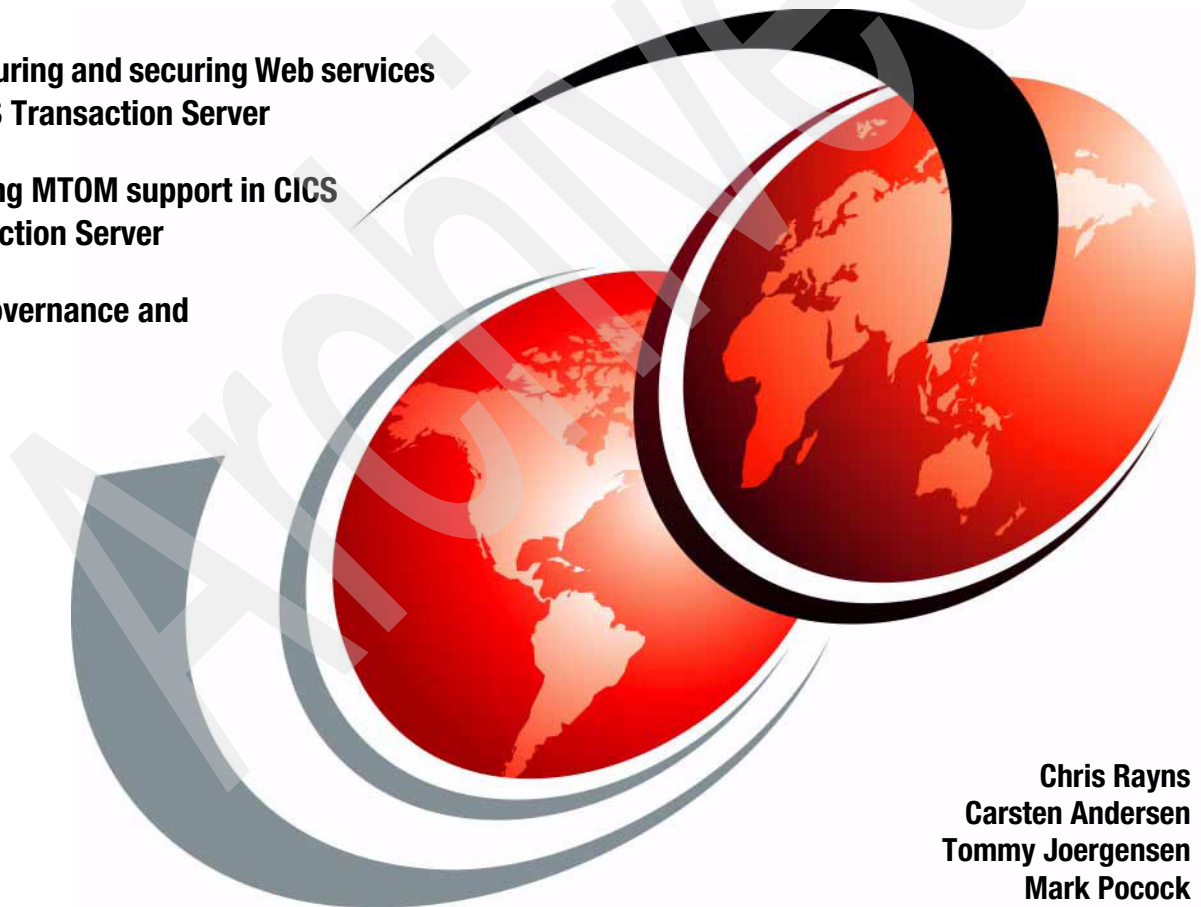


Implementing CICS Web Services

Configuring and securing Web services
in CICS Transaction Server

Enabling MTOM support in CICS
Transaction Server

SOA governance and
WSRR



Chris Rayns
Carsten Andersen
Tommy Joergensen
Mark Pocock



International Technical Support Organization

Implementing CICS Web Services

November 2008

Archived

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

First Edition (November 2008)

This edition applies to Version 3, Release 2, CICS Transaction Server.

© Copyright International Business Machines Corporation 2008. 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

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this book	xi
Become a published author	xii
Comments welcome	xiii
Part 1. Introduction	1
Chapter 1. Overview of Web services	3
1.1 Introduction	4
1.2 Service-oriented architecture	4
1.2.1 Characteristics	6
1.2.2 Web services versus service-oriented architectures	6
1.3 Web services	7
1.3.1 Properties of a Web service	7
1.3.2 Core standards	8
1.3.3 Web Services Interoperability (WS-I)	11
1.3.4 Additional standards	12
1.4 IBM WebSphere Service Registry and Repository	13
1.5 SOAP	13
1.5.1 The envelope	13
1.5.2 Communication styles	18
1.5.3 Encodings	18
1.5.4 Messaging modes	19
1.6 WSDL	19
1.6.1 WSDL Document	20
1.6.2 WSDL document anatomy	21
1.6.3 WSDL definition	25
1.6.4 WSDL bindings	31
1.7 Summary	33
Chapter 2. CICS implementation of Web services	35
2.1 SOAP support in CICS TS V2	36
2.1.1 Pipelines	36
2.1.2 Limitations	38
2.2 Support for Web services in CICS TS V3	38
2.2.1 What is provided in CICS TS V3.1	38

2.2.2	What is new in CICS TS V3.2	41
2.2.3	Comparing CICS TS V3 with the SOAP for CICS feature	43
2.3	CICS as a service provider	46
2.3.1	Preparing to run a CICS application as a service provider	46
2.3.2	Processing the inbound service request	48
2.4	CICS as a service requester	50
2.4.1	Preparing to run a CICS application as a service requester	50
2.4.2	Processing the outbound service request	52
2.5	Catalog manager example application	53
2.6	CICS TS V3 Web service resource definitions	57
2.6.1	URIMAP	57
2.6.2	PIPELINE	58
2.6.3	WEBSERVICE	65
2.6.4	Web service binding file	68
2.7	Tools for application deployment	70
2.7.1	CICS Web services assistant	70
2.7.2	WebSphere Developer for System z	77
2.7.3	Comparing Web services assistant and WebSphere Developer for System z	79
Part 2.	Web service configuration	83
Chapter 3.	Web services using HTTP	85
3.1	Preparation	86
3.1.1	Software checklist	87
3.1.2	Definition checklist	87
3.1.3	The sample application	88
3.2	Configuring CICS as a service provider	88
3.2.1	Configuring code page support	89
3.2.2	Configuring CICS	89
3.2.3	Configuring WebSphere Application Server on Windows	99
3.2.4	Testing the configuration	102
3.3	Configuring CICS as a service requester	106
3.3.1	Configuring CICS	107
3.3.2	Configuring WebSphere Application Server for z/OS	110
3.3.3	Testing the configuration	112
3.4	Configuring for high availability	113
3.4.1	TCP/IP load balancing	114
3.4.2	High availability configuration	114
3.4.3	Routing inbound Web service requests	114
3.5	Problem determination	116
3.5.1	Error calling dispatch service: INVREQ	116
3.5.2	Diagnosing the problem	117

Chapter 4. Web services using WebSphere MQ	123
4.1 Preparation	124
4.1.1 Software checklist	125
4.1.2 Definition checklist	125
4.2 WebSphere MQ configuration	126
4.2.1 Adding WebSphere MQ support to CICS	126
4.2.2 Defining the queues	127
4.2.3 Defining the trigger process	128
4.3 Configuring CICS as a service provider using WMQ	128
4.3.1 Interface for dispatching an order	128
4.3.2 Configuring the service provider pipeline	129
4.4 Configuring CICS as service requester using WMQ	133
4.4.1 Configuring the catalog application	135
4.4.2 Configuring WebSphere Application Server on Windows	136
4.5 Testing the WMQ configuration	137
4.6 High availability with WMQ	139
Chapter 5. MTOM/XOP optimization	143
5.1 Preparation	144
5.1.1 Software checklist	144
5.1.2 Definition checklist	144
5.1.3 The sample application	145
5.1.4 Testing the scenario	145
5.2 Configuring for MTOM/XOP optimization	148
5.2.1 Configuration steps	149
5.2.2 Testing the MTOM/XOP optimization	151
Chapter 6. SOA governance and WSRR	153
6.1 SOA governance	154
6.1.1 What is SOA governance?	154
6.1.2 SOA governance in practice	155
6.1.3 Aspects of SOA governance	156
6.2 WebSphere Service Registry and Repository	163
6.2.1 What is WebSphere Service Registry and Repository?	164
6.2.2 WebSphere Service Registry and Repository in practice	165
6.3 Why interoperate CICS and WebSphere?	168
6.4 CICS SupportPac CA1N	170
6.5 Installation	171
6.5.1 Software requirements	171
6.5.2 Downloading and installing the CICS SupportPac	171
6.6 Publishing WSDL files from z/OS batch to WSRR	173
6.6.1 Job control statements for DFHWS2SR	174
6.6.2 Parameters for DFHWS2SR	174

6.7	Retrieving WSDL files from WSRR using z/OS batch	179
6.7.1	Job control statements for DFHSR2WS	181
6.7.2	Parameters for DFHSR2WS	181
6.8	Setup for testing the SupportPac with WSRR 6.1	185
6.8.1	Software checklist	186
6.8.2	Running DFHWS2SR	186
6.8.3	Running DFHSR2WS	191
6.9	Security considerations	193
6.9.1	Configuring SSL between the z/OS batch utilities and WSRR	193
6.9.2	Example of using self-signed certificates	193
Part 3.	Transaction management	195
Chapter 7.	Introduction to Web services: Atomic transactions	197
7.1	Beginner's guide to atomic transactions	198
7.1.1	What is a classic transaction?	199
7.1.2	Mapping from classic transactions to WS-Atomic Transaction	203
7.2	WS-Addressing	206
7.2.1	Endpoint references	207
7.2.2	Message information headers	210
7.2.3	SOAP binding for endpoint references	212
7.3	WS-Coordination	213
7.3.1	Coordination service	214
7.3.2	CreateCoordinationContext	215
7.3.3	CreateCoordinationContextResponse	218
7.3.4	Register	220
7.3.5	Register response	222
7.3.6	Two applications with their own coordinators	222
7.3.7	Addressing requirements for WS-Coordination message types	224
7.4	WS-Atomic Transaction	224
7.4.1	Completion protocol	225
7.4.2	Two-Phase Commit protocol	227
7.4.3	Two applications with their own coordinators (continued)	229
7.4.4	Addressing requirements for WS-AT message types	230
7.4.5	CICS TS V3.2 and resynchronization processing	231
Chapter 8.	Enabling atomic transactions	235
8.1	Enabling atomic transactions in CICS	236
8.1.1	CICS to CICS configuration	236
8.1.2	More elaborate CICS to CICS configuration	247
8.2	Enabling atomic transactions in WebSphere	249
Chapter 9.	Transaction scenarios	251
9.1	Introduction to our scenarios	252

9.1.1	Software checklist	253
9.1.2	Definition checklist	253
9.2	The simple atomic transaction scenario	255
9.2.1	Setting up CICS for the simple scenario	257
9.2.2	Creating the AtomicClient and ITSO.ORDER table	261
9.2.3	Testing the simple scenario	277
9.3	The daisy chain atomic transaction scenario	298
9.3.1	Setting up CICS for the daisy chain scenario	300
9.3.2	Creating DispatchOrderAtomic and the ITSO.DISPATCH table	302
9.3.3	Testing the daisy chain scenario	305
9.4	Transaction scenario summary	314
	Appendix A. Additional material	315
	Locating the Web material	315
	Related publications	317
	IBM Redbooks	317
	Other publications	317
	Online resources	317
	How to get IBM Redbooks publications	318
	Help from IBM	318
	Index	319

Archived

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

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

CICSplex®
CICS®
DB2®
developerWorks®
IBM®
IMS™

MVS™
Parallel Sysplex®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®

SupportPac™
System z®
Tivoli®
VTAM®
WebSphere®
z/OS®

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.

EJB, J2EE, Java, JavaServer, JDBC, JMX, JSP, JVM, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

Preface

Today more and more companies are embracing the principles of on demand business by integrating business processes end-to-end across the company and with key partners, enabling them to respond flexibly and rapidly to new circumstances. The move to an on demand business environment requires technical transformation, moving the focus from discrete applications to connected, interdependent information technology components.

Open standards such as Web services enable these components to be hosted in the environments most appropriate to their requirements, while still being able to interact easily — independent of hardware, run-time environment, and programming language.

The Web services support in CICS® Transaction Server Version 3 enables your CICS programs to be Web service providers and requesters. CICS supports a number of specifications including SOAP Version 1.1 and Version 1.2, and Web services distributed transactions (WS-Atomic Transaction).

This IBM® Redbooks publication describes how to configure CICS Web services support for HTTP-based and WebSphere® MQ-based solutions, and demonstrates how Web services can be used to integrate J2EE™ applications running in WebSphere Application Server with COBOL programs running in CICS.

The book begins with an overview of Web services standards and the Web services support provided by CICS TS V3. Complete details for configuring CICS Web services using both HTTP and WebSphere MQ are provided next. We concentrate on the implementation specifics such as security, transactions, and availability.

The team that wrote this book

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

Chris Rayns is an IT Specialist and the CICS Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively on all areas of CICS. Before joining the ITSO, Chris worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

Carsten Andersen is a System Designer at Jyske Bank in Denmark. He has more than 25 years of experience in development of IT- Systems. He holds a degree in Financials from The Danish Bank Academy. His areas of expertise include main frame development, project management, DB2®, CICS, COBOL, and infrastructure. He has written extensively on MTOM and XOP.

Tommy Joergensen is a Senior IT Specialist working for IBM Global Services in IBM Denmark. He has more than 25 years of experience working in CICS technical support, including three years at IBM Hursley. In recent years he has delivered services at large accounts in Denmark for both the CICS and WebSphere products. Tommy is the IBM representative in the CICS working group of the Nordic Share Guide organization.

Mark Pocock is a Software Engineer in the CICS Transaction Server level 3 support team and has nine years of experience supporting CICS. He holds a degree in Mathematics and Computer Science from the University of Kent. His areas of expertise within CICS include Web Support, Web Services, 3270 Bridge, data conversion, and Java™ and EJB™ support.

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

Richard M. Conway
International Technical Support Organization, Raleigh Center

Nigel Williams
IBM Montpellier

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will 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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks® in one of the following ways:

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

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbooks@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

Archived



Part 1

Introduction

In this part of the book, we give you a broad overview of different Web services technologies and then explain how to use Web services in CICS Transaction Server V3.

Archived



Overview of Web services

In this chapter, we focus on some of the architectural concepts that have to be considered in a Web services project. We define and discuss *service-oriented architecture* (SOA) and the relationship between SOAs and Web services.

We then take a closer look at Web services, a technology that enables you to invoke applications using Internet protocols and standards. The technology is called “Web services” because it integrates services (applications) using Web technologies (the Internet and its standards).

1.1 Introduction

There is a strong trend for companies to integrate existing systems to implement IT support for business processes that cover the entire business cycle. Today, interactions already exist using a variety of schemes that range from very rigid point-to-point electronic data interchange (EDI) interactions to open Web auctions. Many companies have already made some of their IT systems available to all of their divisions and departments, or even their customers or partners on the Web. However, techniques for collaboration vary from one case to another and are thus proprietary solutions; systems often collaborate without any vision or architecture.

Thus, there is an increasing demand for technologies that support the connecting or sharing of resources and data in a very flexible and standardized manner. Because technologies and implementations vary across companies and even within divisions or departments, unified business processes cannot be smoothly supported by technology. Integration has been developed only between units that are already aware of each other and that use the same static applications.

Furthermore, there is a requirement to further structure large applications into building blocks in order to use well-defined components within different business processes. A shift towards a *service-oriented* approach not only can standardize interaction, but also allows for more flexibility in the process. The complete value chain within a company is divided into small modular functional units, or services. A service-oriented architecture thus has to focus on how services are described and organized to support their dynamic, automated discovery and use.

Companies and their sub-units should be able to easily provide services. Other business units can use these services in order to implement their business processes. This integration can be ideally performed during the runtime of the system, not just at the design time.

1.2 Service-oriented architecture

This section is a short introduction to the key concepts of a service-oriented architecture. The architecture makes no statements about the infrastructure or protocols it uses. Therefore, you can implement a service-oriented architecture using technologies other than Web technologies.

As shown in Figure 1-1, a service-oriented architecture contains three basic components:

- ▶ A service provider:
The service provider creates a Web service and possibly publishes to the service broker the information necessary to access and interface with the Web service.
- ▶ A service broker:
The service broker (also known as a service registry) makes the Web service access and interface information available to any potential service requester.
- ▶ A service requestor:
The service requester binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

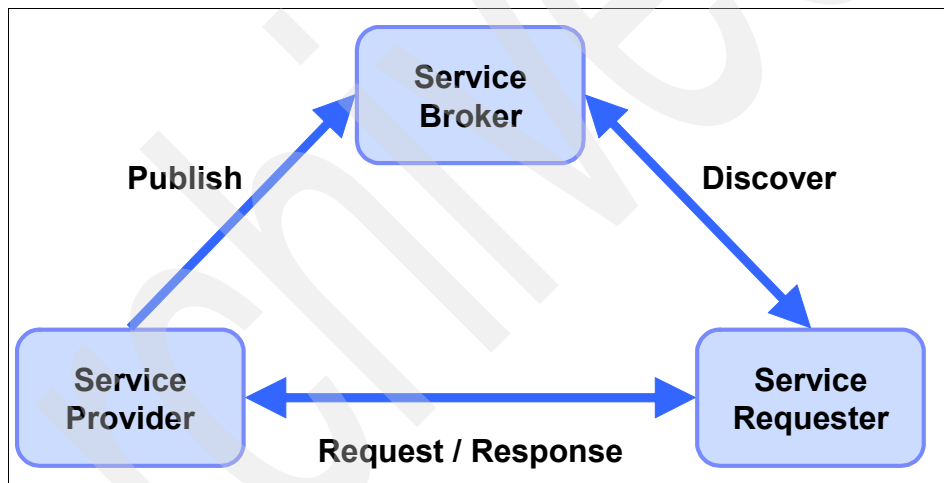


Figure 1-1 Web services components and operations

Each component can also act as one of the two other components. For example, if a service provider requires information that it can only acquire from some other service, it acts as a service requester while still serving the original request.

- ▶ The *service provider* creates a Web service and possibly publishes its interface and access information to the service broker.
- ▶ The *service broker* (also known as *service registry*) is responsible for making the Web service interface and implementation access information available to any potential service requestor.

- ▶ The *service requestor* binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

1.2.1 Characteristics

The service-oriented architecture uses a loose coupling between the participants. Such a loose coupling provides greater flexibility as follows:

- ▶ Old and new functional blocks are encapsulated into components that work as services.
- ▶ Functional components and their interfaces are separated. Therefore, new interfaces can be plugged in more easily.
- ▶ Within complex applications, the control of business processes can be isolated. A business rule engine can be incorporated to control the workflow of a defined business process. Depending on the state of the workflow, the engine calls the respective services.

1.2.2 Web services versus service-oriented architectures

The service-oriented architecture has been used under various guises for many years. It can be, and has been, implemented using a number of different distributed computing technologies, such as CORBA or messaging middleware. The effectiveness of service-oriented architectures in the past has always been limited by the ability of the underlying technology to interoperate across the enterprise.

Web services technology is an ideal technology choice for implementing a service-oriented architecture:

- ▶ Web services are standards based. Interoperability is a key business advantage within the enterprise and is crucial in B2B scenarios.
- ▶ Web services are widely supported across the industry. For the very first time, all major vendors are recognizing and providing support for Web services.
- ▶ Web services are platform and language agnostic; there is no bias for or against a particular hardware or software platform. Web services can be implemented in any programming language or toolset. This is important because continued industry support exists for the development of standards and interoperability between vendor implementations.
- ▶ This technology provides a migration path to gradually enable existing business functions as Web services are required.
- ▶ This technology supports synchronous and asynchronous, RPC-based, and complex message-oriented exchange patterns.

Conversely, there are many Web services implementations that are not a service-oriented architecture. For example, the use of Web services to connect two heterogeneous systems directly together is not an SOA. These uses of Web services solve real problems and provide significant value on their own. They can form the starting point of an SOA.

In general, an SOA has to be implemented at an enterprise or organizational level in order to harvest many of the benefits.

For more information about the relationship between Web services and service-oriented architectures, or the application of IBM Patterns for e-business to a Web services project, refer to *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303.

1.3 Web services

Web services perform encapsulated business functions, ranging from simple request-reply to full business process interactions. These services can be new applications or just wrapped around existing business functions to make them network-enabled. Services can rely on other services to achieve their goals.

It is important to note from this definition that a Web service is not constrained to using SOAP over HTTP/S as the transport mechanism. Web services are equally at home in the messaging world.

1.3.1 Properties of a Web service

All Web services share 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 an HTTP server and a SOAP server are required.
- ▶ Web services are self-describing.
Using Web Services Description Language (WSDL), all the information required to implement a Web service as a provider, or to invoke a Web service as a requester, is provided.
- ▶ 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.

- ▶ Web services are modular.
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 are language-independent and interoperable.
The client and server can be implemented in different environments. Theoretically, any language can be used to implement Web service clients and servers.
- ▶ Web services are inherently open and standard-based.
XML and HTTP are the major technical foundations 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.
- ▶ 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 reconfiguration 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.

1.3.2 Core standards

Web services are built upon four core standards, as we explain in the following sections.

Extensible Markup Language (XML)

XML is the foundation of Web services. However, since much information has already been written about XML, we do not describe it in this document. You can find information about XML at:

<http://www.w3.org/XML/>

SOAP

Originally proposed by Microsoft®, SOAP was designed to be a simple and extensible specification for the exchange of structured, XML-based information in a decentralized, distributed environment. As such, it represents the main means of communication between the three actors in an SOA: the service provider, the service requester, and the service broker. A group of companies, including IBM, submitted SOAP to the W3C for consideration by its XML Protocol Working Group. There are currently two versions of SOAP: Version 1.1 and Version 1.2.

The SOAP 1.1 specification contains three parts:

- ▶ An envelope that defines a framework for describing message content and processing instructions. Each SOAP message consists of an envelope that contains an arbitrary number of headers and one body that carries the payload. SOAP messages might contain faults: faults report failures or unexpected conditions.
- ▶ A set of encoding rules for expressing instances of application-defined data types.
- ▶ A convention for representing remote procedure calls and responses.

A SOAP message is, in principle, independent of the transport protocol that is used, and can, therefore, potentially be used with a variety of protocols, such as HTTP, JMS, SMTP, or FTP. Right now, the most common way of exchanging SOAP messages is through HTTP.

The way SOAP applications communicate when exchanging messages is often referred to as the message exchange pattern (MEP). The communication can be either one-way messaging, where the SOAP message only goes in one direction, or two-way messaging, where the receiver is expected to send back a reply.

Due to the characteristics of SOAP, 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.

Note: The authors of the SOAP 1.1 specification declared that the acronym SOAP stands for Simple Object Access Protocol. The authors of the SOAP 1.2 specification decided not to give any meaning to the acronym SOAP.

Web Services Description Language (WSDL)

This standard describes Web services as abstract service endpoints that operate on messages. Both the operations and the messages are defined in an abstract manner, while the actual protocol used to carry the message and the endpoint's address are concrete.

WSDL is not bound to any particular protocol or network service. It can be extended to support many different message formats and network protocols. However, because Web services are mainly implemented using SOAP and HTTP, the corresponding bindings are part of this standard.

The WSDL 1.1 specification only defines bindings that describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET and POST, and MIME. The specification for WSDL 1.1 can be found at:

<http://www.w3.org/TR/wsd1>

WSDL 2.0 provides a model as well as an XML format for describing Web services. It enables you to separate the description of the abstract functionality offered by a service from the concrete details of a service description. It also describes extensions for Message Exchange Patterns, SOAP modules, and a language for describing such concrete details for SOAP1.2 and HTTP.

