMessage(outputMessage);
              if (debugOn) {
                 System.out.print("CQuery_main commit");
              }
              IMSTransaction.getTransaction().commit();
           }
        } catch (IMSException imsex) {
           System.err.print("CQuery_main imsex " + imsex.toString());
           try {
              outputMessage.setString("message", "CQuery_main imsex "
                    + imsex.toString());
              messageQueue.insertMessage(outputMessage);
              IMSTransaction.getTransaction().commit();
           } catch (Exception imsex2) {
              System.err.print("CQuery_main2 imsex " + imsex2.toString());
           }
        } catch (Exception ex) {
           System.err.print("CQuery_main ex " + ex.toString());
           try {
              outputMessage.setString("message", "CQuery_main ex "
                    + ex.toString());
              messageQueue.insertMessage(outputMessage);
              IMSTransaction.getTransaction().commit();
```

```
      } catch (Exception ex2) {
         System.err.print("CQuery_main2 ex " + ex2.toString());
      }
   }
   if (debugOn) {
      System.out.print("CQuery_main insert in queue");
   }
}

static boolean getCustomerInfo(INDATA inputMessage, OUTDATA outputMessage,
      Connection connection, boolean debugOn) {
   boolean oknok = true;
   try {
      String custnr = inputMessage.getString("custnr").trim();
      //        Parse the input message for ModelTypeCode
      String queryString = "SELECT * FROM SJCUSTJ.Customer WHERE Customer.custnr = '"
            + custnr + "’";
      // Create a statement and execute it to get a ResultSet
      Statement statement = connection.createStatement();
      if (debugOn) {
         System.out.print("CQuery_getCustomerInfo execute ("
               + queryString + ")");
      }

      ResultSet results = statement.executeQuery(queryString);
      // Send back the result of the query
      // Note: because "custnr" is unique - only 1 row is returned
      if (results.next()) {
         if (debugOn) {
            System.out
                  .print("CQuery_getCustomerInfo reading resultset");
         }
         outputMessage.setString("ssn", results.getString("ssn"));
         outputMessage.setString("firstnme", results
               .getString("firstnme"));
         outputMessage.setString("mi", results.getString("mi"));
         outputMessage.setString("lastname", results
               .getString("lastname"));
         outputMessage
               .setString("salutat", results.getString("salutat"));
         outputMessage.setString("address1", results
               .getString("address1"));
         outputMessage.setString("address2", results
               .getString("address2"));
         outputMessage.setString("city", results.getString("city"));
         outputMessage.setString("state", results.getString("state"));
         outputMessage.setString("zipcd", results.getString("zipcd"));
         outputMessage.setString("phone", results.getString("phone"));
         outputMessage.setString("fax", results.getString("fax"));
         outputMessage
               .setString("emladdr", results.getString("emladdr"));
         outputMessage.setString("message", " -- customer query OK --");
         return oknok;
      } else {
         outputMessage.setString("message", "Customer does NOT exist");
      }
      outputMessage.setString("message", " -- custnr OK --");
   } catch (IMSException imsex) {
      System.err
            .print("CQuery_getCustomerInfo imsex " + imsex.toString());
            try {
               outputMessage.setString("message",
                     "CQuery_getCustomerInfo imsex " + imsex.toString());
            } catch (Exception e) {
               //
            }
            oknok = false;
         } catch (Exception ex) {
            System.err.print("CQuery_getCustomerInfo ex " + ex.toString());
            try {
               outputMessage.setString("message", "CQuery_getCustomerInfo ex "
                     + ex.toString());
            } catch (Exception e) {
               //
            }
            oknok = false;
         }
         return true;
      }
}
```

D.3 Definitions for the application

Here we show the application definitions in IMS generation.

D.3.1 Database and application definitions in Stage 1

Add the DATABASE macro statements of Example D-22 to the IMS stage 1 input statements:

Example: D-22 DBDgen

```
DATABASE DBD=SJCUSTDB,ACCESS=UP ....
```

Add the APPLCTN macro statement of Example D-23 to the IMS stage 1 input statements for the sample application's program resource requirements.

Example: D-23 APPLCTN

```
APPLCTN PSB=SJCUSPSJ,PGMTYPE=TP,SCHEDTYP=PARALLEL
   TRANSACT CODE=CQUERY,PRTY=(7,10,2),INQUIRY=NO,MODE=SNGL, X
      MSGTYPE=(SNGLSEG,NONRESPONSE,1)
```

D.3.2 The JMP preparation

Example D-24 shows the allocation of HFS files err/out for JMP procedure.

Example: D-24 SJHFSJMP

```
//VANAERSH JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=0M
//*
/*JOBPARM L=9999,SYSAFF=*
//*
//*****************************************************
//* ALLOCATE HFS OUT/ERR files for JMP
//*****************************************************
```

```
//TCHMOD PROC TPARM=
//BPX EXEC PGM=BPXBATCH,PARM='&TPARM'
//SYSPRINT DD SYSOUT=*
//STDOUT DD PATH='/tmp/SJTCHMOD.OUT',
//   PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/SJTCHMOD.ERR',
//   PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
// PEND
//STEP1 EXEC TCHMOD,TPARM='sh touch /tmp/SJJVM.OUT'
//*
//STEP2 EXEC TCHMOD,TPARM='sh chmod 777 /tmp/SJJVM.OUT'
//*
//STEP3 EXEC TCHMOD,TPARM='sh touch /tmp/SJJVM.ERR'
//*
```

Example D-25 shows the master JVM options.

Example: D-25   DFSJVEMS member

```
***********************************************************ãECPYRT*****
*********************************************************************
* JVMOPMAS= member                                            ãPQ69696
*********************************************************************
*********************************************************************
* Replace SamplesPath with the absolute path of the directory  ãPQ69696
* containing samples.jar file                                  ãPQ69696
* If you are using SDK 1.4.1, add the following JVM property:
* -Djava.endorsed.dirs=pathprefix/usr/lpp/ims/imsjava91/lib
*********************************************************************
-Djava.endorsed.dirs=/imsv9/imsjava91/lib
****************************************************************ãPQ69696
-Dibm.jvm.shareable.application.class.path=>                  ãPQ69696
/imsv9/imsjava91/samples.jar                                   ãPQ69696
*
****************************************************************ãPQ69696
* Replace ImsjavaPath with the absolute path of the directory *ãPQ69696
* containing imsjava.jar file                                *ãPQ69696
****************************************************************ãPQ69696
-Dibm.jvm.trusted.middleware.class.path=>                      ãPQ69696
/imsv9/imsjava91/imsjava.jar                                   ãPQ69696
*                                                              ãPQ69696
****************************************************************ãPQ69696
-Dibm.jvm.trusted.middleware.class.path=>                      ãPQ69696
/u/vanaers/sjcusJMP/jars/sjsample.jar
*********************************************************************
* The following JVM options are a subset of the options allowed    *
* under JDK 1.3.1S                                                  *
*********************************************************************
-Xinitacsh128k
-Xinitsh128k
-Xmaxf0.6
-Xminf0.3
-Xmx64M
-Xoss400k
```

Example D-26 shows the environment member.

Example: D-26 DFSJVEEV member