There are eight Message Exchange Patterns defined. CICS TS V3.2 supports four of them:

- ▶ **In-Only:**
A request message is sent to the Web service provider, but the provider is not allowed to send any type of response to the Web service requester.
- ▶ **In-Out:**
A request message is sent to the Web service provider, and a response message is returned. The response message can be a normal SOAP message or a SOAP fault.
- ▶ **In-Optional-Out:**
A request message is sent to the Web service provider, and a response message is optionally returned to the requester. The response message can be a normal SOAP message or a SOAP fault.
- ▶ **Robust-In-Only:**
A request message is sent to the Web service provider, and no response message is returned to the requester unless an error occurs. In this case, a SOAP fault message is sent to the requester.

The other four Message Exchange Patterns that CICS TS V3.2 does not support are:

- ▶ Out-Only
- ▶ Robust-Out-Only
- ▶ Out-In
- ▶ Out-Optional-In

The specification for WSDL 2.0 can be found at:

<http://www.w3.org/TR/wsd120>

Universal Description, Discovery, and Integration (UDDI)

The Universal Description, Discovery, and Integration standard defines a means to publish and to discover Web services. As of this writing, UDDI Version 3.0 has been finalized, but UDDI Version 2.0 is still more commonly used. For more information, refer to:

<http://www.uddi.org/>

<http://www.oasis-open.org/specs/index.php#wssv1.0>

1.3.3 Web Services Interoperability (WS-I)

Web services can be used to connect computer systems together across organizational boundaries. Therefore, a set of open non-proprietary standards that all Web services adhere to maximizes the ability to connect disparate systems together.

The Web Service Interoperability group (WS-I) is an organization that promotes open interoperability between Web services regardless of platform, operating systems, and programming languages. To promote this cause, the WS-I has released a basic profile that outlines a set of specifications to which WSDL documents and Web services traffic (SOAP over HTTP transport) must adhere in order to be WS-I compliant. The full list of specifications can be found at the WS-I Web site:

<http://www.ws-i.org/>

IBM is a member of the WS-I community, and CICS support for Web services is fully compliant with the WS-I basic profile 1.0.

1.3.4 Additional standards

There are other Web services specifications that are now supported by CICS. For a list of the limitations of CICS support, refer to *CICS Web Services Guide*, SC34-6838.

Web Services Atomic Transaction

This specification, commonly known as WS-Atomic Transaction, defines the atomic transaction coordination type for transactions of a short duration. Together with the Web Services Coordination specification, it defines protocols for short term transactions that enable transaction processing systems to interoperate in a Web services environment. Transactions that use WS-Atomic Transaction have the properties of atomicity, consistency, isolation, and durability (ACID).

Web Services Security: SOAP Message Security

This specification is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. The specification provides three main mechanisms that can be used independently or together:

- ▶ The ability to send security tokens as part of a message, and for associating the security tokens with message content
- ▶ The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- ▶ The ability to protect the contents of a message from unauthorized disclosure (message confidentiality)

Web Services Trust Language

This specification, commonly known as WS-Trust, defines extensions that build on Web Services Security to provide a framework for requesting and issuing security tokens, and broker trust relationships.

SOAP Message Transmission Optimization Mechanism (MTOM)

This specification is one of a related pair of specifications that define how to optimize the transmission and format of a SOAP message. MTOM defines:

- ▶ How to optimize the transmission of base64 binary data in SOAP messages.
- ▶ How to implement optimized MIME multipart serialization of SOAP messages in a binding, independent way.

- ▶ The implementation of MTOM relies on the related XML-binary Optimized Packaging (XOP) specification. As these two specifications are so closely linked, they are normally referred to as MTOM/XOP.

1.4 IBM WebSphere Service Registry and Repository

IBM provides an enterprise strength solution that enables governance of SOA artifacts, most of which are related to Web services. The IBM WebSphere Service Registry and Repository (WSRR) product is such a solution.

The product provides an integrated service metadata repository to govern services and manage the service life cycle, promoting visibility and consistency, and reducing redundancy in your organization. You can seamlessly publish and find capabilities across all phases of SOA, enriching connectivity with dynamic and efficient interactions between services at runtime.

You can use the *CICS TS support for WebSphere Service Registry and Repository SupportPac™ CA1N* to publish Web service artifacts that you develop for CICS in WSRR. You can also fetch them back into the CICS environment. CICS SupportPacs are available at:

<http://www.software.ibm.com/ts/cics/txppacs>

1.5 SOAP

In this section we focus mainly on SOAP 1.1.

1.5.1 The envelope

A SOAP message is an *envelope* containing zero or more *headers* and exactly one *body*:

- ▶ The envelope is the root element of the XML document, providing a container for control information, the addressee of a message, and the message itself.
- ▶ Headers contain control information, such as quality of service attributes.
- ▶ The body contains the message identification and its parameters.
- ▶ Both the headers and the body are child elements of the envelope element.

Figure 1-2 shows a simple SOAP request message.

- ▶ The header tells *who* must deal with the message and *how* to deal with it. When the actor is next or when actor is omitted, the receiver of the message must do what the body says. Furthermore, the receiver must understand and process the application-defined <TranID> element.
- ▶ The body tells *what* has to be done: Dispatch an order for quantityRequired 1 of itemRefNumber 0010 to customerID CB1 in chargeDepartment ITSO.

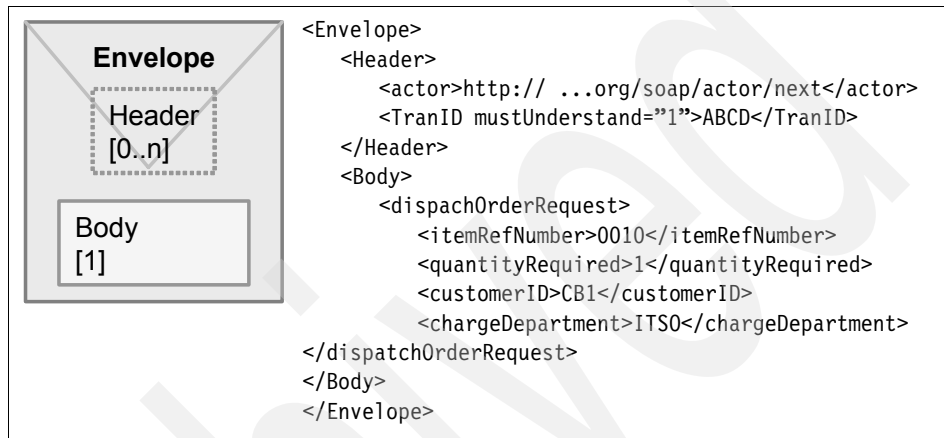


Figure 1-2 Example of a simple SOAP message

Namespaces

Namespaces play an important role in SOAP messages. A namespace is simply a way of adding a qualifier to an element name to ensure that it is unique.

For example, we might have a message that contains an element <customer>. Customers are fairly common, so it is very likely that many Web services have customer elements. To ensure that we know what customer we are talking about, we declare a namespace for it, for example, as follows:

```
xmlns:itso="http://itso.ibm.com/CICS/catalogApplication"
```

This identifies the prefix *itso* with the declared namespace. Then whenever we reference the element <customer> we prefix it with the namespace as follows: <itso:customer>. This identifies it uniquely as a customer type for our application. Namespaces can be defined as any unique string. They are often defined as URLs because URLs are generally globally unique, and they have to be in URL format. These URLs do not have to physically exist though.

The WS-I Basic Profile 1.0 requires that all application-specific elements in the body must be namespace qualified to avoid collisions between names.

Table 1-1 shows the namespaces of SOAP and WS-I Basic Profile 1.0 used in this book.

Table 1-1 SOAP namespaces

Namespace URI	Explanation
http://schemas.xmlsoap.org/soap/envelope/	SOAP 1.1 envelope namespace
http://schemas.xmlsoap.org/soap/encoding/	SOAP 1.1 encoding namespace
http://www.w3.org/2001/XMLSchema-instance	Schema instance namespace
http://www.w3.org/2001/XMLSchema	XML Schema namespace
http://schemas.xmlsoap.org/wsdl	WSDL namespace for WSDL framework
http://schemas.xmlsoap.org/wsdl/soap	WSDL namespace for WSDL SOAP binding
http://ws-i.org/schemas/conformanceClaim/	WS-I Basic Profile

SOAP envelope

The Envelope is the root element of the XML document representing the message; it has the following structure:

```
<SOAP-ENV:Envelope .... >
  <SOAP-ENV:Header>
    <SOAP-ENV:HeaderEntry.... />
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    [message payload]
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In general, a SOAP message is a (possibly empty) set of headers plus one body. The SOAP envelope also defines the namespace for structuring messages. The entire SOAP message (headers and body) is wrapped in this envelope.

Headers

Headers are a generic and flexible mechanism for extending a SOAP message in a decentralized and modular way without prior agreement between the parties involved. They allow control information to pass to the receiving SOAP server and also provide extensibility for message structures.

Headers are optional elements in the envelope. If present, the Header element *must* be the first immediate child element of a SOAP Envelope element. All immediate child elements of the Header element are called *header entries*.

There is a predefined header attribute called `SOAP-ENV:mustUnderstand`. The value of the `mustUnderstand` attribute is either 1 or 0. The absence of the `SOAP mustUnderstand` attribute is semantically equivalent to the value 0.

If the `mustUnderstand` attribute is present in a header entry and set to 1, the service provider must implement the semantics defined by the element:

```
<Header>
  <thens:TranID mustUnderstand="1">ABCD</thens:TranID>
</Header>
```

In the example, the header entry specifies that a service invocation must fail if the service provider does not support the ability to process the `TranID` header.

A SOAP intermediary is an application that is capable of both receiving and forwarding SOAP messages on their way to the final destination. In realistic situations, not all parts of a SOAP message might be intended for the ultimate destination of the SOAP message, but, instead, might be intended for one or more of the intermediaries on the message path. Therefore, a second predefined header attribute, `SOAP-ENV:actor`, is used to identify the recipient of the header information. In SOAP 1.2 the `actor` attribute is renamed `SOAP-ENV:role`. The value of the `SOAP actor` attribute is the URI of the mediator, which is also the final destination of the particular header element (the mediator does not forward the header).

If the `actor` is omitted or set to the predefined default value, the header is for the actual recipient and the actual recipient is also the final destination of the message (body). The predefined value is:

```
http://schemas.xmlsoap.org/soap/actor/next
```

If a node on the message path does not recognize a `mustUnderstand` header and the node plays the role specified by the `actor` attribute, the node must generate a `SOAP MustUnderstand` fault. Whether the fault is sent back to the sender depends on the message exchange pattern in use. For request/response, the `WS-I BP 1.0` requires the fault to be sent back to the sender. Also, according to `WS-I BP 1.0`, the receiver node must discontinue normal processing of the SOAP message after generating the fault message.

Headers can carry authentication data, digital signatures, encryption information, and transactional settings. They can also carry client-specific or project-specific controls and extensions to the protocol; the definition of headers is not just up to standards bodies.

Note: The header must not include service instructions (that would be used by the service implementation).

Body

The SOAP Body element provides a mechanism for exchanging information intended for the ultimate recipient of the message. The Body element is encoded as an immediate child element of the SOAP Envelope element. If a Header element is present, then the Body element *must* immediately follow the Header element. Otherwise it *must* be the first immediate child element of the Envelope element.

All immediate child elements of the Body element are called *body entries*, and each body entry is encoded as an independent element within the SOAP Body element. In the most simple case, the body of a basic SOAP message consists of an XML message as defined by the schema in the `types` section of the WSDL document. It is legal to have any valid XML as the body of the SOAP message, but WS-I conformance requires that the elements be namespace qualified.

Error handling

One area where there are significant differences between the SOAP 1.1 and 1.2 specifications is in the handling of errors. Here we focus on the SOAP 1.1 specification for error handling.

SOAP itself predefines one body element, which is the `fault` element used for reporting errors. If present, the `fault` element must appear as a body entry and must not appear more than once. The children of the `fault` element are defined as follows:

- ▶ `faultcode` is a code that indicates the type of the fault. SOAP defines a small set of SOAP fault codes covering basic SOAP faults:
 - `soapenv:Client`, indicating that the client sent an incorrectly formatted message
 - `soapenv:Server`, for delivery problems
 - `soapenv:VersionMismatch`, which can report any invalid namespaces specified on the Envelope element
 - `soapenv:MustUnderstand`, for errors regarding the processing of header content
- ▶ `faultstring` is a human-readable description of the fault. It must be present in a fault element.
- ▶ `faultactor` is an optional field that indicates the URI of the source of the fault. The value of the `faultactor` attribute is a URI identifying the source that caused the error. Applications that do not act as the ultimate destination of the SOAP message must include the `faultactor` element in a SOAP fault element.

- ▶ `detail` is an application-specific field that contains detailed information about the fault. It must not be used to carry information about errors belonging to header entries. Therefore, the absence of the `detail` element in the `fault` element indicates that the fault is not related to the processing of the body element (the actual message).

For example, a `soapenv:Server` fault message is returned if the service implementation throws a `SOAPException`. The exception text is transmitted in the `faultstring` field.

Although SOAP 1.1 permits the use of custom-defined `faultcodes`, the WS-I Basic Profile only permits the use of the four codes defined in SOAP 1.1.

1.5.2 Communication styles

SOAP supports two different communication styles:

- Document** Also known as *message-oriented* style: This is a very flexible communication style that provides the best interoperability. The message body is any legal XML as defined in the `types` section of the WSDL document.
- RPC** The *remote procedure call* is a synchronous invocation of an operation which returns a result; it is conceptually similar to other RPCs.

1.5.3 Encodings

In distributed computing environments, *encodings* define how data values defined in the application can be translated to and from a protocol format. We refer to these translation steps as *serialization* and *deserialization*, or, synonymously, *marshalling* and *unmarshalling*.

When implementing a Web service, we have to choose one of the tools and programming or scripting languages that support the Web services model. However, the protocol format for Web services is XML, which is independent of the programming language. Thus, SOAP encodings tell the SOAP runtime environment how to translate from data structures constructed in a specific programming language into SOAP XML and vice versa.

The following encodings are defined:

- SOAP encoding** The *SOAP encoding* enables marshalling/unmarshalling of values of data types from the SOAP data model. This encoding is defined in the SOAP 1.1 standard.

Literal

The *literal* encoding is a simple XML message that does not carry encoding information. Usually, an XML Schema describes the format and data types of the XML message.

1.5.4 Messaging modes

The two styles (RPC and document) and the two common encodings (SOAP encoding and literal) can be freely intermixed to produce what is called a SOAP messaging mode. Although SOAP supports four modes, only three of the four modes are generally used, and further, only two are preferred by the WS-I Basic Profile.

- ▶ **Document/literal:** Provides the best interoperability between language environments. The WS-I Basic Profile states that all Web service interactions should use the Document/literal mode.
- ▶ **RPC/literal:** Possible choice between certain implementations. Although RPC/literal is WS-I compliant, it is not frequently used in practice. There are a number of usability issues associated with RPC/literal.
- ▶ **RPC/encoded:** Early Java implementations supported this combination, but it does not provide interoperability with other implementations and is not recommended
- ▶ **Document/encoded:** Not used in practice.

You can find the SOAP 1.1 specification at the following URL:

<http://www.w3.org/TR/2000/NOTE-SOAP-20000508>

The SOAP 1.2 specification is at the following URL:

<http://www.w3.org/TR/SOAP12>

1.6 WSDL

This section introduces Web Services Description Language (WSDL) 1.1. WSDL uses XML to specify the characteristics of a Web service: what the Web service can do, where it resides, and how it is invoked. WSDL can be extended to allow descriptions of different bindings, regardless of what message formats or network protocols are used to communicate.

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

- ▶ Name of the Web service and addressing information
- ▶ Protocol and encoding style to be used when accessing the public operations of the Web service
- ▶ Type information —Operations, parameters, and data types comprising the interface of the Web service, plus a name for this interface

1.6.1 WSDL Document

A WSDL document contains the following main elements:

Types	A container for data type definitions using some type system, usually XML Schema.
Message	An abstract, typed definition of the data being communicated. A message can have one or more typed parts.
Port type	An abstract set of one or more operations supported by one or more ports.
Operation	An abstract description of an action supported by the service that defines the input and output message and optional fault message.
Binding	A concrete protocol and data format specification for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation.
Port	A single endpoint, which is defined as an aggregation of a binding and a network address.
Service	A collection of related ports.

Note that WSDL does not introduce a new type definition language. WSDL recognizes the requirement for rich type systems for describing message formats and supports the XML Schema Definition (XSD) specification.

WSDL 1.1 introduces specific binding extensions for various protocols and message formats. There is a WSDL SOAP binding, which is capable of describing SOAP over HTTP. It is worth noting that WSDL does not define any mappings to a programming language; rather, the bindings deal with transport protocols. This is a major difference from interface description languages, such as the CORBA Interface Definition Language (IDL), which has language bindings.

You can find the WSDL 1.1 specification at the following URL:

<http://www.w3.org/TR/wsdl>

1.6.2 WSDL document anatomy

Figure 1-3 shows the elements comprising a WSDL document and the various relationships between them.

The diagram should be interpreted in the following way:

- ▶ One WSDL document contains zero or more services. A service contains zero or more port definitions (service endpoints), and a port definition contains a specific protocol extension.
- ▶ The same WSDL document contains zero or more bindings. A binding is referenced by zero or more ports. The binding contains one protocol extension, where the style and transport are defined, and zero or more operations bindings. Each of these operation bindings is composed of one protocol extension, where the action and style are defined, and one to three messages bindings, where the encoding is defined.
- ▶ The same WSDL document contains zero or more port types. A port type is referenced by zero or more bindings. This port type contains zero or more operations, which are referenced by zero or more operations bindings.
- ▶ The same WSDL document contains zero or more messages. An operation usually points to an input and an output message, and optionally to some faults. A message is composed of zero or more parts.
- ▶ The same WSDL document contains zero or more types. A type can be referenced by zero or more parts.
- ▶ The same WSDL document points to zero or more XML Schemas. An XML Schema contains zero or more XSD types that define the different data types.

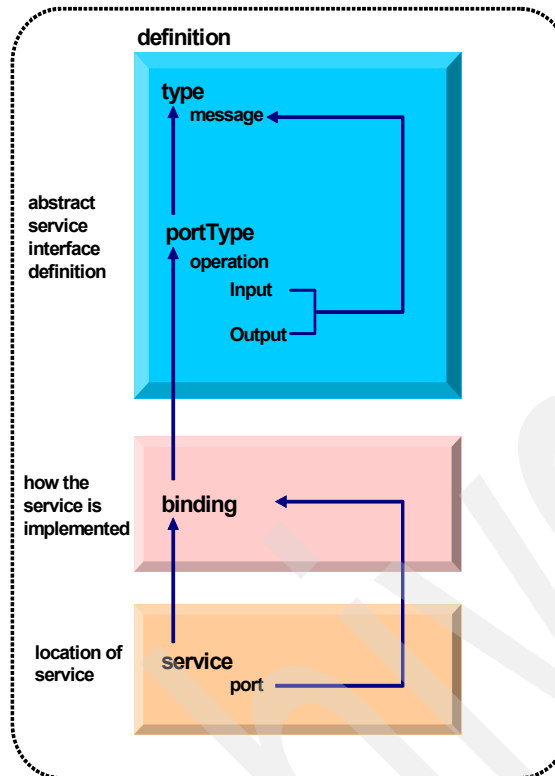


Figure 1-3 WSDL elements and relationships

Example

We now give an example of a simple, complete, and valid WSDL file. As this example shows, even a simple WSDL document contains quite a few elements with various relationships to each other. Example 1-1 contains the WSDL file example. This example is analyzed in detail later in this section.

Example 1-1 Complete WSDL document

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:reqns="http://www.exampleApp.dispatchOrder.Request.com"
  xmlns:resns="http://www.exampleApp.dispatchOrder.Response.com"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.exampleApp.dispatchOrder.com"
  targetNamespace="http://www.exampleApp.dispatchOrder.com">
  <types>
    <xsd:schema xmlns:tns="http://www.exampleApp.dispatchOrder.Request.com"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      attributeFormDefault="qualified">
```

```

        elementFormDefault="qualified"
        targetNamespace="http://www.exampleApp.dispatchOrder.Request.com"
        xmlns:reqns="http://www.exampleApp.dispatchOrder.Request.com">
<xsd:element name="dispatchOrderRequest" nillable="false">
  <xsd:complexType mixed="false">
    <xsd:sequence>
      <xsd:element name="itemReferenceNumber" nillable="false">
        <xsd:simpleType>
          <xsd:restriction base="xsd:short">
            <xsd:maxInclusive value="9999"/>
            <xsd:minInclusive value="0"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="quantityRequired" nillable="false">
        <xsd:simpleType>
          <xsd:restriction base="xsd:short">
            <xsd:maxInclusive value="999"/>
            <xsd:minInclusive value="0"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
<xsd:schema xmlns:tns="http://www.exampleApp.dispatchOrder.Response.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="qualified"
  elementFormDefault="qualified"
  targetNamespace="http://www.exampleApp.dispatchOrder.Response.com">
<xsd:element name="dispatchOrderResponse" nillable="false">
  <xsd:complexType mixed="false">
    <xsd:sequence>
      <xsd:element name="confirmation" nillable="false">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:maxLength value="20"/>
            <xsd:whiteSpace value="preserve"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
<message name="dispatchOrderResponse">
  <part element="resns:dispatchOrderResponse" name="ResponsePart"/>

```

```

</message>
<message name="dispatchOrderRequest">
  <part element="reqns:dispatchOrderRequest" name="RequestPart"/>
</message>
<portType name="dispatchOrderPort">
  <operation name="dispatchOrder">
    <input message="tns:dispatchOrderRequest" name="DFHOXODSRequest"/>
    <output message="tns:dispatchOrderResponse" name="DFHOXODSResponse"/>
  </operation>
</portType>
<binding name="dispatchOrderSoapBinding" type="tns:dispatchOrderPort">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="dispatchOrder">
    <soap:operation soapAction="" style="document"/>
    <input name="DFHOXODSRequest">
      <soap:body parts="RequestPart" use="literal"/>
    </input>
    <output name="DFHOXODSResponse">
      <soap:body parts="ResponsePart" use="literal"/>
    </output>
  </operation>
</binding>
<service name="dispatchOrderService">
  <port binding="tns:dispatchOrderSoapBinding" name="dispatchOrderPort">
    <soap:address
      location="http://myserver:54321/exampleApp/services/dispatchOrderPort"/>
  </port>
</service>
</definitions>

```

Namespaces

WSDL uses the XML namespaces listed in Table 1-2.

Table 1-2 WSDL namespaces

Namespace URI	Explanation
http://schemas.xmlsoap.org/wsdl/	Namespace for WSDL framework.
http://schemas.xmlsoap.org/wsdl/soap/	SOAP binding.
http://schemas.xmlsoap.org/wsdl/http/	HTTP binding.
http://www.w3.org/2000/10/XMLSchema	Schema namespace as defined by XSD.

Namespace URI	Explanation
(URL to WSDL file)	The <i>this namespace</i> (tns) prefix is used as a convention to refer to the current document. Do not confuse it with the XSD <i>target namespace</i> , which is a different concept.

The first three namespaces are defined by the WSDL specification itself; the next definition references a namespace that is defined in the SOAP and XSD standards. The last one is local to each specification.

1.6.3 WSDL definition

The WSDL definition contains types, messages, operations, port types, bindings, ports, and services.

Also, WSDL provides an optional element called `wsdl:document` as a container for human-readable documentation.

Types

The *types* element encloses data type definitions used by the exchanged messages. WSDL uses XML Schema Definition (XSD) as its canonical and built-in type system:

```
<definitions .... >
  <types>
    <xsd:schema .... /> (0 or more)
  </types>
</definitions>
```

The XSD type system can be used to define the types in a message regardless of whether or not the resulting wire format is XML. In our example we have two schema sections; one defines the message format for the input and the other defines the message format for the output.

In our example, the types definition, shown in Example 1-2, is where we specify that there is a complex type called `dispatchOrderRequest`, which is composed of two elements: a `itemReferenceNumber` and a `quantityRequired`.

Example 1-2 Types definition of our WSDL example for the input

```
<types>
  <xsd:schema xmlns:tns="http://www.exampleApp.dispatchOrder.Request.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="qualified"
```

```

        elementFormDefault="qualified"
        targetNamespace="http://www.exampleApp.dispatchOrder.Request.com"
        xmlns:reqns="http://www.exampleApp.dispatchOrder.Request.com">
<xsd:element name="dispatchOrderRequest" nillable="false">
  <xsd:complexType mixed="false">
    <xsd:sequence>
      <xsd:element name="itemReferenceNumber" nillable="false">
        <xsd:simpleType>
          <xsd:restriction base="xsd:short">
            <xsd:maxInclusive value="9999"/>
            <xsd:minInclusive value="0"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="quantityRequired" nillable="false">
        <xsd:simpleType>
          <xsd:restriction base="xsd:short">
            <xsd:maxInclusive value="999"/>
            <xsd:minInclusive value="0"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
.
.
</types>

```

Messages

A *message* represents one interaction between a service requester and a service provider. If an operation is bidirectional at least two message definitions are used in order to specify the transmissions to and from the service provider. A message consists of one or more logical parts.

```

<definitions .... >
  <message name="nmtoken"> (0 or more)
    <part name="nmtoken" element="qname" (0 or 1) type="qname" (0 or
1)/>
    (0 or more)
  </message>
</definitions>

```

The abstract message definitions are used by the operation element. Multiple operations can refer to the same message definition.

Operations and messages are modeled separately in order to support flexibility and simplify reuse of existing definitions. For example, two operations with the same parameters can share one abstract message definition.

In our example, the messages definition, shown in Example 1-3, is where we specify the different parts that compose each message. The request message `dispatchOrderRequest` is composed of an element `dispatchOrderRequest` as defined in the schema in the `parts` section. The response message `dispatchOrderResponse` is similarly defined by the element `dispatchOrderResponse` in the schema. There is no requirement for the names of the message and the schema-defined element to match; in our example we did this merely for convenience.

Example 1-3 Message definition in our WSDL document

```
<message name="dispatchOrderResponse">
  <part element="resns:dispatchOrderResponse" name="ResponsePart" />
</message>
<message name="dispatchOrderRequest">
  <part element="reqns:dispatchOrderRequest" name="RequestPart" />
</message>
```

Port types

A *port type* is a named set of abstract operations and the abstract messages involved:

```
<definitions .... >
  <portType name="nmtoken">
    <operation name="nmtoken" .... /> (0 or more)
  </portType>
</definitions>
```

WSDL defines four types of operations that a port can support:

- | | |
|-------------------------|---|
| One-way | The port receives a message. There is an <i>input</i> message only. |
| Request-response | The port receives a message and sends a correlated message. There is an <i>input</i> message followed by an <i>output</i> message. |
| Solicit-response | The port sends a message and receives a correlated message. There is an <i>output</i> message followed by an <i>input</i> message. |
| Notification | The port sends a message. There is an <i>output</i> message only. This type of operation could be used in a publish/subscribe scenario. |

Each of these operation types can be supported with variations of the following syntax. Presence and order of the input, output, and fault messages determine the type of message:

```
<definitions .... >
  <portType .... > (0 or more)
    <operation name="nmtoken" parameterOrder="nmtokens">
      <input name="nmtoken" (0 or 1) message="qname"/> (0 or 1)
      <output name="nmtoken" (0 or 1) message="qname"/> (0 or 1)
      <fault name="nmtoken" message="qname"/> (0 or more)
    </operation>
  </portType >
</definitions>
```

Note that a request-response operation is an abstract notion. A particular binding must be consulted to determine how the messages are actually sent:

- ▶ Within a single transport-level operation, such as an HTTP request/response message pair, which is the preferred option
- ▶ As two independent transport-level operations, which can be required if the transport protocol only supports one-way communication

In our example, the portType and operation definitions, shown in Example 1-4, are where we specify the port type, called `dispatchOrderPort`, and a set of operations. In this case, there is only one operation, called `dispatchOrder`.

We also specify the interface that the Web service provides to its possible clients, with the input message `DFH0XODSRequest` and the output message `DFH0XODSResponse`. Since the input element appears before the output element in the operation element, our example shows a request-response type of operation.

Example 1-4 Port type and operation definitions in our WSDL document example

```
<portType name="dispatchOrderPort">
  <operation name="dispatchOrder">
    <input message="tns:dispatchOrderRequest" name="DFH0XODSRequest"/>
    <output message="tns:dispatchOrderResponse" name="DFH0XODSResponse"/>
  </operation>
</portType>
```

Bindings

A *binding* contains:

- ▶ Protocol-specific general binding data, such as the underlying transport protocol and the communication style for SOAP.
- ▶ Protocol extensions for operations and their messages.

Each binding references one port type; one port type can be used in multiple bindings. All operations defined within the port type must be bound in the binding. The pseudo XSD for the binding looks like this:

```
<definitions .... >
  <binding name="nmtoken" type="qname"> (0 or more)
    <!-- extensibility element (1) --> (0 or more)
    <operation name="nmtoken"> (0 or more)
      <!-- extensibility element (2) --> (0 or more)
      <input name="nmtoken" (0 or 1) > (0 or 1)
        <!-- extensibility element (3) -->
      </input>
      <output name="nmtoken" (0 or 1) > (0 or 1)
        <!-- extensibility element (4) --> (0 or more)
      </output>
      <fault name="nmtoken"> (0 or more)
        <!-- extensibility element (5) --> (0 or more)
      </fault>
    </operation>
  </binding>
</definitions>
```

As we have already seen, a port references a binding. The port and binding are modeled as separate entities in order to support flexibility and location transparency. Two ports that merely differ in their network address can share the same protocol binding.

The extensibility elements <!-- extensibility element (x) --> use XML namespaces in order to incorporate protocol-specific information into the language- and protocol-independent WSDL specification.

In our example, the binding definition, shown in Example 1-5, is where we specify our binding name, `dispatchOrderSoapBinding`. The connection must be SOAP HTTP, and the style must be document. We provide a reference to our operation, `dispatchOrder`, and we define the input message `DFH0X0DSRequest` and the output message `DFH0X0DSResponse`, both to be SOAP literal.

```
<binding name="dispatchOrderSoapBinding" type="tns:dispatchOrderPort">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="dispatchOrder">
      <soap:operation soapAction="" style="document"/>
      <input name="DFHOXODSRequest">
        <soap:body parts="RequestPart" use="literal"/>
      </input>
      <output name="DFHOXODSResponse">
        <soap:body parts="ResponsePart" use="literal"/>
      </output>
    </operation>
  </binding>
```

Service definition

A *service* definition merely bundles a set of ports together under a name, as the following pseudo XSD definition of the service element shows.

```
<definitions .... >
  <service name="nmtoken"> (0 or more)
    <port .... /> (0 or more)
  </service>
</definitions>
```

Multiple service definitions can appear in a single WSDL document.

Port definition

A *port* definition describes an individual endpoint by specifying a single address for a binding:

```
<definitions .... >
  <service .... > (0 or more)
    <port name="nmtoken" binding="qname"> (0 or more)
    <!-- extensibility element (1) -->
    </port>
  </service>
</definitions>
```

The binding attribute is of type QName, which is a qualified name (equivalent to the one used in SOAP). It refers to a binding. A port contains exactly one network address; all other protocol-specific information is contained in the binding.

Any port in the implementation part must reference exactly one binding in the interface part.

The `<!-- extensibility element (1) -->` is a placeholder for additional XML elements that can hold protocol-specific information. This mechanism is required because WSDL is designed to support multiple runtime protocols.

In our example, the service and port definition, shown in Example 1-6, is where we specify our service, called `dispatchOrderService`, which contains a collection of our ports. In this case, there is only one that uses the `dispatchOrderSoapBinding` and is called `dispatchOrderPort`. In this port, we specify our connection point as, for example, `http://myserver:54321/exampleApp/services/dispatchOrderPort`.

Example 1-6 Service and port definition in our WSDL document example

```
<service name="dispatchOrderService">
  <port binding="tns:dispatchOrderSoapBinding" name="dispatchOrderPort">
    <soap:address
      location="http://myserver:54321/exampleApp/services/dispatchOrderPort"/>
    </port>
  </service>
```

1.6.4 WSDL bindings

We now investigate the WSDL extensibility elements supporting the SOAP transport binding.

SOAP binding

WSDL includes a binding for SOAP 1.1 endpoints, which supports the specification of the following protocol-specific information:

- ▶ An indication that a binding is bound to the SOAP 1.1 protocol
- ▶ A way of specifying an address for a SOAP endpoint
- ▶ The URI for the `SOAPAction` HTTP header for the HTTP binding of SOAP
- ▶ A list of definitions for headers that are transmitted as part of the SOAP envelope

Table 1-3 lists the corresponding extension elements.

Table 1-3 SOAP extensibility elements in WSDL

Extension and attributes		Explanation
<soap:binding ... >		Binding level; specifies defaults for all operations.
	transport="uri" (0 or 1)	Binding level; transport is the runtime transport protocol used by SOAP (HTTP, SMTP, and so on).
	style="rpc document" (0 or 1)	The style is one of the two SOAP communication styles, rpc or document.
<soap:operation ... >		Extends operation definition.
	soapAction="uri" (0 or 1)	URN.
	style="rpc document" (0 or 1)	See binding level.
<soap:body ... >		Extends operation definition; specifies how message parts appear inside the SOAP body.
	parts="nmtokens"	Optional; allows externalizing message parts.
	use="encoded literal"	literal: messages reference concrete XSD (no WSDL type); encoded: messages reference abstract WSDL type elements; encodingStyle extension used.
	encodingStyle="uri-list" (0 or 1)	List of supported message encoding styles.
	namespace="uri" (0 or 1)	URN of the service.
<soap:fault ... >		Extends operation definition; contents of fault details element.
	name="nmtoken"	Relates soap:fault to wsd1:fault for operation.
	use, encodingStyle, namespace	See soap:body.
<soap:address ... >		Extends port definition.
	location="uri"	Network address of RPC router.
<soap:header ... >		Operation level; shaped after <soap:body ... >.
<soap:headerfault ... >		Operation level; shaped after <soap:body ... >.

1.7 Summary

We began by discussing service-oriented architectures and how Web services relate to SOAs. We continued by giving an overview of the major technologies that make Web services possible: XML, SOAP, WSDL, and UDDI. We looked in detail at SOAP, which provides an XML text-based, platform- and language-neutral message format. Finally, we explained how WSDL defines the application data to be conveyed in the SOAP message as well as the information required to access the service, such as the transport protocol used and the location of the service.

Archived

Archived



CICS implementation of Web services

In this chapter, we discuss how CICS implements Web services. We begin by reviewing the support for SOAP that CICS provided in the SOAP for CICS feature with CICS TS V2. Then we discuss the Web services support of CICS TS V3.1 and the enhancements provided with CICS TS V3.2.

We continue by showing you how to prepare to run a CICS application as a service provider and what happens inside CICS when a service request arrives for a service provider application. Likewise, we discuss preparing to run a CICS application as a service requester and how CICS processes the outbound service request. This leads us to a discussion of the resource definitions that support Web services, namely, the URIMAP, PIPELINE, and WEBSERVICE definitions.

We conclude the chapter by introducing the tools associated with designing and developing Web service applications in CICS.

2.1 SOAP support in CICS TS V2

In 2003, IBM delivered its first support for SOAP in the CICS product when it announced that CICS SupportPac CA1M was available for use with CICS TS V1.3 and CICS TS V2.2. SupportPac CA1M was only intended to be a preview of how CICS might support SOAP. When IBM delivered CICS TS V2.3, it also delivered the SOAP for CICS feature for use in both CICS TS V2.2 and V2.3. Both the SupportPac CA1M and the SOAP for CICS feature implement a pipeline approach to processing SOAP messages.

2.1.1 Pipelines

A *pipeline* is a sequence of programs arranged so that the output from one program is used as input to the next. The SOAP for CICS feature consists of a pipeline that supports both service provider and service requester applications.

A service provider pipeline is a pipeline that consists of user-provided and system-provided programs. They receive an inbound SOAP message, process the contents, and send a response.

A service requester pipeline is a pipeline that consists of user-provided and system-provided programs that send an outbound SOAP message. They receive the response and process the contents of the response.

Figure 2-1 shows a service provider pipeline.

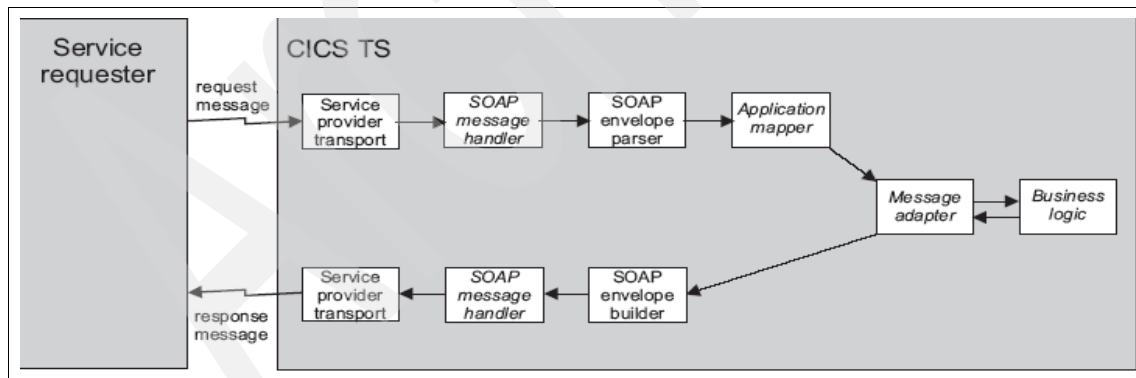


Figure 2-1 Service provider pipeline

The processing stages shown in Figure 2-1 on page 36 are as follows:

- ▶ The service provider transport stage (inbound)

This pipeline stage is responsible for extracting a SOAP request from an incoming message. The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.
- ▶ The SOAP message handler (user-written)

You can use this program to work with the complete SOAP input message. For example, you can extract information from the message or modify its contents.
- ▶ The SOAP envelope parser
This pipeline stage extracts information from the SOAP request envelope and removes the envelope from the incoming message.
- ▶ The application mapper (user-written)

Use this program to specify the name of the message adapter program that is to be invoked for an inbound SOAP request.
- ▶ The message adapter (user-written)

The message adapter is the interface between the pipeline and the business logic. The program:

 - a. Parses the SOAP request body
 - b. Invokes the business logic (typically, by using an EXEC CICS LINK command with a COMMAREA)
 - c. Constructs the SOAP response
- ▶ The SOAP envelope builder
This pipeline stage builds the SOAP response envelope and adds it to the outgoing message.
- ▶ The SOAP message handler (user-written)

You can use this program to work with the complete SOAP output message. For example, you can add SOAP headers to the output message.
- ▶ The service provider transport stage (outbound)

This pipeline stage is responsible for packaging a SOAP response within an outgoing message. The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.

2.1.2 Limitations

The SOAP for CICS feature has the following limitations:

- ▶ You can use only one pipeline for service provider applications, and one for service requester applications.
- ▶ You can define only one message handler per CICS region.
- ▶ XML parsing and generation of the SOAP body must be either user-written, or generated by a tool such as WebSphere Studio Enterprise Edition (WSED), or the more recent WebSphere Developer for System z®.
- ▶ It supports only Version 1.1 of the SOAP protocol.
- ▶ It does not support the WS-Security and WS-Atomic Transaction specifications.

2.2 Support for Web services in CICS TS V3

Application programs running in CICS TS V3 can participate in a heterogeneous Web services environment as service requesters, service providers, or both, using either an HTTP transport or an MQ transport.

2.2.1 What is provided in CICS TS V3.1

CICS TS V3.1 provides the following capabilities:

- ▶ It includes a Web services assistant utility.

The Web services assistant utility contains two programs: DFHWS2LS and DFHLS2WS.

- DFHWS2LS helps you map an existing WSDL document into a high level programming language data structure.
- DFHLS2WS creates a new WSDL document from an existing language structure.

The Web services assistant supports the following programming languages:

- COBOL
- PL/I
- C
- C++

- ▶ It supports two different approaches to deploying your CICS applications in a Web services environment.

- You can use the Web services assistant.

The Web services assistant helps you deploy an application with the least amount of programming effort. For example, if you want to expose an existing application as a Web service, you can start with a high-level language data structure and use DFHLS2WS to generate the Web services description. Alternatively, if you want to communicate with an existing Web service, you can start with its Web service description and use DFHWS2LS to generate a high level language structure that you can use in your program.

Both DFHLS2WS and DFHWS2LS also generate a file called the wsbind file. When your application runs, CICS uses the wsbind file to transform your application data into a SOAP message on output and to transform the SOAP message to application data on input.

- You can take complete control of the processing of your data.

You can write your own code to map between your application data and the message that flows between the service requester and provider. For example, if you want to use non-SOAP messages within the Web service infrastructure, you can write your own code to transform between the message format and the format used by your application.

- ▶ It reads a pipeline configuration file created by the CICS system programmer to determine which message handlers should be invoked in a pipeline.

Note: A pipeline can be configured as either a service requester pipeline or a service provider pipeline but not both.

Whether you use the Web services assistant or take complete control of the processing yourself, you can write your own message handlers to perform additional processing on your request and response messages. You can also use CICS-supplied message handlers.

- ▶ It supplies message handlers designed especially to help you process SOAP messages.

The pipelines that CICS uses to process Web service requests and responses are generic, in that there are few restrictions on what processing can be performed in each message handler. However, many Web service applications use SOAP messages, and any processing of those messages should comply with the SOAP specification. Therefore, CICS provides special SOAP message handler programs that can help you to configure your pipeline as a SOAP node.

- A service requester pipeline is the initial SOAP sender for the request, and the ultimate SOAP receiver for the response.
- A service provider pipeline is the ultimate SOAP receiver for the request, and the initial SOAP sender for the response.

You cannot configure a CICS pipeline to function as an intermediary node in a SOAP message path.

The CICS-provided SOAP message handlers can be configured to invoke one or more user-written header processing programs and to enforce the presence of particular headers in the SOAP message.

- ▶ It allows you to configure many different pipelines.

You can configure a pipeline to support SOAP 1.1 or SOAP 1.2. Within your CICS system, you can have some pipelines that support SOAP 1.1 and others that support SOAP 1.2.

- ▶ It provides the following resource definitions to help you configure your support for Web services:
 - PIPELINE
 - URIMAP
 - WEBSERVICE

If you used the SOAP for CICS feature, you might be able to use CICS resource definitions to replace the logic you provided in your pipeline programs to distinguish one application from another. For example, in a service provider, you might be able to replace code that distinguishes between applications based upon a URI, with a suitable set of URIMAP resources.

- ▶ It provides the following EXEC CICS application programming interface (API) commands:
 - SOAPFAULT ADD | CREATE | DELETE
 - INQUIRE WEBSERVICE
 - INVOKE WEBSERVICE

- ▶ It conforms to open standards, including:
 - SOAP 1.1 and 1.2
 - HTTP 1.1
 - WSDL 1.1
- ▶ It ensures maximum interoperability with other Web services implementations by conforming to the Web Services Interoperability Organization (WS-I) Basic Profile 1.0.
- ▶ It supports the WS-Atomic Transaction and WS-Security specifications.

2.2.2 What is new in CICS TS V3.2

CICS TS V3.2 provides the following new functions:

- ▶ It supports the WSDL 2.0 specification in addition to WSDL 1.1. CICS support for WSDL 2.0, however, is subject to some restrictions that are documented in the *CICS Web Services Guide*, SC34-6838.

The Web services assistant utilities, DFHWS2LS and DFHLS2WS, have been enhanced to enable creation of a Web service description that complies with WSDL 2.0. You can create both WSDL 1.1 and WSDL 2.0 documents during the same run of the assistant utility.

The batch jobs have new and modified parameters that provide you with more flexibility:

- You can specify an absolute URI for your Web service to DFHLS2WS.
- DFHWS2LS automatically determines the WSDL version of the Web service description that has been supplied as input.
- You can select a specific wsdl:Service element within the Web service description.
- You can specify a subset of wsdl:Operation elements that you want to invoke.

These enhancements are useful when you have a large WSDL file with many wsdl:Service and wsdl:Operation elements in it.

- ▶ In accordance with the WSDL 2.0 specification, CICS now supports four out of the eight Message Exchange Patterns (MEPs). These patterns describe typical interactions between requester and provider. The patterns are:
 - In-Only

A request message is sent to the Web service provider, but the provider is not allowed to send any type of response to the Web service requester.

- In-Out

A request message is sent to the Web service provider, and a response message is returned. The response message can be a normal SOAP message or a SOAP fault.

- In-Optional-Out

A request message is sent to the Web service provider, and a response message is optionally returned to the requester. The response message can be a normal SOAP message, or a SOAP fault.

- Robust In-Only

A request message is sent to the Web service provider, and no response message is returned to the requester unless an error occurs. In this case, a SOAP fault message is sent to the requester.

- ▶ When CICS is acting as a service requester, that is, if your program issues an EXEC CICS INVOKE WEBSERVICE command, you can now define how long CICS should wait for a reply. The PIPELINE resource has a new RESPWAIT attribute that determines how many seconds CICS should wait. If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is used.

The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

- ▶ New context containers have been provided in the Web service provider and requester pipelines to support WSDL 2.0.
- ▶ CICS TS V3.2 supports the SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) specification commonly referred to as MTOM/XOP. These specifications define a method for optimizing the transmission of base64Binary data within SOAP messages.

CICS implements this support in both requester and provider pipelines when the transport protocol is HTTP or HTTPS. You can configure MTOM/XOP support by using additional options in the pipeline configuration file. With this support, base64Binary data is sent as a binary attachment and not directly in the SOAP message.

- ▶ CICS support for Web services has been extended to comply with the WSDL 1.1 Binding Extension for SOAP 1.2 specification. This specification defines the binding extensions that are required to indicate that Web service messages are bound to the SOAP 1.2 protocol. The goal is to provide functionality that is comparable with the binding for SOAP 1.1.
- ▶ The support that CICS provides for securing Web services has been enhanced to include an implementation of the Web Services Trust Language (WS-Trust) specification. CICS TS V3.2 can interoperate with a Security Token Service (STS), such as Tivoli® Federated Identity Manager, to validate and issue security tokens in Web services. This enables CICS to send and receive messages that contain a wide variety of security tokens, such as SAML assertions and Kerberos tokens, to interoperate securely with other Web services. CICS support for WS-Trust specification is subject to some restrictions, as shown in the *CICS Web Services Guide*, SC34-6838.

2.2.3 Comparing CICS TS V3 with the SOAP for CICS feature

Table 2-1 summarizes some of the differences between the support for Web services in CICS TS V3.1 and CICS TS V3.2, and the SOAP support in the SOAP for CICS feature.