```
**********************************************************************
* ENVIRON= member                                             âPQ69696
**********************************************************************
*  Change  JavaHome  to the SDK directory. For example:
*      /usr/lpp/java/j1.4
*  Change  imsjavaPath to
*      /usr/lpp/imsv9/imsjava91
**********************************************************************
* LIBPATH environment variable                                       *
* ------------------------------------------------------------âPQ69696
* Replace ImsjavaPath with the absolute path to libJavTDLI.so  âPQ69696
* Replace JavaHome with the absoulte path of Java installation âPQ69696
**********************************************************************
LIBPATH=/usr/lpp/java/J1.4/bin/classic:>
/usr/lpp/java/J1.4/bin:>
/usr/lpp/imsv9/imsjava91
```

Example D-27 shows the Worker JVM options.

Example: D-27 DFSJVEWK member

```
********************************************************************
* Sample JVMOPWKR= member                                          *
********************************************************************
********************************************************************
* The following JVM options are a subset of the options allowed    *
* under JDK 1.3.1S                                                 *
********************************************************************
-Xmaxf0.6
-Xminf0.3
-Xmx64M
-Xoss400k
```

Example D-28 shows the PSB to Class Mapping.

Example: D-28 DFSJVMAP member

```
**********************************************************************
* This is a mapping example for the IMS Java IVP samples which use   *
* the PSBs genned as:                                                *
*                                                                    *
* PSBGEN PSBNAME=DFSIVP37,LANG=JAVA                                  *
* PSBGEN PSBNAME=DFSIVP67,LANG=JAVA                                  *
* The IMS Java IVP samples are bundled inside the samples.jar file.  *
*                                                                    *
* This is the mapping for the IMS Java customer sample which uses    *
* the PSB  genned as:                                                *
* PSBGEN PSBNAME=SJCUSPSJ,LANG=JAVA                                  *
* The IMS Java Customer is bundled inside the sjsample.jar file.     *
*                                                                    *
* The location of this samples.jar, sjsamples.jar                    *
* must be specified separately                                       *
* by the DFSJVEMS member in the shareable application classpath.     *
*                                                                    *
**********************************************************************
*
DFSIVP37=samples/ivp/ims/IMSIVP
SJCUSPSJ=imstmcust/CQuery
*
**********************************************************************
```

Example D-29 shows the JMP Proc.

Example: D-29 JVMJMP Proc

```
//        PROC SOUT=A,RGN=0M,SYS2=,
//             CL1=001,CL2=000,CL3=000,CL4=000,
//             OPT=N,OVLA=0,SPIE=0,VALCK=0,TLIM=00,
//             PCB=000,STIMER=,SOD=,
//             NBA=,OBA=,IMSID=,AGN=,
//             PREINIT=,ALTID=,PWFI=N,APARM=,
//             LOCKMAX=,ENVIRON=,JVMOPWKR=,
//             JVMOPMAS=,XPLINK=N
//*
//JMPRGN EXEC PGM=DFSRRC00,REGION=&RGN,
//             TIME=1440,DPRTY=(12,0),
//             PARM=(JMP,&CL1&CL2&CL3&CL4,
//             &OPT&OVLA&SPIE&VALCK&TLIM&PCB,
//             &STIMER,&SOD,&NBA,
//             &OBA,&IMSID,&AGN,&PREINIT,
//             &ALTID,&PWFI,'&APARM',&LOCKMAX,
//             &ENVIRON,&JVMOPWKR,&JVMOPMAS,
//             &XPLINK)
//STEPLIB  DD DSN=IMS910H.&SYS2.PGMLIB,DISP=SHR
//         DD DSN=IMS910H.&SYS2.SDFSJLIB,DISP=SHR
//         DD DSN=IMS910H.&SYS2.SDFSRESL,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=SYS1.CSSLIB,DISP=SHR
//PROCLIB  DD DSN=IMS910H.&SYS2.PROCLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=&SOUT,
//         DCB=(LRECL=121,BLKSIZE=3129,RECFM=VBA),
//         SPACE=(125,(2500,100),RLSE,,ROUND)
```

Example D-30 shows the Java Message Region JCL.

Example: D-30 Java message region

```
//VANAERSM JOB (999,POK),'VANAERS',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// NOTIFY=VANAERS,REGION=0M
//*
//   JCLLIB ORDER=(IMS910H.PROCLIB)
/*JOBPARM L=9999,SYSAFF=*
//*     XPLINK=Y IF YOU USE SDK 1.4.1
//*     ENVIRON=
//*     JVMOPWKR=
//*     JVMOPMAS=
//*********************************************************************
//* MESSAGE JMP REGION (JAVA) FOR SJSAMPLE APPLICATION
//*********************************************************************
//  EXEC  DFSJMPE,IMSID=IMSH,
//    ENVIRON=DFSJVEEV,JVMOPWKR=DFSJVEWK,
//    JVMOPMAS=DFSJVEMS,XPLINK=Y
//JMPRGN.JAVAOUT DD PATH='/tmp/SJJVM.OUT'
//JMPRGN.JAVAERR DD PATH='/tmp/SJJVM.ERR'
```

E

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247259`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-7259.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg247259.zip	Zipped Code Samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	10 GB minimum
Operating System:	Windows
Processor:	1 Ghz or higher
Memory:	1.5 GB or higher[1]

[1] This configuration is suggested for running the examples with the IBM Rational Application Developer 6.0, DB2 Universal Database and WebSphere Application Server/WebSphere Portal Server.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. The following directories are contained in the ZIP archive:

PHP	The PHP Web Service example demonstrated at 16.5, “Access an enterprise application using PHP” on page 528.
RAD\ITSOBank databases	DDL to create the database objects accessed in the scenarios in Chapter 15, “Developing SOA access services” on page 401.
RAD\CreditCDE	RAD project containing CreditCDE dynamic Web application which is accessed in the scenario at 15.2, “Scenario using DB2 as Web Service consumer” on page 426.
RAD\CreditCDEApp	RAD project containing the enterprise application of the CreditCDE project.
RAD\WORFAccountDB	RAD project containing WORF/DADX dynamic Web application demonstrated at 15.1.2, “Implementation of the Web Services using WORF” on page 404.
RAD\WORFAccountDBApp	RAD project containing the enterprise application of the WORFAccountDB project.
RAD\JavaAccountDB	RAD project containing the dynamic Web application used in the scenario at 15.1.4, “Web Service implementation using Java wrappers” on page 417.
RAD\JavaAccountDBApp	RAD project containing the enterprise application of the JavaAccountDB project.
RAD\ITSOBankPortlets	RAD project containing the portlet application demonstrated at 15.4, “Scenario aggregating services as portlets” on page 455.
RAD\ITSOBankPortletsApp	RAD project containing the enterprise application of the ITSOBankPortlets project.

Usage of RAD projects

You can import the RAD projects into the IBM Rational Application Developer workspace by selecting **File** → **Import** → **Existing project into workspace** and entering the location of the project.

Setup of the example DB2 databases

This section leads through the steps required to setup the example DB2 databases DB2ACCTD and PORTALDB, both of which are used in the scenarios in Chapter 15, “Developing SOA access services” on page 401.

Create the DB2ACCTD database

The DB2ACCTD database is used to demonstrate the access to a DB2 for z/OS database. If you do not have a z/OS system available, you can create a DB2 for Windows database instead. The scenarios in which the DB2ACCTD database is used are platform-independent.

To create the database on DB2 for Windows, open the DB2 Control Center and select **Tools** → **Wizards** → **Create Database Wizard**. When you're asked for the database name, enter DB2ACCTD.

The database contains a few stored procedures which access some local tables. You can create these database objects by running the scripts contained in the Additional Materials. Open a DB2 Command window, change to the directory containing the scripts (ITSOBank databases\DB2ACCTD), and run the following commands:

```
db2 CONNECT TO DB2ACCTD
db2 -tf create_DB2ACCTD_tables.ddl
```

Create the stored procedures in the DB2 Development Center: Add a database connection to the local DB2ACCTD database in that tool, and import the sources of the stored procedures from the files create_DB2ACCTD_CRTMRACT.ddl, create_DB2ACCTD_GETMRACT.ddl, and create_DB2ACCTD_LSTACNBR.ddl.

Create the PORTALDB database

The PORTALDB database is a DB2 for Linux, Unix and Windows database which is used in the portlet application example at 15.4, "Scenario aggregating services as portlets" on page 455. It contains the database objects of the FICO score portlet, and a few additional tables containing portal user and customer data.

Create the database on DB2 for Windows using the DB2 Control Center. Select **Tools** → **Wizards** → **Create Database Wizard** and enter PORTALDB as database name. At the region settings page, select the UTF-8 code set. When the database is created, right-click on it, and select **Web Services** → **Enable Web Services**. This action enables the DB2 XML Extender in this database, and creates the DB2 Web Services SOAP UDFs. The Web Services functionality is required by the credit score UDF which is created by the statements below.

Add the database objects by running the scripts contained in the Additional Materials. Open a DB2 Command window, change to the directory containing the scripts (ITSOBank databases\PORTALDB), and run the following commands:

```
db2 CONNECT TO PORTALDB
db2 -tf create_PORTAL_DAO_objects.ddl
db2 -tf populate_FICO_objects.ddl
```

Glossary

A

address space. A range of virtual storage pages identified by a number (ASID) and a collection of segment and page tables which map the virtual pages to real pages of the computer's memory.

address space connection. The result of connecting an allied address space to DB2. Each address space containing a task connected to DB2 has exactly one address space connection, even though more than one task control block (TCB) can be present. See allied address space and task control block.

allied address space. An area of storage external to DB2 that is connected to DB2 and is therefore capable of requesting DB2 services.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States.

ANSI. American National Standards Institute.

API See *Application Program Interface.*

applet See *Java Applet***.**

application (1) A program or set of programs that perform a task; for example, a payroll application. (2) In Java programming, a self-contained, stand-alone Java program that includes a static main method. It does not require an applet viewer. Contrast with applet.

application plan The control structure produced during the bind process and used by DB2 to process SQL statements encountered during statement execution.

application program interface (API) A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or licensed program.

application requester (AR) See *requester.*

AR application requester. See requester.

artifact A piece of digital information. An artifact may be any size, and may be composed of other artifacts. Examples of artifacts: a message; a URI; an XML document; a PNG image; a bit stream.

ASCII (1) American Standard Code for Information Interchange.A standard assignment of 7-bit numeric codes to characters. See also *Unicode*. (2) An encoding scheme used to represent strings in many environments, typically on PCs and workstations. Contrast with EBCDIC.

attachment facility An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

authorization ID A string that can be verified for connection to DB2 and to which a set of privileges are allowed. It can represent an individual, an organizational group, or a function, but DB2 does not determine this representation.

B

base table (1) A table created by the SQL CREATE TABLE statement that is used to hold persistent data. Contrast with result table and temporary table. (2) A table containing a LOB column definition. The actual LOB column data is not stored along with the base table. The base table contains a row identifier for each row and an indicator column for each of its LOB columns. Contrast with auxiliary table.

bean A definition or instance of a JavaBeans component. See *JavaBeans*.

binary large object (BLOB) See BLOB.

bind The process by which the output from the DB2 precompiler is converted to a usable control structure called a package or an application plan. During the process, access paths to the data are selected and some authorization checking is performed.

automatic bind. (More correctly automatic rebind). A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

dynamic bind. A process by which SQL statements are bound as they are entered.

incremental bind. A process by which SQL statements are bound during the execution of an application process, because they could not be bound during the bind process, and VALIDATE(RUN) was specified.

static bind. A process by which SQL statements are bound after they have been precompiled. All static SQL statements are prepared for execution at the same time. Contrast with dynamic bind.

binding The relationship between a service provider and consumer is dynamic; it is established at runtime by a binding mechanism.

BLOB A sequence of bytes, where the size of the sequence ranges from 0 bytes to 2 GB - 1. Such a string does not have an associated CCSID. The size of binary large object values can be anywhere up to 2 GB-1.

browser (1) In VisualAge for Java, a window that provides information on program elements. There are browsers for projects, packages, classes, methods, and interfaces. (2) An Internet-based too that lets users browse Web sites.

built-in function A function that is supplied by DB2. Contrast with user-defined function.

bytecode Machine-independent code generated by the Java compiler and executed by the Java interpreter.

C

CAF Call attachment facility.

call attachment facility (CAF) A DB2 attachment facility for application programs running in TSO or MVS™ batch. The CAF is an alternative to the DSN command processor and allows greater control over the execution environment.

call level interface (CLI) A callable application program interface (API) for database access, which is an alternative to using embedded SQL. In contrast to embedded SQL, DB2 CLI does not require the user to precompile or bind applications, but instead provides a standard set of functions to process SQL statements and related services at run time.

casting Explicitly converting an object or primitive's data type.

catalog In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

catalog table Any table in the DB2 catalog.

CGI The Common Gateway Interface (CGI) is a means of allowing a Web server to execute a program that you provide rather than to retrieve a file. A number of popular Web servers support the CGI. For some applications, for example, displaying information from a database, you must do more than simply retrieve an HTML document from a disk and send it to the Web browser. For such applications, the Web server has to call a program to generate the HTML to be displayed. The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

character large object (CLOB) See *CLOB*.

class An encapsulated collection of data and methods to operate on the data. A class may be instantiated to produce an object that is an instance of the class.

class hierarchy The relationships between classes that share a single inheritance. All Java classes inherit from the Object class.

class method Methods that apply to the class as a whole rather than its instances (also called a *static method*).

class path When running a program in VisualAge for Java, a list of directories and JAR files that contain resource files or Java classes that a program can load dynamically at run time. A program's class path is set in its Properties notebook.

CLASSPATH In your deployment environment, the environment variable keyword that specifies the directories in which to look for class and resource files.

class variable Variables that apply to the class as a whole rather than its instances (also called a *static field*).

CLI See *call level interface*.

client (1)A networked computer in which the IDE is connected to a repository on a team server. (2) See requester.

CLOB A sequence of bytes representing single-byte characters or a mixture of single and double-byte characters where the size can be up to 2 GB - 1. Although the size of character large object values can be anywhere up to 2 GB - 1, in general, they are used whenever a character string might exceed the limits of the VARCHAR type.

codebase An attribute of the <APPLET> tag that provides the relative path name for the classes. Use this attribute when your class files reside in a different directory than your HTML files.

column function An SQL operation that derives its result from a collection of values across one or more rows. Contrast with scalar function.

commit The operation that ends a unit of work by releasing locks so that the database changes made by that unit of work can be perceived by other processes.

Common Connector Framework In the Enterprise Access Builder, interface and class definitions that provide a consistent means of interacting with enterprise resources (for example, CICS and Encina® transactions) from any Java execution environment.

connection In the VisualAge for Java Visual Composition Editor, a visual link between two components that represents the relationship between the components. Each connection has a source, a target, and other properties.

connection handle The data object that contains information associated with a connection managed by DB2 CLI. This includes general status information, transaction status, and diagnostic information.

consumer Entity utilizing a Web service.

cookie (1) A small file stored on an individual's computer; this file allows a site to tag the browser with a unique identification. When a person visits a site, the site's server requests a unique ID from the person's browser. If this browser does not have an ID, the server delivers one. On the Wintel platform, the cookie is delivered to a file called'cookies.txt,' and on a Macintosh platform, it is delivered to 'MagicCookie.' Just as someone can track the origin of a phone call with Caller ID, companies can use cookies to track information about behavior. (2) Persistent data stored by the client in the Servlet Builder.

choreography A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.

consumer The function that consumes the result of a service supplied by a provider.

cursor A named control structure used by an application program to point to a row of interest within some set of rows, and to retrieve rows from the set, possibly making updates or deletions.

D

Data Access Bean In the VisualAge for Java Visual Composition Editor, a bean that accesses and manipulates the content of JDBC/ODBC-compliant relational databases.

Data Access Builder A VisualAge for Java Enterprise tool that generates beans to access and manipulate the content of JDBC/ODBC-compliant relational databases.

database management system (DBMS) A software system that controls the creation, organization, and modification of a database and access to the data stored within it.

data source A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver which supports the ODBC APIs. In the case of DB2 for OS/390, the data sources are always relational database managers.

DBCLOB A sequence of bytes representing double-byte characters where the size can be up to 2 gigabytes. Although the size of double-byte character large object values can be anywhere up to 2 gigabytes, in general, they are used whenever a double-byte character string might exceed the limits of the VARGRAPHIC type.