Table 2-1 Comparison of CICS TS V3 Web services with SOAP for CICS feature

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
Pipeline data passing mechanism	Channels and containers	Channels and containers	BTS containers
Number of pipelines	Multiple per CICS region	Multiple per CICS region	One service requester pipeline per CICS region One service provider pipeline per CICS region
Number of message handlers	Multiple per each Web service definition	Multiple per each Web service definition	One per CICS region

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
Pipeline container names	<ul style="list-style-type: none"> ▶ DFHWS-APPHANDLE R ▶ DFHWS-BODY ▶ DFHWS-DATA ▶ DFHWS-OPERATION ▶ DFHWS-PIPELINE ▶ DFHWS-SOAPACTION ▶ DFHWS-SOAPLEVEL ▶ DFHWS-TRANID ▶ DFHWS-URI ▶ DFHWS-USERID ▶ DFHWS-WEBSERVICE ▶ DFHWS-XMLNS ▶ DFHERROR ▶ DFHFUNCTION ▶ DFHHEADER ▶ DFHNORESPONSE ▶ DFHREQUEST ▶ DFHRESPONSE ▶ DFH-HANDLERPLIST ▶ DFH-SERVICEPLIST ▶ DFHWS-MEP ▶ DFHWS-CID-DOMAIN ▶ DFHWS-MTOM-IN ▶ DFHWS-MTOM-OUT ▶ DFHWS-XOP-IN ▶ DFHWS-XOP-OUT ▶ DFHWS-IDTOKEN ▶ DFHWS-RESTOKEN ▶ DFHWS-SERVICEURI ▶ DFHWS-STSACTION ▶ DFHWS-STSFault ▶ DFHWS-STSURi ▶ DFHWS-TOKENTYPE 	<ul style="list-style-type: none"> ▶ DFHWS-APPHANDLE R ▶ DFHWS-BODY ▶ DFHWS-DATA ▶ DFHWS-OPERATION ▶ DFHWS-PIPELINE ▶ DFHWS-SOAPACTION ▶ DFHWS-SOAPLEVEL ▶ DFHWS-TRANID ▶ DFHWS-URI ▶ DFHWS-USERID ▶ DFHWS-WEBSERVICE ▶ DFHWS-XMLNS ▶ DFHERROR ▶ DFHFUNCTION ▶ DFHHEADER ▶ DFHNORESPONSE ▶ DFHREQUEST ▶ DFHRESPONSE ▶ DFH-HANDLERPLIST ▶ DFH-SERVICEPLIST 	<ul style="list-style-type: none"> ▶ APP-HANDLER ▶ APP-NAMESPACES ▶ FAULT ▶ INPUT ▶ NAMESPACES ▶ OUTPUT ▶ PIPELINE-ERROR ▶ REQUEST-BODY ▶ RESPONSE-BODY ▶ SOAP-ACTION ▶ TARGET-TRANID ▶ TARGET-URI ▶ TARGET-USERID ▶ USER-CONTAINERS
SOAP protocol level	SOAP 1.1 and 1.2	SOAP 1.1 and 1.2	SOAP 1.1
CICS resource definitions	<ul style="list-style-type: none"> ▶ PIPELINE ▶ URIMAP ▶ WEBSERVICE 	<ul style="list-style-type: none"> ▶ PIPELINE ▶ URIMAP ▶ WEBSERVICE 	None

Description	CICS TS V3.2	CICS TS V3.1	SOAP for CICS feature
CICS API and SPI	<ul style="list-style-type: none"> ▶ CREATE PIPELINE ▶ CREATE URIMAP ▶ CREATE WEBSERVICE ▶ INQUIRE WEBSERVICE ▶ INVOKE WEBSERVICE ▶ PERFORM PIPELINE SCAN ▶ SET WEBSERVICE ▶ SOAPFAULT ADD ▶ SOAPFAULT CREATE ▶ SOAPFAULT DELETE 	<ul style="list-style-type: none"> ▶ CREATE PIPELINE ▶ CREATE URIMAP ▶ CREATE WEBSERVICE ▶ INQUIRE WEBSERVICE ▶ INVOKE WEBSERVICE ▶ PERFORM PIPELINE SCAN ▶ SET WEBSERVICE ▶ SOAPFAULT ADD ▶ SOAPFAULT CREATE ▶ SOAPFAULT DELETE 	none
XML parsing	CICS wsbind file generated by either CICS Web service assistant or WebSphere Developer for System z	CICS wsbind file generated by either CICS Web service assistant or WebSphere Developer for System z	<p>WebSphere Developer for System z</p> <p>WebSphere Studio Enterprise Developer (WSED)</p> <p>Write your own using Enterprise COBOL</p>
WSDL support	1.1 and 2.0	1.1	1.1
WS-Atomic support	Yes	Yes	None
WS-Security support	Yes	Yes	None
MTOM/XOP support	Yes	none	None
WS-Trust support	Yes	None	None

2.3 CICS as a service provider

When CICS is a service provider, it receives a service request, which is passed through a pipeline to a target application program. The response from the application is returned to the service requester through the same pipeline. In this section, we first discuss how to prepare for running a CICS application as a service provider. Then we discuss how CICS processes the incoming service request.

2.3.1 Preparing to run a CICS application as a service provider

Suppose that we have an existing CICS application that we wish to expose as a Web service that uses the HTTP transport. Suppose also that we wish to use the Web services assistant rather than taking control of the processing ourselves.

In this section, we describe a technique that is suitable for a development environment. We allow CICS to autoinstall the URIMAP and WEBSERVICE definitions by scanning the pickup directory of the pipeline. In a production environment, you might prefer to keep tighter control on your URIMAP and WEBSERVICE resource definitions and use hardcoded definitions, as we describe in 2.6.1, “URIMAP” on page 57 and 2.6.3, “WEBSERVICE” on page 65.

We go through the following steps:

1. We generate the wsbind and WSDL files (application developer).
 - a. We first create an HFS directory in which to store the generated files. For example, we might create a directory named `/u/SharedProjectDirectory/MyFirstWebServiceProvider`.
 - b. We next run the DFHLS2WS program. The input we provide to the program includes the following information:
 - The names of the partitioned data set members that contain the high level language structures that the application program uses to describe the Web service request and the Web service response.
 - The fully qualified HFS names of the wsbind file and of the file into which the Web service description is to be written (the WSDL file).
 - The relative URI that a client uses to access the Web service.
 - How CICS should pass data to the target application program (COMMAREA or container).

Typically, an application developer would perform this step.

2. Create a TCPIPSERVICE resource definition (system programmer).

The resource definition should specify PROTOCOL(HTTP) and supply information about the port on which inbound requests are received.

Typically, a system programmer would perform this step.
3. Create a PIPELINE resource definition (system programmer).
 - a. Create a service provider pipeline configuration file.

A pipeline configuration file is an XML file that describes, among other things, the message handler programs and the SOAP header processing programs that CICS invokes when it processes the pipeline.
 - b. Create an HFS directory in which to store installable wsbind and WSDL files.

We call this directory the “pickup” directory since CICS picks up the wsbind and WSDL files from this directory and store them on a “shelf” directory.
 - c. Create an HFS directory for CICS in which to store installed wsbind files.

We call this directory the “shelf” directory.
 - d. Create a PIPELINE resource definition to handle the Web service request.
 - Specify the CONFIGFILE attribute to point to the file created in step 3a.
 - Specify the WSDIR attribute to point to the directory created in step 3b.
 - Specify the SHELF attribute to point to the directory created in step 3c.
 - e. Copy the wsbind and WSDL files created in step 1 to the pickup directory created in step 3b.
4. Install the TCPIPSERVICE and PIPELINE resource definitions (system programmer).

When the CICS system programmer installs the PIPELINE definition, CICS scans the pickup directory for wsbind files. When CICS finds the wsbind file created in step 1, CICS dynamically creates and installs a WEBSERVICE resource definition for it. CICS derives the name of the WEBSERVICE definition from the name of the wsbind file. The WEBSERVICE definition identifies the name of the associated PIPELINE definition and points to the location of the wsbind file in the HFS.

During the installation of the WEBSERVICE resource:

 - CICS dynamically creates and installs a URIMAP resource definition. CICS bases the definition on the URI specified in the input to DFHLS2WS (see step 1) and stored by DFHLS2WS in the wsbind file.

- CICS uses the wsbind file to create main storage control blocks to map the inbound service request (XML) to a COMMAREA or a container and to map to XML the outbound COMMAREA or container that contains the response data.
5. Publish WSDL to clients.
 - a. Customize the location attribute on the <address> element in the WSDL file so that its value specifies the TCP/IP server name of the machine hosting the service and the port number defined in step 2.
 - b. Publish the WSDL to any parties wishing to create clients to this Web service.

2.3.2 Processing the inbound service request

Figure 2-2 shows the processing that occurs when a service requester sends a SOAP message over HTTP to a service provider application running in a CICS TS V3 region.

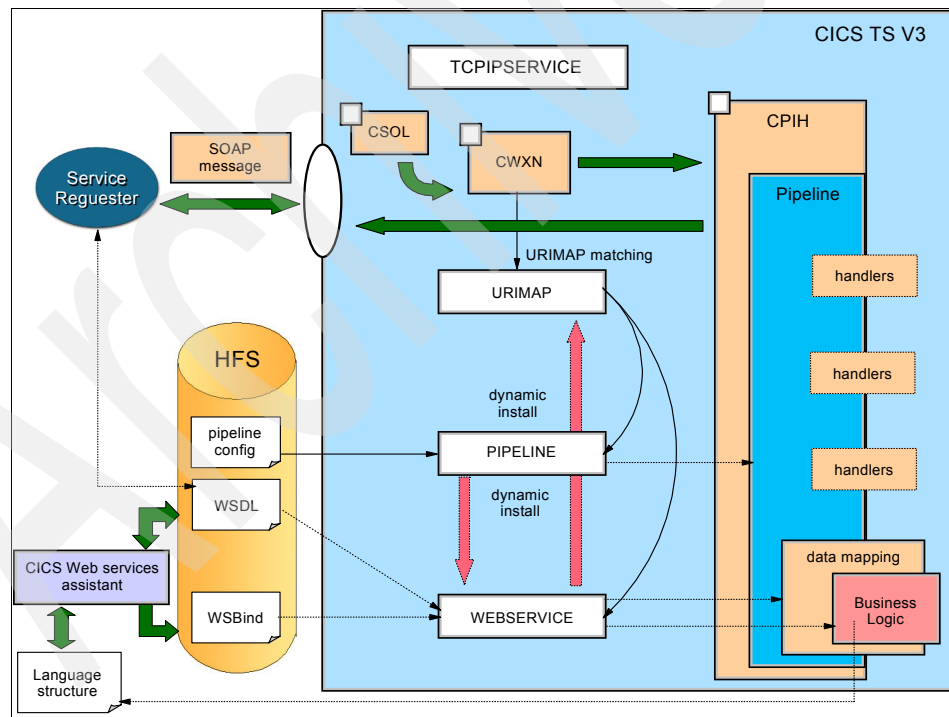


Figure 2-2 Web service runtime service provider processing

The CICS-supplied sockets listener transaction (CSOL) monitors the port specified in the TCPIP SERVICE resource definition for incoming HTTP requests. When the SOAP message arrives, CSOL attaches the transaction specified in the TRANSACTION attribute of the TCPIP SERVICE definition; by default, this is the CICS-supplied Web attach transaction CWXN.

CWXN finds the URI in the HTTP request and then scans the URIMAP resource definitions for a URIMAP that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the HTTP request. If CWXN finds such a URIMAP, it uses the PIPELINE and WEBSERVICE attributes of the URIMAP definition to get the name of the PIPELINE and WEBSERVICE definitions that it uses to process the incoming request. CWXN also uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; by default, this is the CPIH transaction.

CPIH starts the pipeline processing. It uses the PIPELINE definition to find the name of the pipeline configuration file. CPIH uses the pipeline configuration file to determine which message handler programs and SOAP header processing programs to invoke.

A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) removes the SOAP envelope from the inbound request and passes the SOAP body to the data mapper function.

CICS uses the DFHWS-WEBSERVICE container to pass the name of the required WEBSERVICE definition to the data mapper. The data mapper uses the WEBSERVICE definition to locate the main storage control blocks that it requires to map the inbound service request (XML) to a COMMAREA or a container.

The data mapper links to the target service provider application program, providing it with input in the format that it expects. The application program is not aware that it is being executed as a Web service. The program performs its normal processing and then returns an output COMMAREA or container to the data mapper.

The output data from the CICS application program cannot just be sent back to the pipeline code. The data mapper must first convert the output from the COMMAREA or container format into a SOAP body.

2.4 CICS as a service requester

When CICS is a service requester, an application program sends a request, which is passed through a pipeline, to a target service provider. The response from the service provider is returned to the application program through the same pipeline. In this section, we first discuss how to prepare to run a CICS application as a service requester. Then we discuss how CICS processes the outbound service request.

2.4.1 Preparing to run a CICS application as a service requester

Suppose we wish to write a new CICS application that invokes a Web service. Suppose also that we wish to use the Web services assistant rather than taking control of the processing ourselves.

In this section, we describe a technique that is suitable for a development environment, that is, we let CICS autoinstall WEBSERVICE definitions by scanning the pickup directory of the pipeline. In a production environment, you might prefer to keep tighter control of your WEBSERVICE resource definitions and use hardcoded definitions, as we describe in 2.6.3, “WEBSERVICE” on page 65.

We go through the following steps:

1. Generate the wsbind file and the language structures (application developer).
 - a. We first create an HFS directory in which to store the wsbind file. For example, we might create a directory named `/u/SharedProjectDirectory/MyFirstWebServiceRequester`.
 - b. We next run the DFHWS2LS program. The input we provide to the program includes the following information:
 - The fully qualified HFS name of the WSDL file that describes the Web service that we want to request.
 - The names of the partitioned data set members into which DFHWS2LS should put the high level language structures that it generates. The application program uses the language structures to describe the Web service request and the Web service response.
2. Create a PIPELINE resource definition (system programmer).
 - a. Create a service requester pipeline configuration file.

A pipeline configuration file is an XML file that describes, among other things, the message handler programs and the SOAP header processing programs that CICS invokes when it processes the pipeline.

- b. Create an HFS directory in which to store installable wsbind files.
We call this directory the “pickup” directory, since CICS picks up the wsbind file from this directory and store it on a “shelf” directory.
 - c. Create an HFS directory for CICS in which to store installed wsbind files.
We call this directory the “shelf” directory.
 - d. Create a PIPELINE resource definition to handle the Web service request.
 - Specify the CONFIGFILE attribute to point to the file created in step 2a.
 - Specify the WSDIR attribute to point to the directory created in step 2b.
 - Specify the SHELF attribute to point to the directory created in step 2c.
 - e. Copy the wsbind file created in step 1 to the pickup directory created in step 2b.
3. Install the PIPELINE resource definition (system programmer).

When the CICS system programmer installs the PIPELINE definition, CICS scans the pickup directory for wsbind files. When CICS finds the wsbind file created in step 1, CICS dynamically creates and installs a WEBSERVICE resource definition for it. CICS derives the name of the WEBSERVICE definition from the name of the wsbind file. The WEBSERVICE definition identifies the name of the associated PIPELINE definition and points to the location of the wsbind file in the HFS.

During the installation of the WEBSERVICE resource, CICS uses the wsbind file to create main storage control blocks to map the outbound service request to an XML document and to map the inbound XML response document to a language structure.

4. Use the language structure generated in step 1 to write the application program (application developer).
 - a. It issues the following command to place the outbound data into container DFHWS-DATA:

```
EXEC CICS PUT CONTAINER(DFHWS-DATA) CHANNEL(name_of_channel)  
FROM(data_area)
```
 - b. It issues the following command to invoked the Web service:

```
EXEC CICS INVOKE WEBSERVICE(name_of_WEBSERVICE_definition)  
CHANNEL(name_of_channel) OPERATION(name_of_operation)
```

2.4.2 Processing the outbound service request

Figure 2-3 shows the processing that occurs when a service requester running in a CICS TS V3 region sends a SOAP message to a service provider.

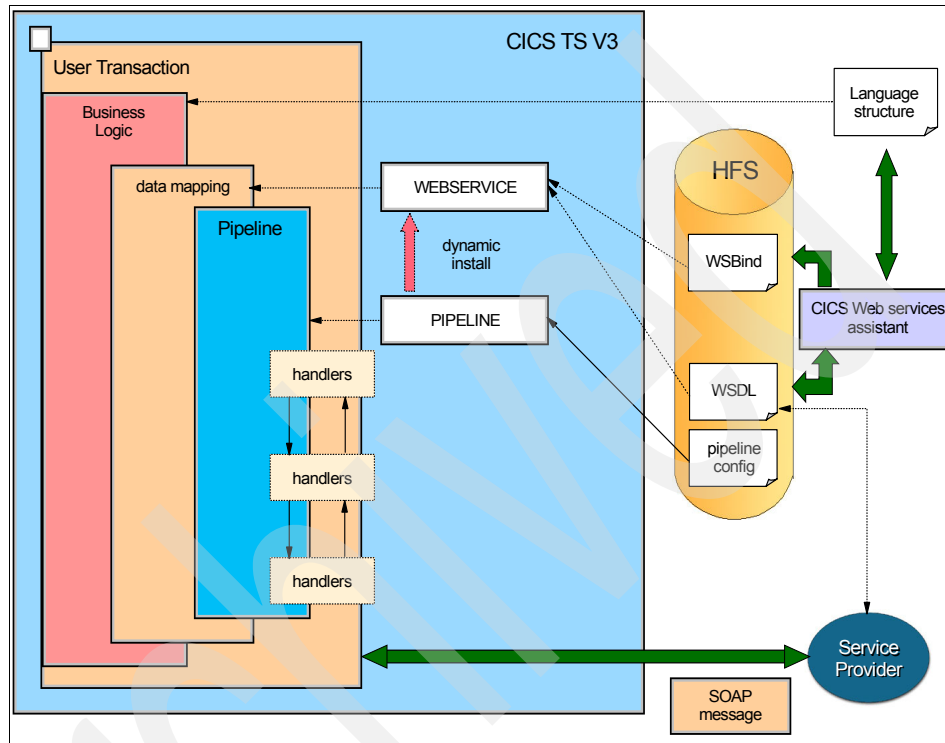


Figure 2-3 Web service requestor resources

When the service requester issues the EXEC CICS INVOKE WEBSERVICE command, CICS uses the information found in the wsbind file that is associated with the specified WEBSERVICE definition to convert the language structure into an XML document. CICS then invokes the message handlers specified in the pipeline configuration file, and they convert the XML document into a SOAP message.

CICS sends the request SOAP message to the remote service provider either through HTTP or WebSphere MQ.

When the response SOAP message is received, CICS passes it back through the pipeline. The message handlers extract the SOAP body from the SOAP envelope, and the data mapping function converts the XML in the SOAP body into a language structure that is passed to the application program in container DFHWS-DATA.

When CICS TS V3.2 is acting as a service requester, that is, if your program issues an INVOKE WEBSERVICE command, you can now define how long CICS waits for a reply. The PIPELINE resource has a new RESPWAIT attribute that determines how many seconds CICS waits. If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is used.

The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

Figure 2-4 shows a CICS TS V3.2 region acting as both a service provider and service requester.

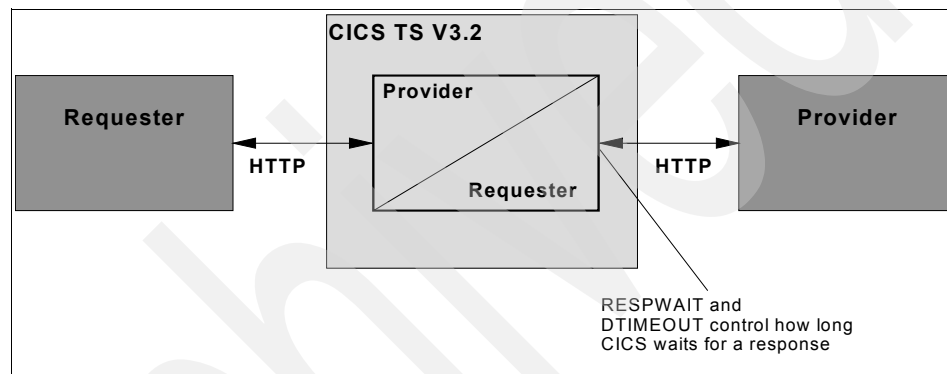


Figure 2-4 CICS TS V3.2 region as a service provider and a service requester

2.5 Catalog manager example application

The CICS catalog manager example application is a COBOL application that is designed to illustrate best practices when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the user can perform these functions:

- ▶ List the items in a catalog.
- ▶ Inquire on individual items in the catalog.
- ▶ Order items from the catalog.

The base application has a 3270 interface. Figure 2-5 shows the structure of the base application.

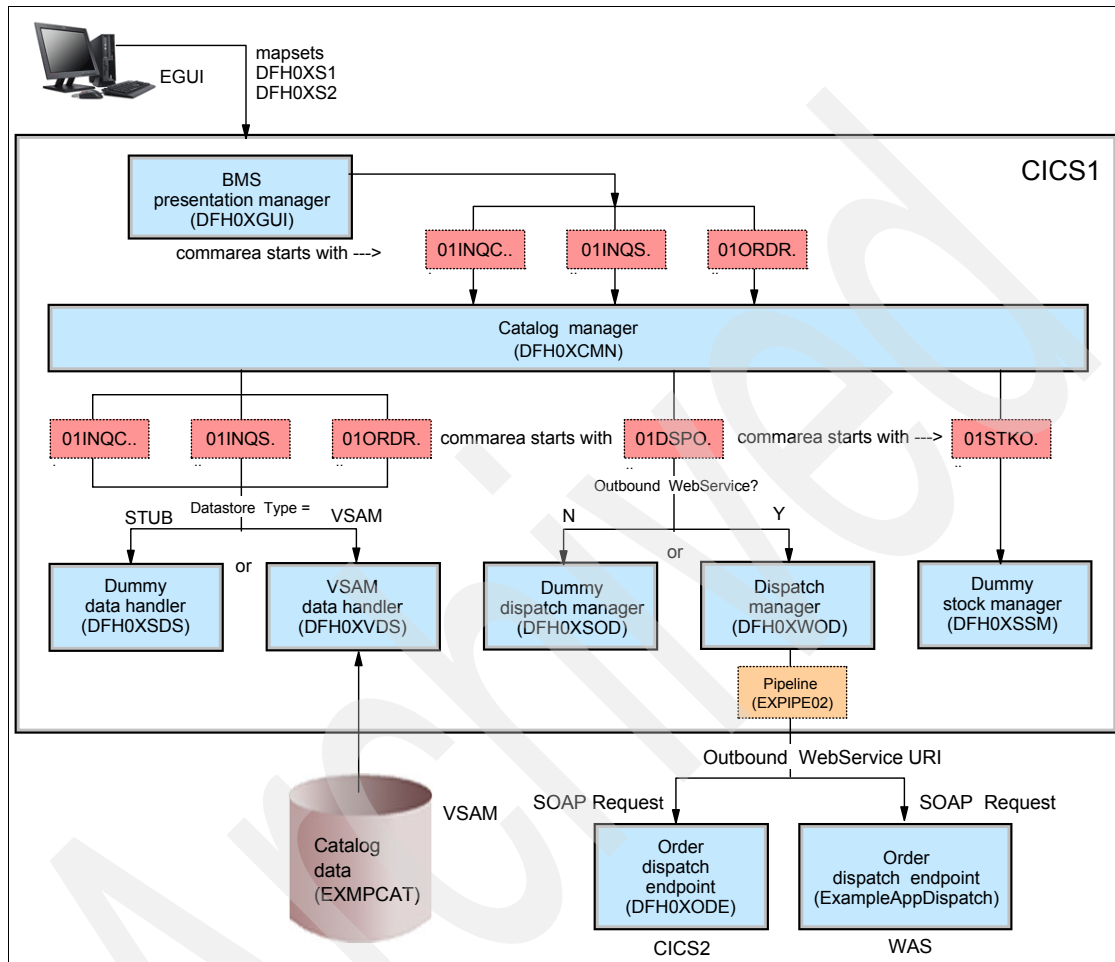


Figure 2-5 Basic catalog manager application structure

The business logic can also be invoked as a Web service in three different ways:

1. By a CICS Web services client program invoked from a 3270 screen
2. By a WebSphere Application Server client application invoked by a browser with direct invocation of the Catalog manager program DFH0XCMN
3. By a WebSphere Application Server application invoked by a browser with three different front-end programs that serve as wrappers for DFH0XCMN

Figure 2-6 shows the structure of the Web service provider application.

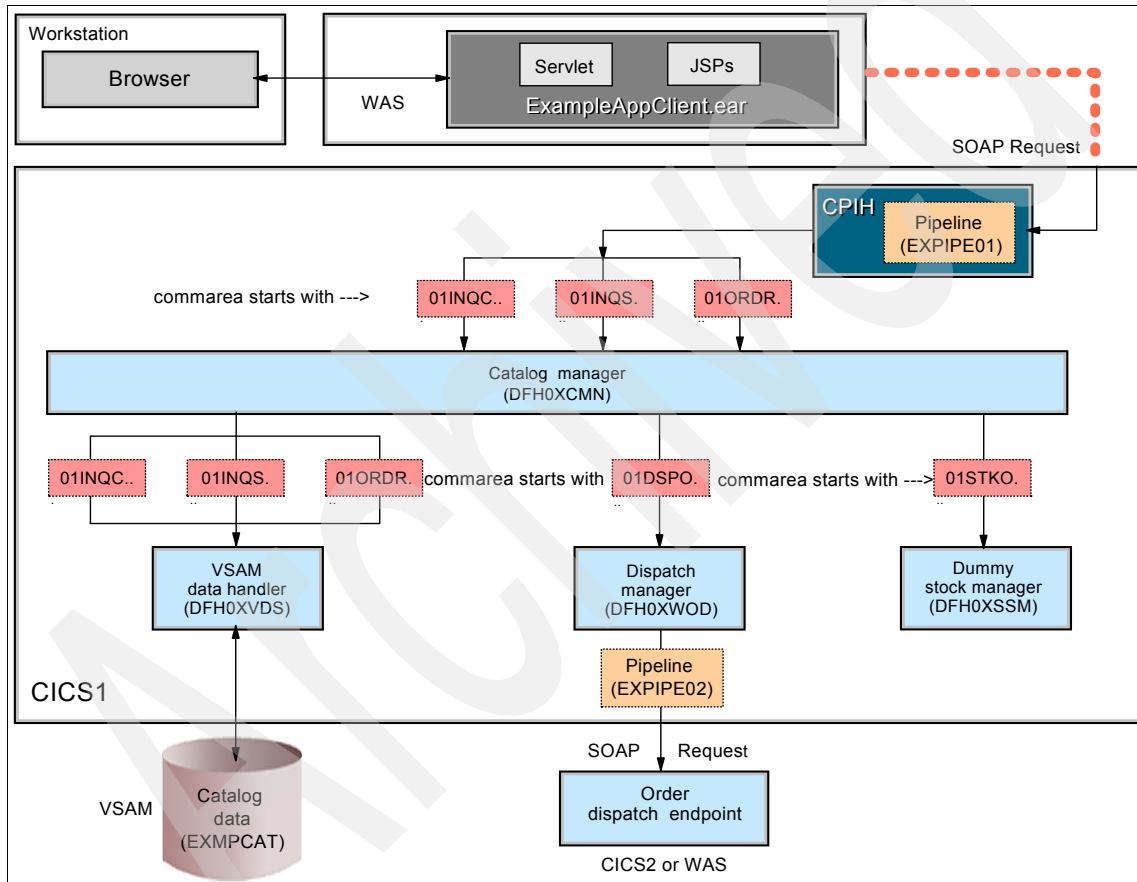


Figure 2-6 Structure of Web service provider application

You configure the sample application with an ECFG transaction. Example 2-1 shows a configuration screen of the ECFG transaction.

Example 2-1 Catalog application configuration screen

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

      Datastore Type ==> VSAM          STUB|VSAM
Outbound WebService? ==> NO          YES|NO
      Catalog Manager ==> DFHOXCMN
      Data Store Stub ==> DFHOXSDS
      Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> DFHOXWOD
      Stock Manager ==> DFHOXSSM
      VSAM File Name ==> EXMPCAT
Server Address and Port ==>
Outbound WebService URI ==> http://9.42.170.163:9080/exampleApp/services
                        ==> /dispatchOrderPort
                        ==>
                        ==>
                        ==>
                        ==>

APPLICATION CONFIGURATION UPDATED
PF          3  END                                12  CNCL
```

For the complete description of the catalog manager application and its setup, refer to the *CICS Web Services Guide*, SC34-6838.

In 3.2, “Configuring CICS as a service provider” on page 88, we explain how we deployed the catalog manager service application as a service provider application in a CICSplex configuration.

In 4.4, “Configuring CICS as service requester using WMQ” on page 133, we explain how we deployed the catalog manager application as a service requester application in a CICSplex configuration.

2.6 CICS TS V3 Web service resource definitions

In this section, we provide Web service resource definitions for CICS TS V3.

2.6.1 URIMAP

URIMAP definitions are used to provide three different Web-related facilities in CICS. It is the value of the USAGE attribute on a URIMAP definition that determines which of the three facilities that particular definition provides:

1. Requests from a Web client, to CICS as an HTTP server:

URIMAP definitions for requests for CICS as an HTTP server have a USAGE attribute of SERVER. These URIMAP definitions match the URLs of HTTP requests that CICS expects to receive from a Web client, and they define how CICS should provide a response to each request. You can use a URIMAP definition to tell CICS to:

- Provide a *static* response to the HTTP request, using a document template or z/OS® UNIX® System Services HFS file.
- Provide a *dynamic* response to the HTTP request, using an application program that issues EXEC CICS WEB application programming interface commands.
- Redirect the request to another server, either temporarily or permanently.

For CICS as an HTTP server, URIMAP definitions incorporate most of the functions that were previously provided by the analyzer program specified on the TCPIP SERVICE definition. An analyzer program might still be involved in the processing path if required.

2. Requests to a server, from CICS as an HTTP client:

URIMAP definitions for requests from CICS as an HTTP client have a USAGE attribute of CLIENT. These URIMAP definitions specify URLs that are used when a user application, acting as a Web client, makes a request through CICS Web support to an HTTP server. Setting up a URIMAP definition for this purpose means that you can avoid identifying the URL in your application program.

3. Web service requests:

URIMAP definitions for Web service requests have a USAGE attribute of PIPELINE. These URIMAP definitions associate a URI for an inbound Web service request (that is, a request by which a client invokes a Web service in CICS) with a PIPELINE or WEBSERVICE resource that specifies the processing to be performed. They might also be used to specify:

- The name of the transaction that CICS should use to run the pipeline.
- The user ID under which the pipeline transaction runs.

Figure 2-2 on page 48 illustrates the purpose of a URIMAP definition for mapping Web service requests. In this book, we only concern ourselves with this third use of the URIMAP definition.

You can create URIMAP resource definitions in the following ways:

- ▶ Use the CEDA transaction.
- ▶ Use the DFHCSDUP batch utility.
- ▶ Use CICSplex® SM Business Application Services.
- ▶ Use the EXEC CICS CREATE URIMAP command.

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory) and creates URIMAP and WEBSERVICE resources dynamically. For each Web service binding file in the directory, that is, for each file with the wsbind suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is more recent than the existing resources.

Whether you install URIMAP resource definitions dynamically or whether you define them manually, it is possible to change the transaction ID of the pipeline processing transaction.

2.6.2 PIPELINE

A PIPELINE resource definition provides information about the message handlers that act on a service request and on the response. The information about the message handlers is supplied indirectly; the PIPELINE definition specifies the name of an HFS file, called the pipeline configuration file, which contains an XML description of the message handlers and their configuration.

Figure 2-2 on page 48 and Figure 2-3 on page 52 illustrate the purpose of the PIPELINE resource definition for service provider and service requester pipelines, respectively.

The most important attributes of the PIPELINE definition are as follows:

▶ WSDIR

The WSDIR attribute specifies the name of the Web service binding directory (also known as the pickup directory). The Web service binding directory contains Web service binding files that are associated with the PIPELINE, and that are to be installed automatically by the CICS scanning mechanism. When the PIPELINE definition is installed, CICS scans the directory and automatically installs any Web service binding files it finds there.

If you specify a value for the WSDIR attribute, it must refer to a valid HFS directory to which the CICS region has at least read access. If this is not the case, any attempt to install the PIPELINE resource fails.

If you do not specify a value for WSDIR, no automatic scan takes place upon installation of the PIPELINE, and PERFORM PIPELINE SCAN commands fail.

▶ SHELF

The SHELF attribute specifies the name of an HFS directory where CICS copies information about installed Web services. CICS regions into which the PIPELINE definition is installed must have full permission to the shelf directory: read, write, and the ability to create subdirectories.

A single shelf directory can be shared by multiple CICS regions and by multiple PIPELINE definitions. Within a shelf directory, each CICS region uses a separate subdirectory to keep its files separate from those of other CICS regions. Within each region's directory, each PIPELINE uses a separate subdirectory.

After a CICS region performs a cold or initial start, it deletes its subdirectories from the shelf before trying to use the shelf.

▶ CONFIGFILE

This attribute specifies the name of the PIPELINE configuration file.

▶ RESPWAIT

This attribute determines how many seconds CICS should wait. If you do not set a value for this attribute, either the default timeout for the transport protocol or the dispatcher timeout for the transaction is being used.

The default timeout for HTTP is 10 seconds; the default timeout for WebSphere MQ is 60 seconds. If the value for the dispatcher timeout for the transaction is less than the default for either protocol, the timeout occurs with the dispatcher.

Note: The RESPWAIT attribute only applies to service requester pipelines.

Pipeline configuration file

When CICS processes a Web service request, it uses a pipeline of one or more message handlers to handle the request. The configuration of the pipeline depends upon the requirements of the application. The configuration of a pipeline used to handle a Web service request is specified in an XML document, known as a pipeline configuration file. Use a suitable XML editor or text editor to work with your pipeline configuration files.

There are two kinds of pipeline configuration files: one describes the configuration of a service provider pipeline and the other describes the configuration of a service requester pipeline. An example configuration file for a CICS service provider pipeline is shown in Example 2-2.

Example 2-2 Configuration file

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd">
  <service>
    <service_handler_list>
      <handler>
        <program>SETTRNID</program>
        <handler_parameter_list/>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The first line of the pipeline configuration file specifies the XML version and the encoding that is being used by this pipeline. Notice that the CICS-provided `basicsoap11provider.xml` and `basicsoap11requester.xml` files specify `encoding="EBCDIC-CP-US"?`. This setting allows you to test the CICS-provided example catalog management application using the SOAP client that runs inside CICS. If you want to use the WebSphere Application Server client, you have to change the encoding in `basicsoap11provider.xml` to `"UTF-8"?`. Similarly, if you want to test the outbound option of the sample application, change the encoding in `basicsoap11requester.xml` to `"UTF-8"?`. Refer to 3.3, "Configuring CICS as a service requester" for a description of how we tested this option.

Each kind of pipeline configuration file is defined by its own schema, and each has a different root element. The root element for a provider pipeline is <provider_pipeline>, while the root element for a requester pipeline is <requester_pipeline>.

The immediate child elements of the <provider_pipeline> element are:

- ▶ A mandatory <service> element, which specifies the message handlers that are invoked for every request, including the terminal message handler. The terminal message handler is the last handler in the pipeline.

Optional message handlers (see “Message handlers” on page 62) are defined in the <handler> tag within the <service_handler_list> tag. The pipeline configuration file in Example 2-2 on page 60 defines a message handler program SETTRNID, which is invoked before the SOAP 1.1 terminal node, and thus before invoking the data conversion program (DFHPITP).

- ▶ An optional <transport> element, which specifies message handlers that are selected at runtime, based upon the resources that are being used for the message transport. For example, for the HTTP transport, you can specify that CICS should invoke the message handler only when the port on which the request was received is defined on a specific TCPIP SERVICE definition. For the WebSphere MQ transport, you can specify that CICS should invoke the message handler only when the inbound message arrives at a specific message queue.
- ▶ An optional <apphandler> element, which specifies the name of the program that the terminal message handler links to by default, that is, the name of the target application program (or wrapper program) that provides the service. Message handlers can specify a different program at runtime by using the DFHWS-APPHANDLER container, so the name coded here is not always the program to which it is linked.

When you use DFHLS2WS or DFHWS2LS to deploy your service provider, you must specify DFHPITP as the target program. DFHPITP obtains the name of your target application program (or wrapper program) from the wsbind file.

The <apphandler> element is used when the last message handler in the pipeline (the terminal handler) is one of the CICS-supplied SOAP message handlers,

If you do not code an <apphandler> element, one of the message handlers must use the DFHWS-APPHANDLER container to specify the name of the program.

- ▶ An optional <service_parameter_list> element, which contains parameters that CICS makes available to the message handlers in the pipeline through container DFH-SERVICEPLIST.

The immediate sub-elements of a <requester_pipeline> element are:

- ▶ An optional <service> element, which specifies the message handlers that are invoked for every request
- ▶ An optional <transport> element, which specifies message handlers that are selected at runtime, based upon the resources that are being used for the message transport
- ▶ An optional <service_parameter_list> element, which contains parameters that CICS makes available to the message handlers in the pipeline through container DFH-SERVICEPLIST

Message handlers

A message handler is a program that is used to process a Web service request during input, and to process the response during output. Message handlers use channels and containers to interact with one another, and with the system.

The message handler interface lets you perform the following tasks in a message handler program:

- ▶ Examine the contents of an XML request or response without changing it.
For example, a message handler that performs logging examines a message, and copy the relevant information from that message to the log. The message that is passed to the next handler is unchanged.
- ▶ Change the contents of an XML request or response.
For example, a message handler that performs encryption and decryption receives an encrypted message on input, and pass the decrypted message to the next handler. On output, it does the opposite: receive a plain text message, and pass an encrypted version to the following handler.
- ▶ In a non-terminal message handler, pass an XML request or response to the next message handler in the pipeline.
- ▶ In a terminal message handler, call an application program, and generate a response.
- ▶ In the request phase of the pipeline, force a transition to the response phase, by absorbing the request, and generating a response.
- ▶ Handle errors.

All programs that are used as message handlers are invoked with the same channel interface. The channel holds a number of containers. The containers can be categorized as:

- ▶ Control containers

These are essential to the operation of the pipeline. Message handlers can use the control containers to modify the sequence in which the message handlers are processed.

- ▶ Context containers

In some situations, message handler programs require information about the context in which they are invoked. CICS provides this information in a set of context containers that are passed to the programs. Some of the context containers hold information that you can change in your message handler. For example, in a service provider pipeline, you can change the user ID and transaction ID of the target application program by modifying the contents of the appropriate context containers DFHWS-USERID and DFHWS-TRANID.

The effect of changing the transaction ID in a message handler is that the data mapping and target application program runs in a separate task using the changed transaction ID. An example of a message handler that changes the transaction ID is provided by *Implementing CICS Web Services*, SG24-7206.

Setting different transaction IDs based on the name of the Web service, for example, can be useful for accounting and security purposes.

- ▶ User containers

These contain information that one message handler requires to pass to another. The use of user containers is entirely a matter for the message handlers.

SOAP message handlers

The SOAP message handlers are CICS-provided message handlers that you can include in your pipeline to process SOAP 1.1 and SOAP 1.2 messages. You can use the SOAP message handlers in a service requester or in a service provider pipeline.

On input, the SOAP message handlers parse inbound SOAP messages, and extract the SOAP <Body> element for use by your application program. On output, the handlers construct the complete SOAP message, using the <Body> element that your application provides.

If you use SOAP headers in your messages, the SOAP handlers can invoke user-written *header processing programs* that allow you to process the SOAP headers on inbound messages, and to add them to outbound messages.

Note: Do not confuse header processing programs with message handlers. A header processing program can only be invoked by a CICS-supplied SOAP message handler to process a specific kind of SOAP header.

Typically, you require just one SOAP message handler in a pipeline. However, there are some situations where more than one is required. For example, you can ensure that SOAP headers are processed in a particular sequence by defining multiple SOAP message handlers.

SOAP message handlers, and any header processing programs, are specified in the pipeline configuration file, using the <cics_soap_1.1_handler> and the <cics_soap_1.2_handler> elements, and their sub-elements.

An example usage of a header processing program is to process a Security header that can be passed in a WS-Security SOAP header. An example of such a header processing program is provided in *Implementing CICS Web Services*, SG24-7206.

SOAPFAULT commands

SOAP message handlers and SOAP header processing programs can use three API commands that are new in CICS TS V3 to manage SOAP faults:

▶ EXEC CICS SOAPFAULT CREATE

Use this command to create a SOAP fault. If a SOAP fault already exists in the context of the SOAP message that is being processed by the message handler, the existing fault is overwritten.

▶ EXEC CICS SOAPFAULT ADD

Use this command to add either of the following items to a SOAPFAULT object that was created with an earlier SOAPFAULT CREATE command:

- A subcode.
- A fault string for a particular national language.

If the fault already contains a fault string for the language, then this command replaces the fault string for that language. In SOAP 1.1, only the fault string for the original language is used.

▶ EXEC CICS SOAPFAULT DELETE

Use this command to delete a SOAPFAULT object that was created with an earlier SOAPFAULT CREATE command.

These commands require information that is held in containers on the channel of the CICS-supplied SOAP message handler. To use these commands, you must have access to the channel. Only the following types of programs have this access:

- ▶ Programs that are invoked directly from a CICS-supplied SOAP message handler, including SOAP header processing programs.
- ▶ Programs deployed with the Web services assistant that have a channel interface. Programs with a COMMAREA interface do not have access to the channel.

Many of the options on the SOAPFAULT CREATE and SOAPFAULT ADD commands apply to SOAP 1.1 and SOAP 1.2 faults, although their behavior is slightly different for each level of SOAP. Other options apply to one SOAP level or the other, but not to both, and if you specify any of them when the message uses a different level of SOAP, the command raises an INVREQ condition. To help you determine which SOAP level applies to the message, container DFHWS-SOAPLEVEL contains a binary fullword with one of the following values:

- ▶ 1: The request or response is a SOAP 1.1 message.
- ▶ 2: The request or response is a SOAP 1.2 message.
- ▶ 10: The request or response is not a SOAP message.

2.6.3 WEBSERVICE

Three objects define the execution environment that allows a CICS application program to operate as a Web service provider or a Web service requester:

- ▶ The Web service description
- ▶ The Web service binding file
- ▶ The pipeline

These three objects are defined to CICS using the following attributes of the WEBSERVICE resource definition:

- ▶ WSDLFILE
- ▶ WSBIND
- ▶ PIPELINE

The WEBSERVICE definition has a fourth attribute, VALIDATION. This attribute specifies whether full validation of SOAP messages against the corresponding schema in the Web service description should be performed at runtime. Validating a SOAP message against a schema incurs considerable processing impact, and you should normally specify VALIDATION(NO) in a production environment. VALIDATION(YES) ensures that all SOAP messages that are sent and received are valid XML with respect to the XML schema. If VALIDATION(NO) is specified, sufficient validation is performed to ensure that the message contains well-formed XML.

Important: In order to be able to use the VALIDATION option, your CICS region has to be set up with Java support. Otherwise, the pipeline processing does not require Java support to be set up for the region.

Figure 2-2 on page 48 and Figure 2-3 on page 52 illustrate the purpose of the WEBSERVICE resource definition for service provider and service requester pipelines respectively.

You can create WEBSERVICE resource definitions in the following ways:

- ▶ Using the CEDA transaction
- ▶ Using the DFHCSDUP batch utility
- ▶ Using CICSplex SM Business Application Services
- ▶ Using the EXEC CICS CREATE WEBSERVICE command

When you install a PIPELINE resource, or when you issue a PERFORM PIPELINE SCAN command (using CEMT or the CICS system programming interface), CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory), and creates URIMAP and WEBSERVICE resources dynamically. For each Web service binding file in the directory, that is, for each file with the wsbind suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is more recent than the existing resources.

The CEMT INQUIRE WEBSERVICE command is used to obtain information about a WEBSERVICE resource definition. The data returned depends on the type of Web service. Table 2-2 shows the types and the data returned for each service.

Table 2-2 CEMT INQUIRE WEBSERVICE command output

Attributes	Service Provider	Service Requester to a local service	Service Requester to a remote service
PIPELINE	Yes	Yes	Yes
VALIDATIONST	Yes	Yes	Yes
STATE	Yes	Yes	Yes
CCSID	Yes	Yes	Yes
URIMAP	Yes, if dynamically installed	Empty	Empty
PROGRAM	Yes	Yes	Empty
PGMINTERFACE	Yes	Yes	No
XOPSUPPORTST	Yes	Yes	Yes
XOPDIRECTST	Yes	Yes	Yes
MAPPINGLEVEL	Yes	Yes	Yes
MINRUNLEVEL	Yes	Yes	Yes
CONTAINER	Yes, if Channel is used	Yes, if Channel is used	No
WSDLFILE	Yes	Yes	Yes
WSBIND	Yes	Yes	Yes
ENDPOINT	Empty	Empty	Yes
BINDING	Yes	Yes	Yes

2.6.4 Web service binding file

Figure 2-7 shows the role of the Web service binding (wsbind) file. The wsbind file is created by the CICS Web services assistant utilities or WebSphere Developer for System z.

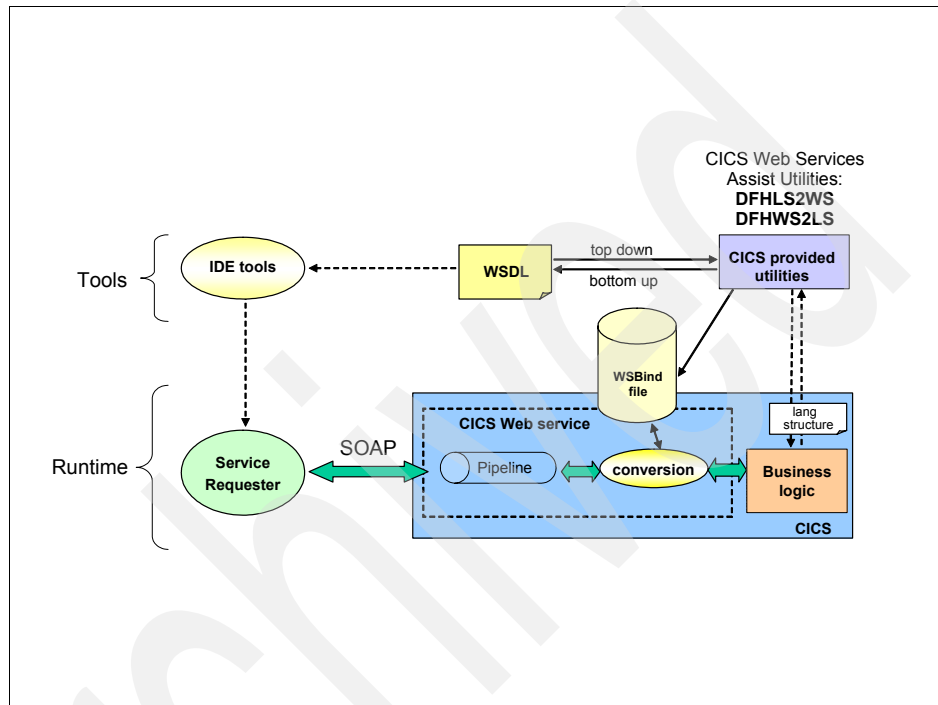


Figure 2-7 wsbind file

The wsbind file is associated with a WEBSERVICE resource and is read when the WEBSERVICE is installed. It contains all of the information necessary to interpret the body of a SOAP message at runtime. This information is used to map data from a SOAP input message to an application data structure (for example, a COMMAREA) and from an application data structure to a SOAP output message.

A sample SOAP message is shown in Example 2-3.

Example 2-3 SOAP message

Sample SOAP Message

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
...>
  <SOAP-ENV:Body>
    <TRGTAPPL0peration>
      <data_structure>
        <structure_part_1>
          <data_item_1>ABCDE</data_item_1>
          <data_item_2>12345678</data_item_2>
          <data_item_3>X</data_item_3>
        </structure_part_1>
        <structure_part_2>
          <data_item_4>Y</data_item_4>
          <data_item_5>ABCDEFG</data_item_5>
          <data_item_6>123</data_item_6>
        </structure_part_2>
      </data_structure>
    </TRGTAPPL0peration>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The CICS Web services assistant data conversion program converts the XML in Example 2-3 to the data structure shown in Example 2-4.

Example 2-4 Data structure