DBMS Database management system.

DB2 thread The DB2 structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to DB2 resources. and services.

discovery The act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.

distributed relational database architecture (DRDA) A connection protocol for distributed relational database processing that is used by IBM's relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems.

DLL See *dynamic link library.*

double-byte character large object (DBCLOB) See DBCLOB.

double precision A floating-point number that contains 64 bits. See also *single precision*.

DRDA Distributed relational database architecture.

dynamic link library A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously. The DLLs. Enterprise Access Builders also generate platform-specific DLLs for the workstation and OS/390 platforms.

dynamic SQL SQL statements that are prepared and executed within an application program while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather than being coded into the application program. The SQL statement can change several times during the application program's execution.

E

EBCDIC Extended binary coded decimal interchange code. An encoding scheme used to represent character data in the MVS, VM, VSE, and OS/400Ñ environments. Contrast with ASCII.

embeddedJava An API and application environment for high-volume embedded devices, such as mobile phones, pagers, process control, instrumentation, office peripherals, network routers and network switches. EmbeddedJava™ applications run on real-time operating systems and are optimized for the constraints of small-memory footprints and diverse visual displays.

embedded SQL SQL statements coded within an application program. See static SQL.

enclave In Language Environment for MVS & VM, an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit.

Enterprise Java Includes Enterprise JavaBeans as well as open API specifications for: database connectivity, naming and directory services, CORBA/IIOP interoperability, pure Java distributed computing, messaging services, managing system and network resources, and transaction services.

Enterprise JavaBeans A cross-platform component architecture for the development and deployment of multi-tier, distributed, scalable, object-oriented Java applications.

environment handle In DB2 ODBC, the data object that contains global information regarding the state of the application. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

exception An exception is an object that has caused some sort of new condition, such as an error. In Java, *throwing* an exception means passing that object to an interested party; a signal indicates what kind of condition has taken place. *Catching* an exception means receiving the sent object. *Handling* this exception usually means taking care of the problem after receiving the object, although it might mean doing nothing (which would be bad programming practice).

executable content Code that runs from within an HTML file (such as an applet).

extends A subclass or interface extends a class or interface if it add fields or methods, or overrides its methods. See also *derived type.*

external function A function for which the body is written in a programming language that takes scalar argument values and produces a scalar result for each invocation. Contrast with sourced function and built-in function.

Extranet In some cases intranets have connections to other independent intranets. An example would be one company connecting its intranet to the intranet of one of its suppliers. Such a connection of intranets is called an extranet. Depending on the implementation, they may or may not be fully or partially visible to the outside.

F

factory A bean that dynamically creates instances of beans.

FastCGI FastCGI is a way of combining the advantages of CGI programming with some of the performance benefits you get by using the GWAPI. FastCGI, written by Open Market, Inc., is an extension to normal Web server processing. It requires server-specific API support, which is available for AIX, Sun Solaris, HP-UX, and OS/390. With FastCGI you can start applications in independent address spaces and pass requests for these applications from the Web server. The communication is through either the TCP/IP sockets interface or UNIX Domain socket bind path in the Hierarchical File System (HFS).

field A data object in a class; for example, a variable.

first tier The client; the hardware and software with which the end user interacts.

File Transfer Protocol (FTP) In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

foreign key A key that is specified in the definition of a referential constraint. Because of the foreign key, the table is a dependent table. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table.

form data A generated class representing the HTML form elements in a visual servlet.

FTP See *File Transfer Protocol*.

function A specific purpose of an entity or its characteristic action such as a column function or scalar function. (See column function and scalar function.). Furthermore, functions can be user-defined, built-in, or generated by DB2. (See built-in function, cast function, user-defined function, external function, sourced function.)

G

garbage collection Java's ability to clean up inaccessible unused memory areas ("garbage") on the fly. Garbage collection slows performance, but keeps the machine from running out of memory.

GWAPI Because CGI has some architectural limitations, most Web servers provide an equivalent mechanism that is optimized for their native environment. Domino Go Web Server, IBM′s strategic Web server, offers the Domino Go Web Server Application Programming Interface (GWAPI), optimized for a given environment, such as OS/390. The GWAPI enables you to create dynamic content similar to the CGI, but in a more specialized way than the generalized CGI. The GWAPI process is similar to OS/390 exit processing. There is an exit point for various server functions that can be exploited.

H

handle In DB2 CLI, a variable that refers to a data structure and associated resources. See statement handle, connection handle, and environment handle.

hierarchy The order of inheritance in object-oriented languages. Each class in the hierarchy inherits attributes and behavior from its superclass, except for the top-level Object class.

HPJ High Performance Java (HPJ) is a Java bytecode binder that generates extended text decks. These extended text decks can then be bound by an OS/390 binder into a dynamic link library (DLL) or program. The resulting OS/390 programs can reside in either an hierarchical file system (HFS) file or a partitioned data set extended (PDSE) (load library). The programs can be executed by OS/390 without the need for a JVM, thereby eliminating a large fraction of the performance overhead associated with JVM today.

HTML See *Hypertext Markup Language*

Hypertext Markup Language (HTML) A file format, based on SGML, for hypertext documents on the Internet. Allows for the embedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are embedded using URLs, enabling readers to jump directly to the referenced document.

Hypertext Transfer Protocol (HTTP) The Internet protocol, based on TCP/IP, used to fetch hypertext objects from remote hosts.

HTTPS HTTPS is a de facto standard developed by Netscape for making HTTP flows secure. Technically, it is the use of HTTP over SSL.

I

IDE See *Integrated Development Environment*.

Integrated Development Environment (IDE) In VisualAge for Java, the set of windows that provide the user with access to development tools. The primary windows are the Workbench, Log, Console, Debugger, and Repository Explorer.

Internet The vast collection of interconnected networks that use TCP/IP and evolved from the ARPANET of the late 1960s and early 1970s. The number of independent networks connected into this vast global net is growing daily.

Intranet A private network inside a company or organization that uses the same kinds of software that you would find on the Internet, but that are only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks, for example, many companies have Web servers that are available only to employees.

Internet Protocol (IP) In the Internet suite of protocols, a connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical network. However, this protocol does not provide error recovery and flow control and does not guarantee the reliability of the physical network.

interpreter A tool that translates and executes code line-by-line.

IP See *Internet Protocol*.

J

JAR file format JAR (Java Archive) is a platform-independent file format that aggregates many files into one. Multiple Java applets and their requisite components (.class files, images, sounds and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction.

Java An object-oriented programming language for portable, interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated. The Java environment consists of the JavaOS™, the Virtual Machines for various platforms, the object-oriented Java programming language, and several class libraries.

Java applet A small Java program designed to run within a Web browser. It is dowloadable and executable by a browser or network computer.

Java beans Java's component architecture, developed by Sun, IBM, and others. The components, called Java beans, can be parts of Java programs, or they can exist as self-contained applications. Java beans can be assembled to create complex applications, and they can run within other component architectures (such as ActiveX® and OpenDoc).

Java Development Kit (JDK) The Java Development Kit is the set of Java technologies made available to licensed developers by Sun Microsystems. Each release of the JDK contains the following: the Java Compiler, Java Virtual Machine, Java Class Libraries, Java Applet Viewer, Java Debugger, and other tools.

Java Naming and Directory Interface (JNDI) A set of APIs that assist with the interfacing to multiple naming and directory services. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Native Interface (JNI) A native programming interface that allows Java code running inside a Java Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C and C++.