```
01 DATA-STRUCTURE.
02 STRUCTURE-PART-1.
03 DATA-ITEM-1    PIC X(5).
03 DATA-ITEM-2    PIC 9(8).
03 DATA-ITEM-3    PIC X.
02 STRUCTURE-PART-2.
03 DATA-ITEM-4    PIC X.
03 DATA-ITEM-5    PIC X(7).
03 DATA-ITEM-6    PIC 9(3).
```

The wsbind file is a binary file and contains three sections:

- ▶ Header

The header section contains information about the characteristics of the service, including the target program name for a service provider or the URI for a service requester, and the type of interface (COMMAREA or channel).

- ▶ Index

The index section contains an index of all the operations that are described in this Web service and a pointer to which ICMs describe the data for each operation (input and output).

- ▶ Internal COMMAREA Model (ICM)

An ICM contains the input / output message conversion details for each operation. The purpose of the ICM is to act as an intermediary format for conversion between both (language structures / WSDL) and (SOAP / COMMAREA). It encapsulates sufficient information such that a specific SOAP instance can be converted to a specific COMMAREA or vice versa at runtime.

2.7 Tools for application deployment

In this section, we consider the tools available for application deployment.

2.7.1 CICS Web services assistant

The CICS Web services assistant is a set of batch utilities that can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. It contains two utility programs:

- ▶ DFHLS2WS

Generates a Web service binding file from a language structure. This utility also generates a Web service description.

- ▶ DFHWS2LS

Generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs.

The assistant supports rapid deployment of CICS applications for use in service providers and service requesters, with a minimum of programming effort. When you use the Web services assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages; CICS maps data between the body of a SOAP message and the application program's data structure.

The assistant can create a WSDL document from a simple language structure, or a language structure from an existing WSDL document. It supports COBOL, C/C++, and PL/I. However, the assistant cannot deal with every possibility, and there are times when you must take a different approach. For example:

- ▶ You do not want to use SOAP messages.
If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs are responsible for parsing inbound messages and constructing outbound messages.
- ▶ You want to use SOAP messages, but you do not want CICS to parse them.
For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you might want to parse the SOAP body yourself.
- ▶ You have an application written in an unsupported language.
In this case you should either write a wrapper program in a supported language, or write a program to perform the mapping.
- ▶ The CICS Web services assistant does not support your application's data structure.
Although the CICS Web services assistant supports the most common data types and structures, there are some that are not supported. That is, there might not always be a one to one mapping between XML data types and the data types in your language structure. In this situation, you should first consider providing a "wrapper" program that maps your application's data to a format that the assistant can support. If this is not possible, consider using Rational® Developer. As a last resort, you might have to change your application's data structure.

If you decide not to use the CICS Web services assistant, you have to:

- ▶ Provide your own code for parsing inbound messages, and constructing outbound messages (unless you use Rational Developer)
- ▶ Provide your own pipeline configuration file
- ▶ Define and install your own URIMAP and PIPELINE resources

MAPPING-LEVEL enhancements

Some restrictions that existed on data types supported by the CICS Web services assistant when CICS TS V3.1 was shipped were removed by APARs PK15904 and PK23547. Further restrictions were also removed by PK59794 for CICS TS V3.2. The MAPPING-LEVEL parameter for both DFHLS2WS and DFHWS2LS specifies the level of mapping that the batch assistant should use when generating the Web service binding file and Web service description. The value of this parameter can be:

- 1.0 The original default mapping level of CICS TS V3.1.
- 1.1 APAR PK15904 has been applied to the CICS TS V3.1 region where the Web service binding file is deployed. At this level of mapping, there are improvements to DFHWS2LS when mapping XML character and binary data types, in particular when mapping data of variable length.
- 1.2 Both APARs PK15904 and PK23547 have been applied to the CICS TS V3.1 region where the Web service binding file is deployed.

Important: MAPPING-LEVEL 1.2 introduced the CHAR-VARYING parameter for use with DFHWS2LS. The default value for this parameter is YES. If you had previously used DFHWS2LS with MAPPING-LEVEL 1.0 or 1.1 then the generated copybooks look very different. A length field is now at the start of every character string. The MAPPING-LEVEL 1.0 or 1.1 behavior can be restored by specifying CHAR-VARYING=NO.

- 2.0 The generated Web service binding file can only be installed into a CICS TS V3.2 region.
- 2.1 APAR PK59794 has been applied to CICS TS V3.2. xsd:any and xsd:anyType elements in WSDL are now supported by DFHWS2LS. You can also specify that variably repeating data is mapped inline instead of being mapped to a container and instance count.

Important: The xsd:any and xsd:anyType support passes the associated XML elements to and from the application program within a container. This container must be populated in CHAR mode. It does not work if populated in BIT mode.

At MAPPING-LEVEL1.2, the support for data type mapping that CICS TS V3.1 provides is identical to the one provided by base CICS TS V3.2 at MAPPING-LEVEL 2.0. At this level of support, you can use additional parameters in DFHLS2WS and DFHWS2LS to control how character and binary data is transformed at runtime. You can use the CHAR-VARYING parameter to specify whether to process character fields as fixed length fields or as null terminated strings.

MINIMUM-RUNTIME-LEVEL

APAR PK23547 introduced the MINIMUM-RUNTIME-LEVEL parameter. This parameter defaults to MINIMUM. It can also have the values CURRENT or any of the valid MAPPING-LEVEL values.

By setting MINIMUM-RUNTIME-LEVEL when running the assistant, you are setting a restriction on the CICS systems that can install the resultant WSBIND file. For example specifying MINIMUM-RUNTIME-LEVEL=1.1 the CICS system must be V3.1 with APAR PK15904 or be a V3.2 system. A base level V3.1 system would not be able to install the WSBIND file.

The default value of MINIMUM calculates the runtime level required based on all the other parameters specified to run the assistant. The opposite is CURRENT which is the latest level available.

Tip: Usually you would just let this parameter default to MINIMUM. The CCSID parameter is one case where you might want to use MAPPING-LEVEL 1.0 or 1.1 but specify MINIMUM-RUNTIME-LEVEL=1.2.

Using DFHLS2WS

This program is typically used when creating a Web service to run in a CICS service provider pipeline, where the COMMAREA or CONTAINER input and output data structures (and perhaps also the program that is to be the target application) already exist. DFHLS2WS takes the input and output data structures from the service provider program, and generates the wsbind and WSDL files.

An example of the JCL required to run DFHLS2WS is shown in Example 2-5.

Example 2-5 JCL for DFHLS2WS

```
//LS2WS1 JOB (999,P0K),'CICS LS2WS TOOL',MSGCLASS=T,  
//          CLASS=A,NOTIFY=&SYSUID,TIME=1440,REGION=OM  
//*  
// JCLLIB ORDER=CICSTS32.CICS.SDFHINST  
//*  
// SET QT=''''  
//LS2WS EXEC DFHLS2WS,  
// JAVADIR='java142s/J1.4',  
// USSDIR='cicsts31',  
// TMPFILE='tmp',  
// PATHPREF=''  
//INPUT.SYSUT1 DD *  
LOGFILE=/u/cicsdev1/soap1.log  
PDSLIB=//CICSDEV1.WEBSERV.SOURCE  
REQMEM=DATA-STRUCTURE-INPUT  
RESPMEM=DATA-STRUCTURE-OUTPUT  
LANG=COBOL  
PGMNAME=TRGTAPPL  
URI=http://myserver.example.org:8080/example/wsd1/soap11/APPL01  
PGMINT=COMMAREA  
TRANSACTION=ABCD  
WSBIND=/u/cicsdev1/webservices/wsbind/app101.wsbind  
WSDL=/u/cicsdev1/webservices/wsd1/app101.wsd1  
MAPPING-LEVEL=2.0  
MINIMUM-RUNTIME-LEVEL=2.0  
CHAR-VARYING=NULL  
SOAPVER=ALL  
WSDL_2_0=/u/cicsdev1/webservices/wsd1/app101_20.wsd1  
*/
```

REQMEM and RESPMEM specify the member names containing the input and output data structures, respectively:

- ▶ PGMNAME specifies the target application.
- ▶ URI specifies the relative or (in this case) an absolute URI that a client uses to invoke this Web service.

In CICS TS V3.1, you can only specify a relative URI. CICS uses the value specified when it generates a URIMAP resource definition from the wsbind file created by DFHLS2WS. This is the path component of the URI to which the URIMAP definition applies. The complete URI that the Web service HTTP client uses is composed by concatenating the SCHEME (HTTP or HTTPS) followed by HOST and followed by the PATH, as they appear in URIMAP.

When the generated WSDL is shipped to the client, it has to be edited to specify the correct host address and port number.

In CICS TS V3.2, you can specify the full URI to the CICS Web Services Assistant, so in this case, there is no necessity to edit the WSDL file before it is shipped to the client environment.

- ▶ PGMINT specifies that the data structure is passed to the application in a COMMAREA. If using a container, specify PGMINT=CONTAINER and add CONTID=container_name as an extra parameter.
- ▶ TRANSACTION specifies the name of the alias transaction that can start the pipeline in a service provider or run a user application to compose the response. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically during the PIPELINE scan.
- ▶ WSBIND specifies where the wsbind file is to be stored.
- ▶ WSDL specifies where the WSDL file is to be stored.
- ▶ WSDL_2_0 specifies where the WSDL file that conforms to 2.0 specification is to be stored.
- ▶ SOAPVER specifies which SOAP level to use in the generated Web service description.
- ▶ MINIMUM-RUNTIME-LEVEL specifies the minimum CICS runtime environment on which the Web service bind file can be deployed.
- ▶ MAPPING-LEVEL specifies the level of mapping that DFHLS2WS should use when generating the wsbind file and Web service description.
- ▶ CHAR-VARYING specifies how character fields in the language structure should be mapped when the mapping level is 1.2 or higher.

Using DFHWS2LS

This program is typically used when creating a Web service to run in a CICS service requester pipeline, where the WSDL to access the service provider already exists. DFHWS2LS takes the WSDL as input, along with the binding to be used (that is, whether to use HTTP or WebSphere MQ at the transport layer), and generates the input and output data structures and the wsbind file to be used by the service requester.

At runtime, the service requester application stores the outbound data structure in a container, issues an EXEC CICS INVOKE WEBSERVICE request to send the Web service request to the service provider, and then reads the inbound data structure response, coming back from the service provider, from the same container.

An example of the JCL required to run DFHWS2LS is shown in Example 2-6.

Example 2-6 JCL for DFHWS2LS

```
//WS2LS1 JOB (999,P0K),'CICS WS2LS TOOL',MSGCLASS=T,  
//          CLASS=A,NOTIFY=&SYSUID,TIME=1440,REGION=OM  
//*  
// JCLLIB ORDER=CICSTS31.CICS.SDFHINST  
//*  
// SET QT=''''  
//WS2LS EXEC DFHWS2LS,  
// JAVADIR='java142s/J1.4',  
// USSDIR='cicsts31',  
// TMPFILE='tmp',  
// PATHPREF=''  
//INPUT.SYSUT1 DD *  
LOGFILE=/u/cicsdev1/soap1.log  
PDSLIB=//CICSDEV1.WEBSERV.SOURCE  
REQMEM=DATA-STRUCTURE-INPUT  
RESPMEM=DATA-STRUCTURE-OUTPUT  
LANG=COBOL  
BINDING=APPL01HTTPSoapBinding  
WSBIND=/u/cicsdev1/webservices/wsbind/app101.wsbind  
WSDL=/u/cicsdev1/webservices/wsd1/app101.wsd1  
TRANSACTION=XYZW  
MAPPING-LEVEL=2.0  
MINIMUM-RUNTIME-LEVEL=2.0  
  
*/
```

These are the main differences in the parameters between DFHLS2WS and DFHWS2LS:

- ▶ REQMEM and RESPMEM specify the member names for the data structures to be created, while these were input parameters for DFHLS2WS.
- ▶ PGMNAME and PGMINT are omitted.
- ▶ URI is omitted. The URI to invoke this Web service is in the WSDL.
- ▶ WSDL specifies where the WSDL file is located.
- ▶ TRANSACTION specifies the name of the alias transaction that can start the pipeline in a service provider, or run a user application to compose the response. The value of this parameter is used to define the TRANSACTION attribute of the URIMAP resource when it is created automatically during the PIPELINE scan.
- ▶ BINDING specifies which binding name within the WSDL is to be used. This is how either HTTP or WebSphere MQ is selected by a CICS client.

- ▶ **MINIMUM-RUNTIME-LEVEL** specifies the minimum CICS runtime environment on which the Web service bind file can be deployed.
- ▶ **MAPPING-LEVEL** specifies the level of mapping that DFHLS2WS should use when generating the wsbind file and Web service description.

SupportPac CS04: CICS TS for z/OS WSBIND File Display and Change Utility

A common task to want to do is to specify the transaction id or userid that the Web service provider task in CICS is to run under. To do this for a WSBIND file that is to be deployed in a single environment, is easily achieved by specifying those parameters when the Web services assistant is run and the WSBIND file is generated. However, if that WSBIND file is to be deployed in a CICS system with different runtime requirements you have a problem. There are several ways to handle this though:

- ▶ Regenerate the WSBIND file.

There is a danger here that it does not get created using the same parameters as the original leading to runtime incompatibility with either the target application program or the SOAP messages being sent and received.

- ▶ Manually create a URIMAP.

The URIMAP can specify exactly what you want and can override the one automatically generated when the WEBSERVICE is installed. However, it is an extra administrative overhead to create and maintain this, especially when there are hundreds of them to manage.

- ▶ Use the supportpac utility.

The utility supplied by this SupportPac can modify deployment characteristics of a WSBIND file in a batch environment before it gets installed in to CICS. This does not regenerate the WSBIND file so the original mappings, and therefore runtime compatibility remain intact.

Note: The SupportPac utility requires either CICS TS V3.1 or V3.2 to be installed as it uses the Web Services Assistants code shipped by CICS.

2.7.2 WebSphere Developer for System z

IBM WebSphere Developer for System z V7 combines the power of service-oriented architectures that use Java 2 Enterprise Edition (J2EE), CICS, IMS™, COBOL, and PL/I technologies with a Rapid Application Development (RAD) paradigm and teaming, and it brings this power to diverse enterprise application development organizations.

Today's applications are not developed by super technologists in a vacuum. Instead, they are developed by teams of people with varying levels of technology backgrounds but with the following common goals:

- ▶ A firm understanding of the business
- ▶ A requirement to integrate across many business processes
- ▶ A desire to leverage currently executing (as well as new) Web-oriented technologies
- ▶ A desire to promote usage of the Internet in meeting business requirements
- ▶ A desire to leverage development standards and expertise to move the business forward

Two overriding goals drive these organizations:

- ▶ Meeting requirements of time to market
- ▶ Meeting high quality standards

To help enterprise customers meet these goals, IBM delivered WebSphere Developer for System z. WebSphere Developer for System z supports a development process that includes:

- ▶ A standards-based Java server faces visual environment that supports rapid application development (RAD).
- ▶ A visual Web-flow construction environment built on the Struts-based Model-View-Controller (MVC) paradigm. The elements of the MVC paradigm are as follows:
 - Model and Business Logic: Invoked by or implemented in Struts action classes. Business logic can be implemented in a variety of technologies, such as Java, COBOL, and PL/I.
 - View: Implemented using HTML and Java Server Pages with special Struts tags.
 - Controller: Flow of Web applications, implemented as Struts actions or actions defined as supporting various underlying business and technical processes.

The MVC paradigm is recommended by J2EE experts, and used by architects and developers to define, reuse, and debug application organization, flow, and actions.

- ▶ Editors and interactive development environments (IDEs) for Java, COBOL, PL/I, and Enterprise Generation Language (EGL) components, including language understanding, syntax checking, and unit testing in WebSphere, CICS, and IMS transactional environments.

- ▶ Web services support, including client generation, XML editors, COBOL and PL/I adapter generation, and support for Web services deployed to WebSphere, CICS, and IMS, through SOAP for IMS.
- ▶ Support for multiple databases, including DB2.
- ▶ Integration of Web services and XML processes, including transformation of messages in WebSphere, CICS, and IMS transactional environments.
- ▶ COBOL transformers.
 - Generated from XML Schema Definition (XSD) mapping leveraging IBM Enterprise COBOL parsing and generation.
 - Deployable to CICS, IMS, and batch runtime environments.
- ▶ Support for Java and COBOL stored procedure generation and deployment.
- ▶ Remote z/OS support, including data set access, job submission, queue management, and TSO command processing.
- ▶ Cross platform interactive testing and debugging, including WebSphere, CICS, and IMS transactional environments.
- ▶ Deployment in WebSphere, CICS, and IMS transactional environments.
- ▶ Local CICS development environment.

In short, the goal of WebSphere Developer is to lower the technological requirements of entry to modern application development, and to embrace a service-oriented architecture (SOA), leveraging a reusable model-based development paradigm that helps organizations meet their business requirements today and in the future. Organizations that use WebSphere Developer should be able to:

- ▶ Simplify the process of developing J2EE and integrated mixed workload applications that include CICS and IMS processing.
- ▶ Include and support the collaboration of multiple roles in the process.
- ▶ Promote reuse and transformation of existing applications to an e-business model.
- ▶ Lessen training costs and leverage common developer skills across mixed workload environments, thus increasing overall organizational knowledge and flexibility.

2.7.3 Comparing Web services assistant and WebSphere Developer for System z

Table 2-3 compares the XML parsing and generation capabilities of the CICS Web services assistant at mapping level 1.2 or 2.0 with those of the WebSphere Developer for System z V7.

Table 2-3 CICS Web services assistant and WebSphere Developer parsing/generation

Description	Web services assistant at mapping levels 1.2 or 2.0	WebSphere Developer for System z V7
Implementation	ICM - metadata	Generated Cobol parser
Language support	<ul style="list-style-type: none"> ▶ COBOL ▶ PL/I ▶ C or C++ 	COBOL
COBOL data types	<ul style="list-style-type: none"> ▶ DISPLAY ▶ Alphanumeric edited ▶ Numeric edited ▶ COMP ▶ COMP-1 ▶ COMP-2 ▶ COMP-3 ▶ COMP-4 ▶ COMP-5 ▶ BINARY 	<ul style="list-style-type: none"> ▶ DISPLAY ▶ Alphanumeric edited ▶ Numeric edited ▶ COMP ▶ COMP-1 ▶ COMP-2 ▶ COMP-3 ▶ COMP-4 ▶ COMP-5 ▶ BINARY
COBOL structure	Elementary items	<ul style="list-style-type: none"> ▶ Elementary items ▶ 88 level items ▶ Group items ▶ Redefines ▶ OCCURS ▶ OCCURS DEPENDING ON
C and C++ data types	<ul style="list-style-type: none"> ▶ char ▶ unsigned char ▶ signed char ▶ short ▶ unsigned short ▶ int ▶ unsigned int ▶ long ▶ unsigned long ▶ long long ▶ unsigned long long ▶ bool (C++ only) ▶ float ▶ double 	N/A

Description	Web services assistant at mapping levels 1.2 or 2.0	WebSphere Developer for System z V7
PL/I data types	<ul style="list-style-type: none"> ▶ FIXED BINARY ▶ UNSIGNED FIXED BINARY ▶ FIXED DECIMAL ▶ BIT ▶ CHARACTER ▶ GRAPHIC ▶ WIDECHAR ▶ ORDINAL ▶ BINARY FLOAT ▶ DECIMAL FLOAT 	N/A

Archived

Archived

Web service configuration

In this part of the book, we provide detailed instructions on how we configured our test CICS environment to support Web services using both HTTP and WebSphere MQ as transport mechanisms. We also describe how we deployed service requester and service provider applications in WebSphere Application Server, and how we used Web services to connect between WebSphere Application Server and CICS.

In chapter 5, we introduce the service integration bus and we outline the configuration steps for connecting to a CICS Web service via the bus.

Archived



Web services using HTTP

In this chapter, we describe how we configured our test CICS environment to support Web services using HTTP as the transport mechanism.

3.1 Preparation

After outlining our test configuration (Figure 3-1), we explain how we configured CICS as a service provider. In 3.2, “Configuring CICS as a service provider” on page 88, we show details of how we set up the environment, including:

- ▶ Configuring code page support
- ▶ Configuring the HFS file system
- ▶ Enabling the service provider application in CICS
- ▶ Deploying the service requester client in WebSphere Application Server for Windows®
- ▶ Managing the WebSphere Application Server connection pool for Web services outbound connections

In 3.3, “Configuring CICS as a service requester” on page 106, we show how we configured the same CICS region to act as a service requester, including:

- ▶ Enabling the service requester application in CICS
- ▶ Deploying the service provider application in WebSphere Application Server for z/OS

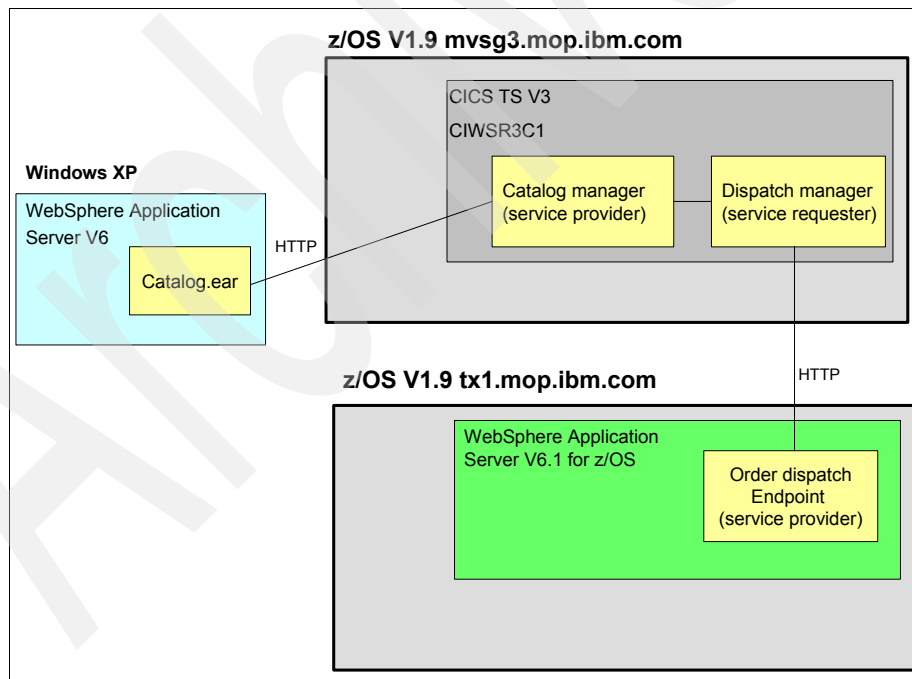


Figure 3-1 Software components: Web services using HTTP transport

We do not provide information about how to install the software products, and we assume that the reader has a working knowledge of both CICS and WebSphere Application Server.

3.1.1 Software checklist

For the configuration shown in Figure 3-1, we used the levels of software shown in Table 3-1.

Table 3-1 Software used in the HTTP scenarios

Windows	z/OS
Windows XP SP4	z/OS V1.9
IBM WebSphere Application Server - ND V6.1.0.170	WebSphere Application Server for z/OS V6.0.1
	CICS Transaction Server V3.2
Internet Explorer® V6.0	
Our J2EE application <ul style="list-style-type: none"> ▶ Catalog.ear ▶ Catalog manager service requester application 	CICS-supplied catalog Manager application Our user-supplied CICS programs <ul style="list-style-type: none"> ▶ SNIFFER (message handler program) ▶ CIWSMSGH (message handler program) Our J2EE application <ul style="list-style-type: none"> ▶ dispatchOrder.ear ▶ Catalog application service provider

3.1.2 Definition checklist

The z/OS definitions we used to configure the scenario are listed in Table 3-2.

Table 3-2 Definition checklist

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	tx1.mop.ibm.com
IP address	9.100.193.167	9.100.193.122
TCP/IP port	13301	13880
Job name	CIWSR3C1	CITGRS1S

Value	CICS TS	WebSphere Application Server
APPLID	A6POR3C1	
TCPIPSERVICE	R3C1	
Provider PIPELINE	EXPIPE01	
Requester PIPELINE	EXPIPE02	

3.1.3 The sample application

For our tests we used the sample program described in 2.5, “Catalog manager example application” on page 53. We do not document how to install the sample application itself, because this is explained in detail in *CICS Web Services Guide V3.1*, SC34-6458.

3.2 Configuring CICS as a service provider

In this section we discuss how we configured CICS as a service provider. The configuration we used is shown in Figure 3-2.

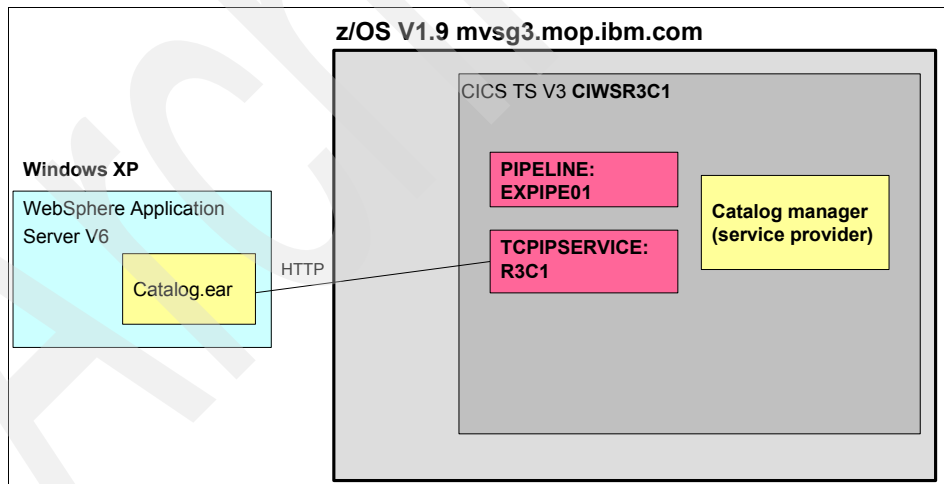


Figure 3-2 CICS as a service provider

3.2.1 Configuring code page support

We configured our z/OS system to support conversions between the two coded character sets used by the example application; these are shown in Example 3-1.

Example 3-1 CCSID description