Java Platform The Java Virtual Machine and the Java Core classes make up the Java Platform. The Java Platform provides a uniform programming interface to a 100%. Pure Java program regardless of the underlying operating system. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Remote Method Invocation (RMI) Java Remote Method Invocation is method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java. Included in JDK 1.1.

Java Runtime Environment (JRE) A subset of the Java Development Kit for end-users and developers who want to redistribute the JRE. The JRE consists of the Java Virtual Machine, the Java Core Classes, and supporting files. (Definition copyright 1996-1999 Sun Microsystems, Inc. All Rights Reserved. Used by permission.)

Java Virtual Machine (JVM) A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

JavaDoc Sun's tool for generating HTML documentation on classes by extracting comments from the Java source code files.

JavaScript A scripting language used within an HTML page. Superficially similar to Java but JavaScript scripts appear as text within the HTML page. Java applets, on the other hand, are programs written in the Java language and are called from within HTML pages or run as stand-alone applications.

Java servlet Servlets are similar to CGI programs, except that they are written in Java and run in a Java Virtual Machine managed by the Web server. Servlets are an effective substitute for CGI scripts because they provide an easier and faster way to generate dynamic documents. They also address the problem of doing server-side programming with platform-specific APIs because they are developed with the Java Servlet API, a standard Java extension. Servlets are modules that run inside Java-enabled Web servers and extend them in some manner. For example, a servlet might be responsible for validating the data in an HTML order-entry form. Servlets thus are a natural choice for choice for extending and enhancing Web servers.

JDBC (Java Database Connectivity) In the JDK, the specification that defines an API that enables programs to access databases that comply with this standard.

JIT See *Just-In-Time Compiler.*

JNDI See *Java Naming and Directory Interface.*

JNI See *Java Native Interface.*

JRE See *Java Runtime Environment*.

Just-In-Time compiler (JIT) A platform-specific software compiler often contained within JVMs. JITs compile Java bytecodes on-the-fly into native machine instructions, thereby reducing the need for interpretation.

JVM See *Java Virtual Machine*.

L

large object (LOB) See LOB.

link-edit To create a loadable computer program using a linkage editor.

linker A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses. In Java, the linker creates an executable from compiled classes.

load module A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.

LOB A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single and double-byte characters. A LOB can be up to 2 GB -1 byte in length. See also BLOB, CLOB, and DBCLOB.

M

method A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

middleware A layer of software that sits between a database client and a database server, making it easier for clients to connect to heterogeneous databases.

middle tier The hardware and software that resides between the client and the enterprise server resources and data. The software includes a Web server that receives requests from the client and invokes Java servlets to process these requests. The client communicates with the Web server via industry standard protocols such as HTTP and IIOP.

multithreading Multiple TCBs executing one copy of DB2 ODBC code concurrently (sharing a processor) or in parallel (on separate central processors).

MVS/ESA™ Multiple Virtual Storage/Enterprise Systems Architecture.

N

native class Machine-dependent C code that can be invoked from Java. For multi-platform work, the native routines for each platform need to be implemented.

null A special value that indicates the absence of information.

NUL-terminated host variable A varying-length host variable in which the end of the data is indicated by the presence of a NUL terminator.

NUL terminator In C, the value that indicates the end of a string. For character strings, the NUL terminator is X'00'.

O

object The principal building block of object-oriented programs. Objects are software programming modules. Each object is a programming unit consisting of related data and methods.

ODBC See Open Database Connectivity.

ODBC driver A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

Open Database Connectivity (ODBC) A Microsoft database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called database drivers that link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

P

package A program element that contains classes and interfaces.

persistence In object models, a condition that allows instances of classes to be stored externally, for example in a relational database.

Persistence Builder In VisualAge for Java, a persistence framework for object models, which enables the mapping of objects to information stored in relational databases and also provides linkages to earlier or existing data on other systems.

plan See application plan.

plan name The name of an application plan.

portal A single, secure point of access to diverse information, applications, and people that can be customized and personalized.

precompilation A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

prepare The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

prepared SQL statement A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

primary key A unique, non-null key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

process A program executing in its own address space, containing one or more threads.

property An initial setting or characteristic of a bean, for example, a name, font, text, or positional characteristic.

provider The function that performs a service in response to a request from a consumer

R

RDBMS Relational database management system.

relational database management system (RDBMS). A relational database manager that operates consistently across supported IBM systems.

reentrant Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. Re-entrancy is a compiler and operating system concept, and re-entrancy alone is not enough to guarantee logically consistent results when multithreading. See threadsafe.

reference An object's address. In Java, objects are passed by reference rather than by value or by pointers.

remote Refers to any object maintained by a remote DB2 subsystem; that is, by a DB2 subsystem other than the local one. A remote view, for instance, is a view maintained by a remote DB2 subsystem. Contrast with local.

Remote Method Invocation (RMI) RMI is a specific instance of the more general term RPC. RMI allows objects to be distributed over the network; that is, a Java program running on one computer can call the methods of an object running on another computer. RMI and java.net are the only 100% pure Java APIs for controlling Java objects in remote systems.

Remote Object Instance Manager In Remote Method Invocation, a program that creates and manages instances of server beans through their associated server-side server proxies.

Remote Procedure Calls (RPC) RPC is a generic term referring to any of a series of protocols used to execute procedure calls or method calls across a network. RPC allows a program running on one computer to call the services of a program running on another computer.

requester Also application requester (AR). The source of a request to a remote RDBMS, the system that requests the data.

RMI See *Remote Method Invocation.*

rollback The process of restoring data changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with commit.

RPC See *Remote Procedure Calls*.

runtime system The software environment where compiled programs run. Each Java runtime system includes an implementation of the Java Virtual Machine.

S

sandbox A restricted environment, provided by the Web browser, in which Java applets run. The sandbox offers them services and prevents them from doing anything naughty, such as doing file I/O or talking to strangers (servers other than the one from which the applet was loaded). The analogy of applets to children led to calling the environment in which they run the "sandbox."

scalar function An SQL operation that produces a single value from another value and is expressed as a function name followed by a list of arguments enclosed in parentheses. See also column function.

Secure Socket Layer (SSL) SSL is a security protocol which allows communications between a browser and a server to be encrypted and secure. SSL prevents eavesdropping, tampering or message forgery on your Internet or intranet network.

security Features in Java that prevent applets downloaded off the Web from deliberately or inadvertently doing damage. One such feature is the digital signature, which ensures that an applet came unmodified from a reputable source.

serialization Turning an object into a stream, and back again.

server The computer that hosts the Web page that contains an applet. The .class files that make up the applet, and the HTML files that reference the applet reside on the server. When someone on the Internet connects to a Web page that contains an applet, the server delivers the .class files over the Internet to the client that made the request. The server is also known as the originating host.

server bean The bean that is distributed using RMI services and is deployed on a server.

servlet See *Java servlet*.

service-oriented architecture A conceptual description of the structure of a software system in terms of its components and the services they provide, without regard for the underlying implementation of these components, services and connections between components.

SGML See *Standardized Generalized Markup Language.*

single precision A floating-point number that contains 32 bits. See also double precision.

SmartGuide In IBM software products, an active form of help that guides you through common tasks.