```
037    EBCDIC Group 1: USA, Canada (z/OS), Netherlands, Portugal,  
        Brazil, Australia, New Zealand  
1208 UTF-8 Level 3
```

To do this, we added the statements shown in Example 3-2 to the conversion image for our z/OS system.

Example 3-2 Required conversions

```
CONVERSION 037,1208;  
CONVERSION 1208,037;
```

We used the SET UNI=31 z/OS command to activate the updated conversion image, where **31** is the suffix of the CUNUNIXx member of SYS1.PARMLIB.

For more information about code page support, see *CICS Installation Guide V3.1*, GC34-6426.

Tip: With z/OS V1R7, the Unicode Services environment can be dynamically updated when a conversion service is requested. If the appropriate table necessary for the service is not already loaded into storage, Unicode Services loads the table without requiring an IPL or disrupting the caller's request. For more information, see *z/OS Support for Unicode: Unicode Services*, SA22-7649-06.

3.2.2 Configuring CICS

To enable CICS to receive Web service requests using HTTP, we performed the following tasks:

- ▶ Updating CICS system initialization table (SIT) parameters
- ▶ Creating the HFS directories
- ▶ Configuring the TCPIP SERVICE resource definition
- ▶ Customizing the pipeline configuration file
- ▶ Writing a message handler program that changes the transaction ID
- ▶ Configuring the PIPELINE resource definition
- ▶ Installing the TCPIP SERVICE and PIPELINE definitions

Updating SIT parameters

Since we decided to use HTTP as the transport for the service request flows, we added the following SIT parameter:

```
TCPIP=YES
```

We then restarted the CICS region to put the parameter into effect.

Creating the HFS directories

Next we created the HFS directories (Example 3-3) used in the PIPELINE definition.

Example 3-3 HFS directories used in the PIPELINE definition

```
/CIWS/R3C1/config  
/CIWS/R3C1/shelf  
/CIWS/R3C1/wsbinding/provider  
/CIWS/R3C1/wsbinding/requester
```

The CICS region user ID must have read permission to the /config directory, and update permission to the /shelf directory.

Configuring the TCPIP SERVICE definition

Next we logged onto CICS and created the TCPIP SERVICE resource definition in CICS using the command:

```
CEDA DEFINE TCPIP SERVICE(R3C1) GROUP(R3C1)
```

We defined the R3C1 TCPIP SERVICE as shown in Figure 3-3.

```

OVERTYPE TO MODIFY                                CICS RELEASE = 0650
CEDA DEFINE TCPIPService( R3C1      )
TCPIPService   : R3C1
GRoup          : R3C1
DEscription    ==> TCPIPService DEFINITION FOR CATALOG APPLICATION
Urm            ==> NONE
PORTnumber     ==> 13301                1-65535
STatus         ==> Open                 Open | Closed
PROtocol       ==> Http                 Iiop | Http | Eci | User
TRansaction    ==> CWXN
Backlog        ==> 00001                0-32767
TSqprefix      ==>
Ippaddress     ==>
SOcketclose    ==> No                   No | 0-240000 (HHMMSS)
Maxdatalen     ==> 000032                3-524288
SECURITY
SS1            ==> No                   Yes | No | Clientauth
CErtificate    ==>
(Mixed Case)

SYSID=R3C1 APPLID=A6POR3C1

```

Figure 3-3 CEDA DEFINE TCPIPService

We set the PORTNUMBER to 13301, the PROTOCOL to HTTP, and the URM to NONE. We allowed the other attributes to default, and we installed the R3C1 group.

The default setting for the SOCKETCLOSE attribute is NO. Therefore, when a connection is made between a Web service client and CICS, by default CICS keeps the connection open until the Web service client closes the connection. You could set a value (in seconds) for the SOCKETCLOSE attribute if you want to close a persistent connection after the timeout period is reached.

Recommendation: Do not set the SOCKETCLOSE attribute to 0 because this closes the connection after each request.

We used the default setting for SOCKETCLOSE and we configured WebSphere Application Server to timeout idle persisting connections (see “Managing the WebSphere Application Server connection pool” on page 101).

Customizing the pipeline configuration file

The default pipeline alias transaction ID used for inbound HTTP Web service requests is CPH. We wanted to assign different transaction IDs to different service requests. To do that, we wrote a message handler program CIWSMSGH that replaced the transaction ID in the DFHWS-TRANID container with a transaction ID based on the service request (which can be retrieved from the DFHWS-WEBSERVICE container).

To activate the message handler program, we had to make changes to the PIPELINE configuration file. We copied the file, basicsoap12provider.xml to the /CIWS/R3C1/config directory shown in Example 3-3 on page 90. The change we made is shown here in Example 3-4.

Example 3-4 Service provider configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
xmlns="http://www.ibm.com/software/htp/cics/pipeline"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
provider.xsd ">
  <transport>
    <default_transport_handler_list>
      <handler>
        <program>CIWSMSGH</program>
        <handler_parameter_list/>
      </handler>
    </default_transport_handler_list>
  </transport>
  <service>
    <terminal_handler>
      <cics_soap_1.2_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Note: The <default_transport_handler_list> specifies the message handlers that are invoked by default when any transport is in use.

Writing the message handler program

In this section we show how we used a message handler program to change the default transaction ID (CPIH) to a transaction ID based on the Web service request in the DFHWS-WEBSERVICE container.

Table 3-3 shows the relationship that we established between the transaction ID and the Web service request.

Table 3-3 Transaction ID to Web service name relationship

Transaction ID	Web service request
INQS	inquireSingle
INQC	inquireCatalog
ORDR	placeOrder

There are many reasons why you might want to change the transaction ID based on the service request, for example:

- Security** You might want to use transaction security to control access to specific services.
- Priority** You might want to assign different performance goals to specific services.
- Accounting** You might have to charge users based on access to different services.

Before we activated the message handler program, we had to create the new TRANSACTION definitions with the same characteristics as the CICS-supplied definition for CPIH. We used the CEDA COPY commands shown in Example 3-5 and then installed the definitions.

Example 3-5 CICS definitions - TRANSACTION

```
CEDA COPY TRANSACTION(CPIH) GROUP(DFHPIPE) TO(R3C1) AS(INQS)
CEDA COPY TRANSACTION(CPIH) GROUP(DFHPIPE) TO(R3C1) AS(INQC)
CEDA COPY TRANSACTION(CPIH) GROUP(DFHPIPE) TO(R3C1) AS(ORDR)
```

The program logic we used for the message handler program is shown in Example 3-6 through Example 3-9.

Note: A message handler can be written in any of the languages CICS supports. The CICS commands in the DPL subset can be used.

Example 3-6 shows the flow of control of the message handler program. The program only executes for Web service requests.

Example 3-6 Message handler program - Flow of control

```
007700     IF WS-DFHFUNCTION equal 'RECEIVE-REQUEST'
007800         PERFORM VALIDATE-REQUEST THRU END-VAL-REQUEST
007900         PERFORM CHANGE-TRANID     THRU END-CHANGE-TRANID
008000         EXEC CICS
008100             DELETE CONTAINER('DFHRESPONSE')
008200         END-EXEC
008300     END-IF
008400     EXEC CICS RETURN END-EXEC.
```

Tip: When the message handler processes a request, it must delete the DFHRESPONSE container if a transition to the response phase of the pipeline does not take place.

Example 3-7 shows the code to get the Web service request from the DFHWS-WEBSERVICE container.

Example 3-7 Message handler program - Get the DFHWS-WEBSERVICE container

```
010300     EXEC CICS
010400         GET CONTAINER('DFHWS-WEBSERVICE')
010500         SET(ADDRESS OF CONTAINER-DATA)
010600         FLENGTH(CONTAINER-LEN)
010700     END-EXEC.
```

Example 3-8 shows the code that determines the new transaction ID, replacing the default transaction ID in the DFHWS-TRANID container.

Example 3-8 Message handler program - Determine new transaction ID

```
011400*----- CHANGE DEFAULT TRANID CPIH/CPIL -----
011500 CHANGE-TRANID.
011600     EXEC CICS GET CONTAINER('DFHWS-TRANID')
011700         SET(ADDRESS OF CONTAINER-DATA)
011800         FLENGTH(CONTAINER-LEN)
011900     END-EXEC.
012000     IF WS-WEBSERVICES = 'inquireSingle'
012100         MOVE 'INQS' TO CA-TRANID
012200         PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
012300     END-IF
012400     IF WS-WEBSERVICES = 'inquireCatalog'
012500         MOVE 'INQC' TO CA-TRANID
012600         PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
012700     END-IF
012800     IF WS-WEBSERVICES = 'placeOrder'
012900         MOVE 'ORDR' TO CA-TRANID
013000         PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
013100     END-IF.
013200 END-CHANGE-TRANID. EXIT.
```

Example 3-9 shows how the program changes the transaction ID in the DFHWS-TRANID container, and performs an EXEC CICS PUT CONTAINER.

Example 3-9 Message handler program - Change transaction ID

```
013600 CHANGE-CONTAINER.
013700     MOVE CA-TRANID TO CONTAINER-DATA(1:4)
013800     EXEC CICS PUT CONTAINER('DFHWS-TRANID')
013900         FROM(CONTAINER-DATA)
014000         FLENGTH(CONTAINER-LEN)
014100     END-EXEC.
015000 END-CHANGE-CONTAINER. EXIT.
```

Configuring the PIPELINE definition

We then defined the pipeline for the CICS service provider using the following CICS command:

```
CEDA DEFINE PIPELINE(EXPIPE01) GROUP(R3C1)
```

We defined the EXPIPE01 pipeline as shown in Figure 3-4.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0650
CEDA DEFine Pipeline(EXPIPE01 )
Pipeline      : EXPIPE01
Group         : R3C1
Description   ==>
STatus        ==> Enabled          Enabled | Disabled
Configfile    ==> /CIWS/R3C1/config/ITSO_7206_basicsoap12provider.xml
(Mixed Case) ==>
              ==>
              ==>
              ==>
SHeLF         ==> /CIWS/R3C1/shelf
(Mixed Case) ==>
              ==>
              ==>
              ==>
WsdIr         : /CIWS/R3C1/wsbind/provider/
(Mixed Case) :
              :
```

Figure 3-4 CEDA DEFINE PIPELINE

Tip: The colons in front of WsdIr on the CEDA screen in Figure 3-4 mean that you are not able to enter input on the lines. You must press F8 to be able to enter the path for the directory.

- ▶ We set CONFIGFILE to the name of our pipeline configuration file:
/CIWS/R3C1/config/ITSO_7206_basicsoap12provider.xml

Note: In a subsequent section we show how we modified the basicsoap12provider.xml file, which is why we did not use the pipeline configuration file provided in the /usr/lpp/cicsts/cicsts31/samples/pipelines directory.

- ▶ We set SHELF to the name of the shelf directory:
/CIWS/R3C1/shelf
- ▶ We copied the following wsbind files to directory
/CIWS/R3C1/wsbind/provider from the CICS supplied directory
/usr/lpp/cicsts/cicsts31/samples/webservices/wsbind/provider/:
 - inquireSingle.wsbind
 - inquireCatalog.wsbind
 - placeOrder.wsbind

Note: /usr/lpp/cicsts/cicsts31 is our CICS HFS install root.

- ▶ We set WSDIR to the Web service binding directory that contains the wsbind files for the sample application:
/CIWS/R3C1/wsbind/provider/

Installing the PIPELINE definition

We then used CEDA to install the PIPELINE definition. When the PIPELINE is installed CICS scans the wsdir directory, and dynamically creates WEBSERVICE and URIMAP definitions for the wsbind files found.

Figure 3-5 shows a CEMT INQUIRE PIPELINE for EXPIPE01.

```
INQUIRE PIPELINE
RESULT - OVERTYPE TO MODIFY
  Pipeline(EXPIPE01)
  Enablestatus( Enabled )
  Configfile(/CIWS/R3C1/config/ITS0_7206_basicoap12provider.xml)
  Shelf(/CIWS/R3C1/shelf/)
  Wsdir(/CIWS/R3C1/wsbind/provider/)

SYSID=R3C1 APPLID=A6P0R3C1
```

Figure 3-5 CEMT INQUIRE PIPELINE - EXPIPE01

WEBSERVICE resource definitions

In our configuration the WEBSERVICE resource definitions are dynamically installed when the PIPELINE is installed. Optionally, we could define and install the Web services using the CEDA DEFINE WEBSERVICE command; however, this is not normally necessary when using the CICS Web services assistant.

Note: The name for an explicitly defined WEBSERVICE is limited to 8 characters in length, whereas the automatically installed Web service names can be up to 32 characters in length.

When a Web service is dynamically installed, the name of the service is taken from the wsbind file. Figure 3-6 shows the Catalog manager application Web services dispatchOrder, inquireCatalog, inquireSingle and placeOrder.

```
INQUIRE WEBSERVICE
STATUS: RESULTS - OVERTYPE TO MODIFY
Webs(inquireCatalog ) Pip(EXPIPE01)
    Ins Uri($606021 ) Pro(DFH0XCMN) Com Dat(20050408)
Webs(inquireSingle ) Pip(EXPIPE01)
    Ins Uri($606023 ) Pro(DFH0XCMN) Com Dat(20050408)
Webs(placeOrder ) Pip(EXPIPE01)
    Ins Uri($606025 ) Pro(DFH0XCMN) Com Dat(20050408)
SYSID=R3C1 APPLID=A6POR3C1
```

Figure 3-6 CEMT INQUIRE WEBSERVICE

URIMAP resource definition

In our configuration the URIMAP resource definitions are dynamically installed when the PIPELINE is installed. Optionally, we could define and install them manually using the CEDA DEFINE URIMAP command; however, this is not normally necessary when using the CICS Web services assistant.

Using the URIMAP to change the default transaction ID

As an alternative to using the message handler program to change the transaction IDs the services run under, we could have used the URIMAP resource definition. That would, however, mean that we would have to define a URIMAP for each deployed Web service.

A sample resource definition that could have been used is shown in Example 3-10.

Example 3-10 CEDA URIMAP definitions

```
CEDA DEFINE URIMAP(INQS) GROUP(R3C1) HOST(*)
PATH(/exampleApp/inquireSingle)
    PIPELINE(EXPIPE01) TRANSACTION(INQS) USAGE(PIPELINE)
    WEBSERVICE(inquireSingle)
CEDA DEFINE URIMAP(INQC) GROUP(R3C1) HOST(*)
PATH(/exampleApp/inquireCatalog)
```

```

PIPELINE(EXPIPE01) TRANSACTION(INQC) USAGE(PIPELINE)
WEBSERVICE(inquireCatalog)
CEDA DEFINE URIMAP(ORDR) GROUP(R3C1) HOST(*)
PATH(/exampleApp/placeOrder)
PIPELINE(EXPIPE01) TRANSACTION(ORDR) USAGE(PIPELINE)
WEBSERVICE(placeOrder)

```

Figure 3-7 shows a URIMAP resource definition dynamically installed when the PIPELINE is installed.

```

INQUIRE URIMAP
RESULT - OVERTYPE TO MODIFY
  Urimap($606021)
  Usage(Pipe)
  Enablestatus( Enabled )
  Analyzerstat(Noanalyzer)
  Scheme(Http)
  Redirecttype( None )
  Tcpipservice()
  Host(*)
  Path(/exampleApp/inquireCatalog)
  Transaction(CPIH)
  Converter()
  Program()
  Pipeline(EXPIPE01)
  Webservice(inquireCatalog)
  Userid()
  Certificate()
  Ciphers()
  Templatename()

```

SYSID=R3C1 APPLID=A6P0R3C1

Figure 3-7 CEMT INQUIRE URIMAP

3.2.3 Configuring WebSphere Application Server on Windows

In this section we discuss how we deployed the Web service client on WebSphere Application Server for Windows. We describe how we used the WebSphere administrative console to install the Web service client.

Installing the service requester

CICS TS V3.1 provides a sample Web service client, ExampleAppClient.ear. This application archive is built at the J2EE 1.3 level. We planned to use the client in a J2EE 1.4 environment (WebSphere Application Server V6), therefore we migrated the client. We called the new application archive file Catalog.ear.

Note: The CICS-supplied ExampleAppClient.ear file is located in the /usr/lpp/cicsts/cicsts31/samples/webservices/client directory.

Deploying the Catalog.ear file on WebSphere Application Server

Next we deployed the Catalog.ear file on WebSphere Application Server for Windows. To log on to the WebSphere administrative console, we opened a Web browser window and entered the following URL:

http://cam21-pc11:9060/admin

We entered a user ID and were presented with the window shown in Figure 3-8. We clicked **Local file system** and then clicked **Browse** to locate the EAR file:

F:\Web Services Sysprog\LAN book\addmat\src\ears\Catalog.ear

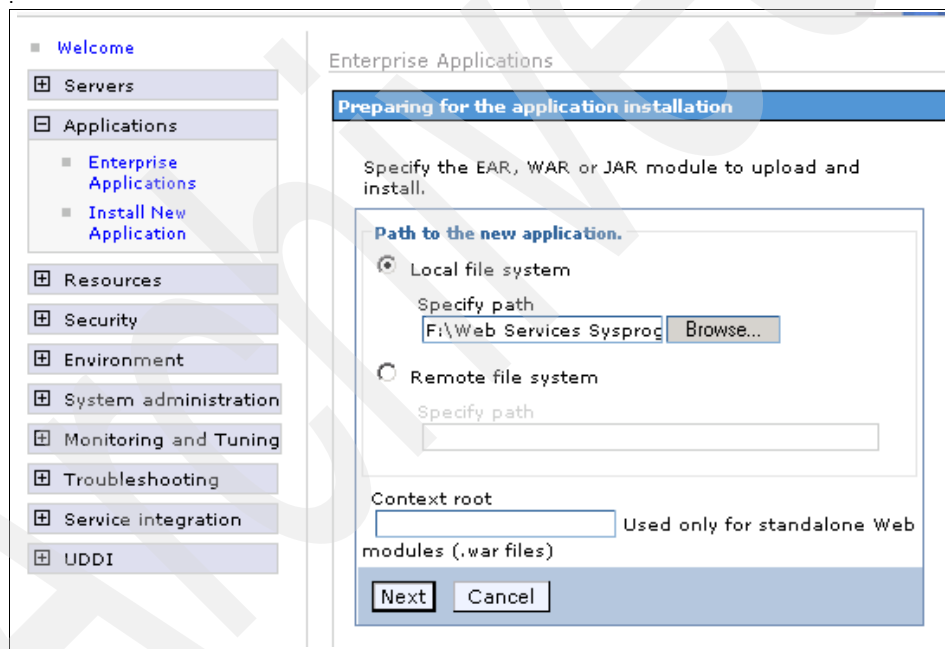


Figure 3-8 WebSphere administrative console - Install new application

We clicked **Next** → **Next** → **Next** → **Next** → **Finish** and then saved the configuration.

Next we clicked **Enterprise Applications**, selected the Catalog application, and clicked **Start** to start the application.

Managing the WebSphere Application Server connection pool

Since our service requester runs in WebSphere Application Server, the application can take advantage of the connection pooling for Web services HTTP outbound connections.

The HTTP transport properties are set using the JVM™ custom property panel in the WebSphere administrative console. The following properties apply to our scenario:

▶ **com.ibm.websphere.webservices.http.connectionTimeout**

This property specifies the interval, in seconds, after which a connection request times out and the `WebServicesFault("Connection timed out")` error occurs. The wait time is required when the maximum number of connections in the connection pool is reached. For example, if the property is set to 300 and the maximum number of connections is reached, the connector waits for 300 seconds until a connection is available. After 300 seconds, the `WebServicesFault("Connection timed out")` error occurs if a connection is not available. If the property is set to 0 (zero), the connector waits until a connection is available.

We allowed this property setting to default to 300 seconds.

▶ **com.ibm.websphere.webservices.http.maxConnection**

This property specifies the maximum number of connections that are created in the HTTP outbound connector connection pool. If the property is set to 0 (zero), the `com.ibm.websphere.webservices.http.connectionTimeout` property is ignored. The connector attempts to create as many connections as allowed by the system.

We allowed this property setting to default to 50.

▶ **com.ibm.websphere.webservices.http.connectionPoolCleanup**

This property specifies the interval, in seconds, between runs of the connection pool maintenance thread. When the pool maintenance thread runs, the connector discards any connections remaining idle for longer than the time set in `com.ibm.websphere.webservices.http.connectionIdleTimeout` property.

We allowed this property setting to default to 180 seconds.

▶ **com.ibm.websphere.webservices.http.connectionIdleTimeout**

This property specifies the interval, in seconds, after which an idle connection is discarded.

We changed this property setting from the default (5 seconds) to 60 seconds because we wanted the connections to persist for a longer period.

We used the WebSphere administrative console to change the connection idle timeout from the default 5 seconds to 60 seconds:

We clicked **Servers** → **Application servers** → **server1** → **Java and Process management** → **Environment Entries** → **New**, and on the presented window, we entered the value shown in Figure 3-9.

We restarted the application server to activate the change.