SOA See *service-oriented architecture*.

sourced function A function that is implemented by another built-in or user-defined function already known to the database manager. This function can be a scalar function or a column (aggregating) function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with external function and built-in function.

source type An existing type that is used to internally represent a distinct type.

SQL Structured Query Language. A language used by database engines and servers for data acquisition and definition.

SQL authorization ID (SQL ID) The authorization ID that is used for checking dynamic SQL statements in some situations.

SQL Communication Area (SQLCA) A structure used to provide an application program with information about the execution of its SQL statements.

SQL Descriptor Area (SQLDA) A structure that describes input variables, output variables, or the columns of a result table.

SQLCA SQL Communication Area.

SQLDA SQL Descriptor Area.

SSL See secure socket layer.

Standardized Generalized Markup Language An ISO/ANSI/ECMA standard that specifies a way to annotate text documents with information about types of sections of a document.

stateless Not depending on any pre-existing condition. In an SOA, services should not depend on the condition of any other service. They receive all information needed to provide a response from the request. Given the statelessness of services, service consumers can sequence (orchestrate) them into numerous flows (sometimes referred to as pipelines) to perform application logic.

statement handle In DB2 ODBC, the data object that contains information about an SQL statement that is managed by DB2 CLI. This includes information such as dynamic arguments, bindings for dynamic arguments and columns, cursor information, result values and status information. Each statement handle is associated with the connection handle.

static SQL SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of host variables specified by the statement might change).

stored procedure A user-written application program, that can be invoked through the use of the SQL CALL statement.

Structured Query Language (SQL) A standardized language for defining and manipulating data in a relational database.

T

table A named data object consisting of a specific number of columns and some number of unordered rows. Synonymous with base table or temporary table.

task control block (TCB) An MVS control block used to communicate information about tasks within an address space that are connected to DB2. An address space can support many task connections (as many as one per task), but only one address space connection. See address space connection.

TCB See *task control block*.

TCP/IP See *Transmission Control Protocol based on IP.*

Telnet Telnet provides a virtual terminal facility that allows users of one computer to act as though they were using a terminal connected to another computer. The Telnet client program communicates with the Telnet daemon on the target system to provide the connection and session.

temporary table A table created by the SQL CREATE GLOBAL TEMPORARY TABLE statement that is used to hold temporary data. Contrast with result table.

thin client Thin client usually refers to a system that runs on a resource-constrained machine or that runs a small operating system. Thin clients don't require local system administration, and they execute Java applications delivered over the network.

third tier The third tier, or back end, is the hardware and software that provides database and transactional services. These back-end services are accessed through connectors between the middle-tier Web server and the third-tier server. Though this conceptual model depicts the second and third tier as two separate machines, the NCF model supports a logical three-tier implementation in which the software on the middle and third tier are on the same box.

thread A separate flow of control within a program.

timestamp A seven-part value that consists of a date and time expressed in years, months, days, hours, minutes, seconds, and microseconds.

trace A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

transaction (1) In a CICS program, an event that queries or modifies a database that resides on a CICS server. (2) In the Persistence Builder, a representation of a path of code execution. (3) The code activity necessary to manipulate a persistent object. For example, a bank application might have a transaction that updates a company account.

Transmission Control Protocol based on IP (1) A network communication protocol used by computer systems to exchange information across telecommunication links. (2) An Internet protocol that provides for the reliable delivery of streams of data from one host to another.

type In VisualAge for Java, a generic term for a class or interface.

U

UDF User-defined function

UDT User-defined data type

Uniform Resource Locator (URL) The unique address that tells a browser how to find a specific Web page or file.

Unicode A 16-bit international character set defined by ISO 10646. See also ASCII.

user-defined data type (UDT) See distinct type.

user-defined function (UDF) A function defined to DB2 using the CREATE FUNCTION statement that can be referenced thereafter in SQL statements. A user-defined function can be either an external function or a sourced function. Contrast with built-in function.

URL See *Uniform Resource Locator.*

V

variable (1) An identifier that represents a data item whose value can be changed while the program is running. The values of a variable are restricted to a certain data type. (2)A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a variable. Contrast with constant.

virtual machine A software or hardware implementation of a central processing unit (CPU) that manages the resources of a machine and can run compiled code. See *Java Virtual Machine*.

visual bean In the Visual Composition Editor, a bean that is visible to the end user in the graphical user interface.

visual servlet A servlet that is designed to be built using the VisualAge for Java Visual Composition Editor.

VisualAge for Java, Enterprise Edition An edition of VisualAge for Java that is designed for building enterprise Java applications, and has all of the Professional Edition features plus support for developers working in large teams, developing high-performance or heterogeneous applications, or needing to connect Java programs to existing enterprise systems.

W

Web See World Wide Web

Web browser The Web uses a client/server processing model. The Web browser is the client component. Examples of Web browsers include Mosaic, Netscape Navigator, and Microsoft Internet Explorer. The Web browser is responsible for formatting and displaying information, interacting with the user, and invoking external functions, such as Telnet, or external viewers for data types that it does not directly support. Web browsers are fast becoming the universal client for the GUI workstation environment, in much the same way that the ability to emulate popular terminals such as the DEC VT100 or IBM 3270 allows connectivity and access to character-based applications on a wide variety of computers. Web browsers are available for all popular GUI workstation platforms and are inexpensive (often included with operating systems or related products for no additional charge.)

Web server Web servers are responsible for servicing requests for information from Web browsers. The information can be a file retrieved from the server¢ s local disk or generated by a program called by the server to perform a specific application function. Web servers are sometimes referred to as httpd servers or deamons. A number of Web servers are available for most platforms including most UNIX variants, OS/2® Warp, OS/390, and Windows NT. In addition, commercial Web servers that offer higher levels of vendor support and additional function are available. IBM has released the IBM Internet Connection Secure Server (ICSS) and its follow-on, the Domino Go Web server (DGW), for the AIX, OS/2 Warp, Windows NT, and OS/390 platforms.

Web services Web services is a set of standards meant to enable interoperable integration between heterogeneous information technology processes and systems.

Web Services Description Language The Web Services Description Language (WSDL) is an XML format published for describing Web services. It is commonly abbreviated as WSDL in technical literature and is usually pronounced wiz-dell. WSDL describes the public interface to the Web service. WSDL is often used in combination with SOAP and XML Schema to provide Web services over the internet. A client program connecting to a Web service can read the WSDL to determine what functions are available on the server. Any special datatypes used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

WebSphere WebSphere is the cornerstone of IBM's overall Web strategy, offering customers a comprehensive solution to build, deploy and manage e-business Web sites. The product line provides companies with an open, standards-based, Web server deployment platform and Web site development and management tools to help accelerate the process of moving to e-business.

wizards Web-based assistants.

World Wide Web A network of servers that contain programs and files. Many of the files contain hypertext links to other documents available through the network.

WSDL See *Web Services Description Language*

WWW See *World Wide Web*.

X

XML The Extensible Markup Language (XML) is an important new standard emerging for structured documents on the Web. XML extends HTML beyond a limited tag set and adapts SGML, making it easy for developers to write programs that process this markup and providing for a rich, more complex encoding of information. The importance of XML is indicated by support from many companies including IBM, Microsoft and Netscape.

Abbreviations and acronyms

AC	autonomic computing
AIX	advanced interactive eXecutive from IBM
APAR	authorized program analysis report
API	application programming interface
AR	application requester
AS	application server
ASCII	American National Standard Code for Information Interchange
B2B	business-to-business
BI	business intelligence
BPEL	Business Process Execution Language
CCI	common client interface
CCSID	coded character set identifier
CD	compact disk
CEC	central electronics complex
CGI	Common Gateway Interface
CICS	customer information control system
CLI	call level interface
CLP	command line processor
CMOS	complementary metal oxide semiconductor
CORBA	common object request broker architecture
CP	central processor
CPU	central processing unit
CSF	integrated cryptographic service facility
CUoD	Capacity Upgrade on Demand
DAD	document access definition
DASD	direct access storage device
DB	database
DB2	IBM Database 2™
DBA	database administrator
DBAT	database access thread
DBD	database descriptor
DBID	database identifier
DBM1	database master address space
DBRM	database request module
DCL	data control language
DDCS	distributed database connection services
DDF	distributed data facility
DDL	data definition language
DES	Data Encryption Standard
DML	data manipulation language
DNS	domain name server
DOM	document object module
DRDA	Distributed Relational Data Architecture
DSNZPARMs	DB2's system configuration parameters
EAR	enterprise application archive
DTD	document type definition
EAS	enterprise application solution
EBCDIC	extended binary coded decimal interchange code
EDM	environmental descriptor manager
EJB	Enterprise JavaBeans
ERP	enterprise resource planning
ESB	enterprise service bus
ESS	IBM TotalStorage Enterprise Storage Server®
EWLC	entry workload license charges
EWLM	Enterprise Workload Manager
FTP	File Transfer Program
GB	gigabyte (1,073,741,824 bytes)
HPJ	high performance Java
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HW	hardware
I/O	input/output
IBM	International Business Machines Corporation
ICSF	integrated cryptographic service facility
IFCID	instrumentation facility component identifier
IFI	instrumentation facility interface
IFL	integrated facility for Linux
IMS	Information Management System

IPLA	International Program Licence Agreement
ISPF	interactive system productivity facility
ISV	independent software vendor
IT	information technology
ITSO	International Technical Support Organization
IVP	installation verification process
J2C	J2EE Connector architecture
J2EE	Java 2 Platform Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAR	Java archive
JDBC	Java Database Connectivity
JFS	journaled file systems
JNDI	Java Naming and Directory Interface
JRE™	Java runtime environment
JSP	JavaServer Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
KB	kilobyte (1,024 bytes)
LDAP	Lightweight Directory Access Protocol
LPAR	logical partition
LRECL	logical record length
MB	megabyte (1,048,576 bytes)
MBps	megabytes per second
NALC	new application license charge
NFS	Network File System
OASIS	Organization for the Advancement of Structured Information Standards
ODBC	Open Database Connectivity
OLE	Object Linking and Embedding
OLTP	online transaction processing
PDS	partitioned data set
PHP	Hypertext Preprocessor
PTF	program temporary fix
RACF®	Resource Access Control Facility
RAS	reliability, availability and serviceability
RBA	relative byte address
RDBMS	relational database management system
RDS	relational data system
RECFM	record format
ROI	return on investment
RPO	recovery point objective
RRSAF	resource recovery services attach facility
SAN	storage area networks
SAX	simple API for XML
SCUBA	self contained underwater breathing apparatus
SDP	Software Development Platform
SLA	service-level agreement
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SQLJ	Structured Query Language for Java
SRM	Service Request Manager
SSL	Secure Sockets Layer
TCO	total cost of ownership
TPF	Transaction Processing Facility
UCB	unit control block
UDB	Universal Database
UDDI	Universal Description, Discovery and Integration
UDF	user-defined functions
UDT	user-defined (data) type
UOW	unit of work
UR	unit of recovery
URL	universal resource locator
USS	UNIX System Services
VIPA	Virtual IP Addressing
VM	virtual machine
VSIP	Visual Studio® Integrator Program
VWLC	variable workload license charges
WLC	Workload License Charge
WLM	Workload Manager
WSDL	Web Services Description Language
XML	Extensible Markup Language
WS-I	Web Services Interoperability Organization
WSIF	Web Services Invocation Framework
WSIL	Web Services Invocation Language

XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations
z800	zSeries 800
z890	zSeries 890
z990	zSeries 990
zAAP	zSeries Application Assist Processor
zELC	zSeries Entry License Charge

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 707. Note that some of the documents referenced here may be available in softcopy only.

- *Enabling SOA Using WebSphere Messaging*, SG24-7163
- *IBM Rational Application Developer V6 Portlet Application Development and Portal Tools*, SG24-6681
- *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151
- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- *Patterns: Implementing Self-Service in an SOA Environment*, SG24-6680
- *Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6*, SG24-6494
- *High Availability Considerations: SAP R/3 on DB2 for OS*, SG24-2003
- *XML for DB2 Information Integration*, SG24-6994
- *IBM HTTP Server (powered by Apache): An Integrated Solution for IBM eServer iSeries Servers*, SG24-6716
- *IBM WebSphere Portal V5 A Guide for Portlet Application Development*, SG24-6076
- *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794
- *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435
- *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- *DB2 UDB/WebSphere Performance Tuning Guide*, SG24-6417
- *Using Informix Dynamic Server with WebSphere*, SG24-6948
- *WebSphere for z/OS to CICS and IMS Connectivity Performance*, REDP-3959

Other publications

These publications are also relevant as further information sources:

- *DB2 UDB for z/OS Version 8 Installation Guide,* GC18-7418
- *DB2 Universal Database for z/OS Version 8 Application Programming Guide and Reference for Java*, SC18-7414
- *IMS Version 9: Installation Volume 1: Installation Verification,* GC26-9429
- *IMS Version 9: IMS Java Guide and Reference*, SC18-7821

© Copyright IBM Corp. 2006. All rights reserved.

- *IBM WebSphere Application Server for z/OS Version 6.0.1: Developing and Deploying Applications*, SA22-7959
- *Service-Oriented Architecture,* Systems Journal, Vol.44, No.4, 2005. Available from:

  http://www.research.ibm.com/journal/sj/

Online resources

These Web sites and URLs are also relevant as further information sources:

- SOA information

  http://www.ibm.com/software/solutions/soa/
- SOA and Web services information

  http://www.ibm.com/developerworks/webservices
- The WebSphere Information Integrator

  http://www.ibm.com/software/data/integration/db2ii/
- Information Integrator Web Service wrapper and examples of data sources

  http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp
- WebSphere Portal Server PUMA

  http://publib.boulder.ibm.com/infocenter/wpdoc/v510/index.jsp?topic=/com.ibm.wp.ent.doc/wps/wpspuma.html
- IBM Web Services Navigator: Overview

  http://www.alphaworks.ibm.com/tech/wsnavigator
- Open source XML frameworks

  http://xml.apache.org/
- The Web Services Interoperability Organization

  http://www.ws-i.org/
- IMS SOAP Gateway with XML Adapter Beta

  https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=imssoap
- IMS Connector for Java

  http://www-306.ibm.com/software/data/db2imstools/imstools/imsjavcon.html
- WebSphere Application Server Version 6.0 Information Center, available at:

  http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp
- The XML Encryption workgroup home page is available at:

  http://www.w3.org/Encryption/
- The WS-Security specification 1.0 is available at:

  http://www.ibm.com/developerworks/library/ws-secure/
- *Security in a Web Services World: A Proposed Architecture and Roadmap*, proposed by IBM and Microsoft

  http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/
- OASIS WS-Security 1.0 and token profiles is available at:

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

- Web Services Security: SOAP Message: Errata 1.0

  http://www.oasis-open.org/committees/download.php/9292/oasis-200401-wsssoap-message-security-1%200-errata-003.pdf
- PHP home page

  http://www.php.net
- The PHP 5.1.2 source code and Windows Binaries can be download from:

  http://www.php.net/downloads.php
- The IBM HTTP Server can be downloaded from the product page:

  http://www-306.ibm.com/software/webservers/httpservers
- Netcraft

  http://news.netcraft.com
- CICS family products homepage at:

  http://www.ibm.com/software/htp/cics
- CICS Transaction Server v3.1 documentation

  http://publib.boulder.com/infocenter/cicsts/v3r1/index.jsp

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

© Copyright IBM Corp. 2006. All rights reserved.

IBM

Redbooks

Powering SOA with IBM Data Servers

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Powering SOA with IBM Data Servers

Understand the role of data servers within service-oriented architecture (SOA)

Map the current portfolio of products to the architecture

Follow an implementation premised on diversity

Flexibility in business has become equal in importance with operational efficiency. Service-oriented architecture (SOA) can help businesses respond more quickly and cost-effectively to the changing market conditions by promoting reuse and interconnection of existing IT assets rather than time-consuming and costly reinvention.

SOA has been the top fashionable topic in IT for a few years now. This is because there is a consensus of opinions among enterprise architects that SOA is the key to making the IT department a catalyst for growth and innovation.

This IBM Redbook helps you get started with SOA by showing the implementation of the minimum requirements: The creation of Web services that allow access to data that is stored in data servers or applications and the realization of interaction services for business to consumer integration. The data servers included in our scenario are DB2 for z/OS, DB2 for Linux, UNIX and Windows, Informix Dynamic Server and IMS.

This redbook is a roadmap showing how SOA can significantly improve the IT business value.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7259-00 ISBN 0738494542