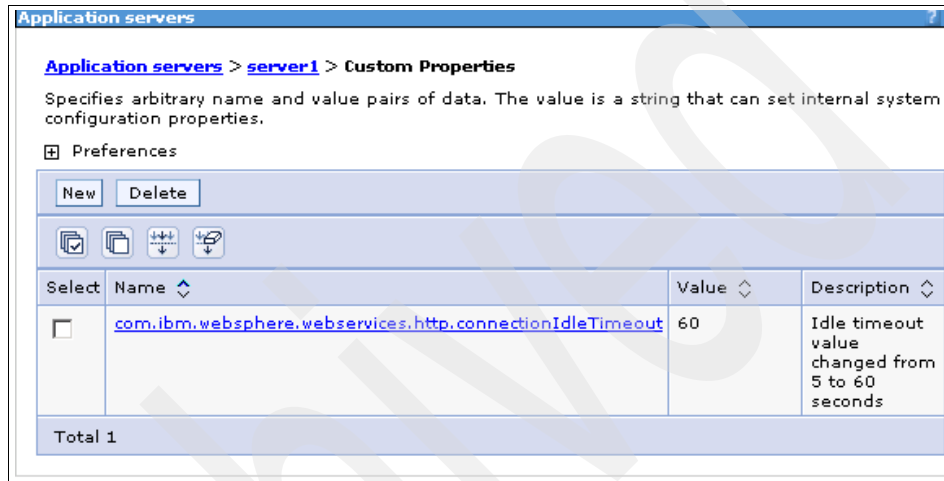


Figure 3-9 WebSphere admin console - Setting connection idle timeout

Tip: For more information about the WebSphere Application Server connection pooling properties see “Additional HTTP transport properties for Web services applications” in the WebSphere Application Server information center.

3.2.4 Testing the configuration

In this section we discuss how we tested the configuration by invoking the Web client application running on WebSphere Application Server for Windows.

Running the Web client application

We started a browser session and entered the URL:

http://cam21-pc11:9080/CatalogWeb/Welcome.jsp

The window shown in Figure 3-10 was displayed.

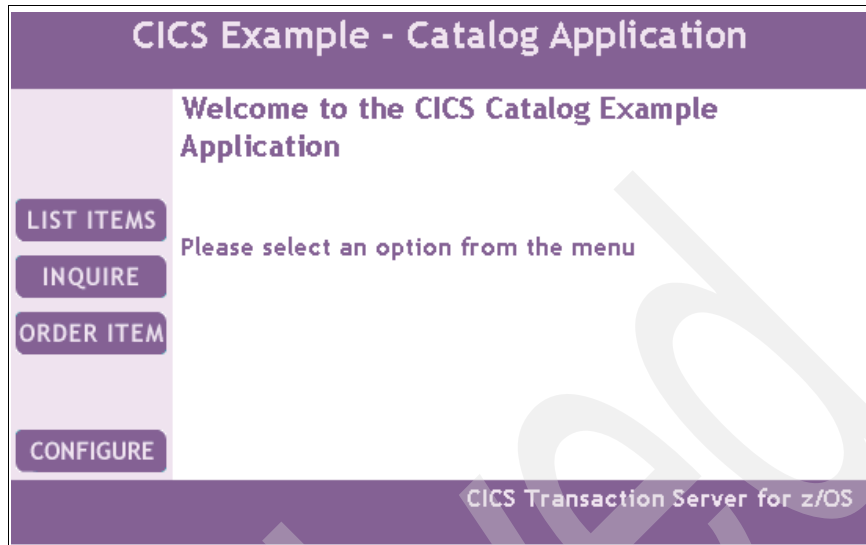


Figure 3-10 CICS - Catalog application

We clicked **CONFIGURE**, and the window in Figure 3-11 was presented. We entered the following addresses:

- ▶ Inquire catalog:
<http://mvsg3.mop.ibm.com:13301/exampleApp/inquireCatalog>
- ▶ Inquire item:
<http://mvsg3.mop.ibm.com:13301/exampleApp/inquireSingle>
- ▶ Place order:
<http://mvsg3.mop.ibm.com:13301/exampleApp/placeOrder>

We then clicked **SUBMIT**.

Configure Application

LIST ITEMS

INQUIRE

ORDER ITEM

Inquire Catalog Service Endpoint

Current `http://mvsg3.mop.ibm.com:13301/exampleApp/inquireCatalog`

New `http://mvsg3.mop.ibm.com:13301/exampleApp/inquireCatalog`

Inquire Item Service Endpoint

Current `http://mvsg3.mop.ibm.com:13301/exampleApp/inquireSingle`

New `http://mvsg3.mop.ibm.com:13301/exampleApp/inquireSingle`

Place Order Service Endpoint

Current `http://mvsg3.mop.ibm.com:13301/exampleApp/placeOrder`

New `http://mvsg3.mop.ibm.com:13301/exampleApp/placeOrder`

SUBMIT

Figure 3-11 CICS - Catalog application configuration

Next we started three Web browser sessions and entered the URL for each browser:

`http://cam21-pc11:9080/CatalogWeb/Welcome.jsp`

The Catalog application welcome page (Figure 3-10 on page 103) was presented. We then invoked a different service in each of the browsers:

- ▶ LIST ITEMS in browser one
- ▶ INQUIRE in browser two
- ▶ ORDER ITEM in browser three

From a CICS 3270 screen, we used the CICS Execution Diagnostic Facility (EDF) to intercept each of the INQC, INQS and ORDR transactions. We then used the CEMT INQUIRE TASK command to view the in-flight transactions (Figure 3-12).


```

INQUIRE TASK
STATUS: RESULTS - OVERTYPE TO MODIFY
Tas(0000052) Tra(CPIH)          Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE06893FAF5D9305) Hty(RZCBNOTI)
Tas(0000053) Tra(INQC)         Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE06893FDD7796AE) Hty(EDF      )
Tas(0000057) Tra(CPIH)          Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE06895367D28546) Hty(RZCBNOTI)
Tas(0000058) Tra(INQS)         Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE06895368907606) Hty(EDF      )
Tas(0000062) Tra(CPIH)          Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE068962FFC04308) Hty(RZCBNOTI)
Tas(0000063) Tra(ORDR)         Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE06896300499061) Hty(EDF      )

```

SYSID=R3C1 APPLID=A6P0R3C1

Figure 3-12 CEMT INQUIRE TASK

Figure 3-12 shows one instance of each of the INQC, INQS, and ORDR transactions. For each transaction there is an associated pipeline alias transaction CPIH. We noted that these transactions are currently all running under the CICS default user ID CICSUSER.

Example 3-11 shows the output from our message handler program CIWSMSGH for the three service requests. Both the DFHWS-WEBSERVICE and DFHWS-TRANID containers are logged.

Example 3-11 Sample output from message handler program - CIWSMSGH

```

CIWSMSGH: >=====
CIWSMSGH: Container Name: : DFHWS-WEBSERVICE
CIWSMSGH: Container content: inquireCatalog
CIWSMSGH: -----
CIWSMSGH: Container Name: : DFHWS-TRANID
CIWSMSGH: Container content: INQC
CIWSMSGH: >=====
CIWSMSGH: Container Name: : DFHWS-WEBSERVICE
CIWSMSGH: Container content: inquireSingle
CIWSMSGH: -----
CIWSMSGH: Container Name: : DFHWS-TRANID
CIWSMSGH: Container content: INQS
CIWSMSGH: >=====

```

```
CIWMSGH: Container Name: : DFHWS-WEBSERVICE
CIWMSGH: Container content: placeOrder
CIWMSGH: -----
CIWMSGH: Container Name: : DFHWS-TRANID
CIWMSGH: Container content: ORDR
```

3.3 Configuring CICS as a service requester

In this section we discuss how we configured CICS to support outbound Web service requests. The configuration we used is shown in Figure 3-13.

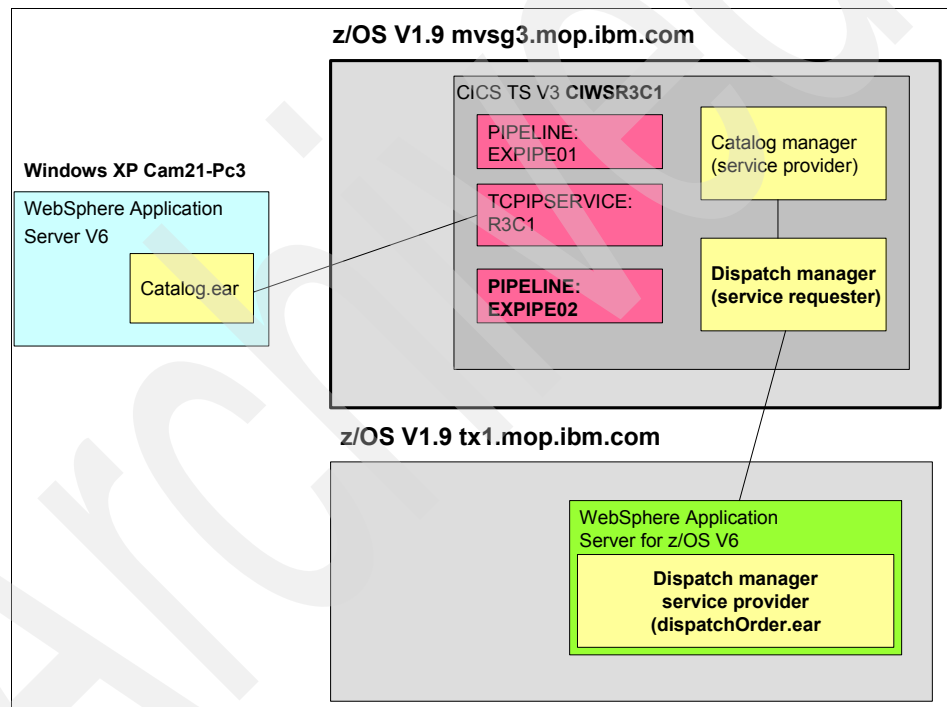


Figure 3-13 CICS as service requester

Figure 3-13 shows the TCPIP SERVICE R3C1 used for inbound HTTP Web service requests. Note that a TCPIP SERVICE is not required for outbound HTTP Web service requests from CICS.

3.3.1 Configuring CICS

To enable CICS to generate Web service requests using HTTP, we performed the following tasks:

- ▶ Configuring the PIPELINE definition
- ▶ Configuring the requester TRANSACTION definition
- ▶ Configuring the sample application

Configuring the PIPELINE definition

We defined the PIPELINE for the CICS service requester using the following CICS command:

```
CEDA DEFINE PIPELINE(EXPIPE02) GROUP(R3C1)
```

We defined the EXPIPE02 pipeline as shown in Figure 3-14.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0650
CEDA DEFine PIpeline( EXPIPE02 )
PIpeline      : EXPIPE02
Group        : R3C1
Description   ==> PIPELINE DEFINITION FOR DISPATCH ORDER REQUESTER
STatus       ==> Enabled           Enabled | Disabled
Configfile   ==> /CIWS/R3C1/config/ITS0_7206_basicsoap11requester.xml
(Mixed Case) ==>
              ==>
              ==>
              ==>
SHeLF        ==> /CIWS/R3C1/shelf
(Mixed Case) ==>
              ==>
              ==>
              ==>
WsdIr        : /CIWS/R3C1/wsbind/requester/
(Mixed Case) :
              :
```

SYSID=R3C1 APPLID=A6POR3C1

Figure 3-14 CEDA DEFINE PIPELINE command

- ▶ We set the CONFIGFILE attribute to:
/CIWS/R3C1/config/ITS0_7206_basicsoap11requester.xml
- ▶ We set the SHELF attribute to:
/CIWS/R3C1/shelf

- ▶ We copied the wsbind file dispatchOrder.wsbind to directory /CIWS/R3C1/wsbind/requester from the CICS-supplied directory: /usr/lpp/cicsts/cicsts31/samples/webservices/wsbind/requester/
- ▶ We set WSDIR to the Web service binding directory that contains the wsbind files for the sample application: /CIWS/R3C1/wsbind/requester/

Note: In 3.2, “Configuring CICS as a service provider” on page 88, we used the basicsoap12provider.xml configuration file, which supports both SOAP 1.1 and SOAP 1.2 inbound service requests. CICS only supplies a basicsoap11requester.xml configuration file for SOAP 1.1 outbound requests.

Figure 3-15 shows a CEMT INQUIRE PIPELINE for EXPIPE02.

```
INQUIRE PIPELINE
RESULT - OVERTYPE TO MODIFY
  Pipeline(EXPIPE02)
  Enablestatus( Enabled )
  Configfile(/CIWS/R3C1/config/ITS0_7206_basicsoap11requester.xml)
  Shelf(/CIWS/R3C1/shelf/)
  Wsdir(/CIWS/R3C1/wsbind/requester/)

SYSID=R3C1 APPLID=A6POR3C1
```

Figure 3-15 CEMT INQUIRE PIPELINE - EXPIPE02

Configuring the requester transaction

The duration a Web service requester task waits for a response is controlled by the DTIMOUT attribute on the TRANSACTION definition. The CICS default is NO, meaning that the request waits indefinitely. We used the CEDA ALTER command to change the DTIMOUT value for the ORDR transaction to 30 seconds:

```
CEDA ALTER TRANSACTION(ORDR) GROUP(R3C1) DTIMOUT(30)
```

Timeout considerations

When a CICS application is the service *provider*, normal resource timeout mechanisms such as RTIMEOUT (read timeout) apply. If, however, the requester decides to time out before CICS is ready to send the response, the provider transaction abends and CICS issues the messages shown in Example 3-12.

Example 3-12 CICS service provider error message

DFHPI0401 12/15/2005 15:51:12 A6POR3C1 ORDR The CICS pipeline HTTP transport mechanism failed to send a response or receive a request because the connection was closed.

DFHPI0503 12/15/2005 15:51:12 A6POR3C1 ORDR The CICS Pipeline Manager has failed to send a response on the underlying transport. TRANSPORT: HTTP, PIPELINE: EXPIPE01.

For a CICS application which is a service *requester* (Figure 3-16), timeout is controlled by the DTIMOUT attribute on the TRANSACTION definition.

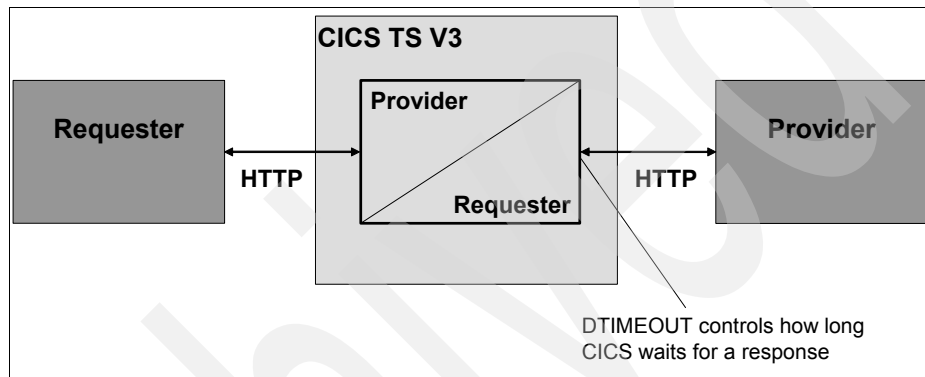


Figure 3-16 CICS timeout considerations

Note: The DTIMOUT attribute on the TRANSACTION definition only controls service requester timeout if HTTP is used as the transport mechanism.

If the request times out, CICS issues the message shown in Example 3-13.

Example 3-13 CICS service requester error message

DFHPI0504 12/15/2005 15:55:33 A6POR3C1 ORDR The CICS Pipeline Manager has failed to communicate with a remote server due to an error in the underlying transport. TRANSPORT: HTTP, PIPELINE: EXPIPE02.

Configuring the sample application

We used the catalog manager configuration transaction (ECFG) to configure the example application. Figure 3-17 shows how we changed the setting of Outbound WebService to **Yes**, and entered the URI of the service provider for our outbound service request:

`http://tx1.mop.ibm.com:13880/exampleApp/services/dispatchOrderPort`

We then pressed PF3 to save the configuration.

Tip: The 3270 terminal we used to configure the sample application had to be set to NOUCTRAN. We used the following CICS command:

```
CEOT NOUCTRAN
```

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

      Datastore Type ==> VSAM           STUB|VSAM
Outbound WebService? ==> YES         YES|NO
      Catalog Manager ==> DFHOXCMN
      Data Store Stub ==> DFHOXSDS
      Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> DFHOXWOD
      Stock Manager ==> DFHOXSSM
      VSAM File Name ==> EXMPCAT
Server Address and Port ==>
Outbound WebService URI ==> http://tx1.mop.ibm.com:13880/exampleApp/serv
                        ==> ices/dispatchOrderPort
                        ==>
                        ==>
                        ==>

APPLICATION CONFIGURATION UPDATED

PF              3  END                                12  CNCL
```

Figure 3-17 Catalog application configuration screen

With this configuration, the sample application uses the command EXEC CICS INVOKE WEBSERVICE("dispatchOrder") to invoke the dispatchOrder service which in our configuration runs in WebSphere Application Server for z/OS.

3.3.2 Configuring WebSphere Application Server for z/OS

In this section we discuss how we deployed the ExampleAppDispatchOrder service provider application on WebSphere Application Server, including:

- ▶ How we used FTP to download the ear file
- ▶ How we used the WebSphere administrative console to install the application

Downloading the EAR file

The CICS-supplied ExampleAppDispatchOrder.ear file is located in the directory:
`/usr/lpp/cicsts/cicsts31/samples/webservices/client`

We used the Windows **ftp** command shown in Example 3-14 to download the file to the workstation.

Example 3-14 Using ftp to download the EAR file

```
F:\>cd F:\Web Services Sysprog\LAN book\addmat\src\Catalog
Application\Configuration part
F:\Web Services Sysprog\LAN book\addmat\src\Catalog Application\Configuration
part>ftp mvsg3.mop.ibm.com
Connected to 9.100.193.167.
220-FTPD1 IBM FTP CS V1R6 at MVSG3.pssc.mop.ibm.com, 17:39:13 on 2005-11-24.
220 Connection will close if idle for more than 5 minutes.
User (9.100.193.167:(none)): CIWSTJ
331 Send password please.
Password:
230 CIWSTJ is logged on. Working directory is "CIWSTJ.".
ftp> cd /usr/lpp/cicsts/cicsts31/samples/webservices/client
250 HFS directory /usr/lpp/cicsts/cicsts31/samples/webservices/client is the
current working directory
ftp> get ExampleAppDispatchOrder.ear
200 Port request OK.
125 Sending data set
/usr/lpp/cicsts/cicsts31/samples/webservices/client/Example
AppDispatchOrder.ear
250 Transfer completed successfully.
ftp: 50623 bytes received in 0,03Seconds 1687,43Kbytes/sec.
ftp> bye
221 Quit command received. Goodbye.
```

Installing the service provider

CICS TS V3.1 provides a sample Web service provider application ExampleAppDispatchOrder.ear. This application archive is built at the J2EE 1.3 level. We planned to use the application in a J2EE 1.4 environment (WebSphere Application Server V6), therefore we migrated the application. We called the new application archive file dispatchOrder.ear.

Next we installed the dispatchOrder.ear file on WebSphere Application Server for z/OS. We opened a Web browser window and entered the URL:

`http://tx1.mop.ibm.com:13880/ibm/console`

After logging in, we clicked **Applications** → **Install New Application**. On the next window (Figure 3-18) we clicked **Local file system** and entered the path of the EAR file:

F:\Web Services Sysprog\LAN book\addmat\src\lears\dispatchOrder.ear

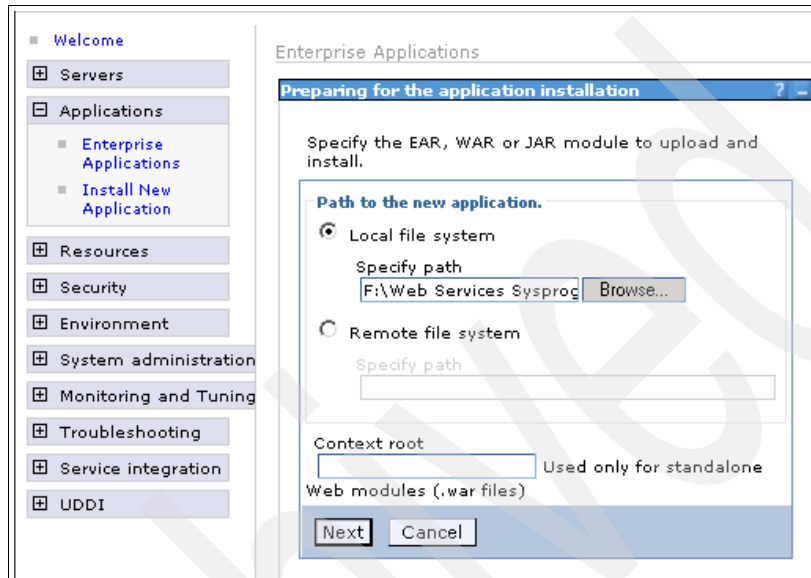


Figure 3-18 WebSphere Administrative console

We clicked **Next** → **Next** → **Next** → **Next** → **Next** → **Finish**, and then saved the changes to the master configuration.

3.3.3 Testing the configuration

In this section we discuss how we tested the configuration using the same method described in 3.2.4, “Testing the configuration” on page 102.

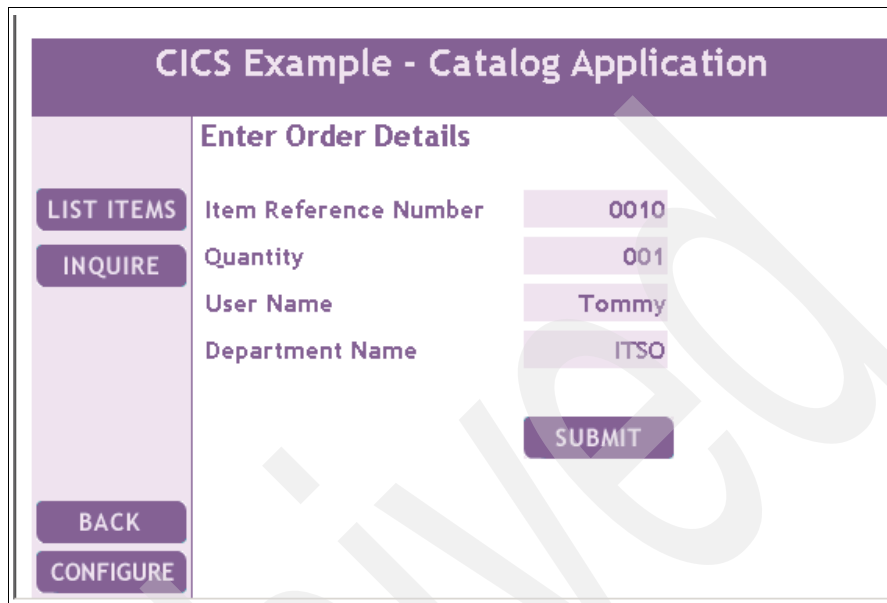
Running the Web client application

We started a Web browser session and entered the URL:

`http://cam21-pc11:9080/CatalogWeb/Welcome.jsp`

The window in Figure 3-10 on page 103 was presented, and we clicked **ORDER ITEM**.

The window in Figure 3-19 was presented. We entered values for User Name and Department Name and clicked **SUBMIT**.



CICS Example - Catalog Application		
Enter Order Details		
LIST ITEMS	Item Reference Number	0010
INQUIRE	Quantity	001
	User Name	Tommy
	Department Name	ITSO
		SUBMIT
BACK		
CONFIGURE		

Figure 3-19 CICS - Catalog application order window

We received a message back saying “ORDER SUCCESSFULLY PLACED.” We also noted that the service provider application wrote a message confirming the order to the WebSphere Application Server for z/OS SYSPRINT DD (Example 3-15).

Example 3-15 ExampleAppDispatchOrder output from WebSphere Application Server

```
DispatchOrderSoapBindingImpl: dispatchOrder(): ItemRef=10 Quantity=1  
CustomerName=Tommy Dept=ITSO
```

3.4 Configuring for high availability

After you have successfully configured and tested your CICS Web service configuration, you should consider how you can clone the CICS regions in order to improve scalability and availability.

The principal areas for consideration are:

- ▶ How to load balance TCP/IP requests across multiple CICS listener regions
- ▶ How to load balance Web service requests dynamically across multiple CICS AORs

3.4.1 TCP/IP load balancing

CICS is designed to work with Sysplex Distributor. Sysplex Distributor is an integral part of z/OS Communications Manager, which offers the ability to load balance incoming socket open requests across different address spaces running on different IP stacks (usually on different LPARs). The routing decision is based on real-time socket status and z/OS Quality of Service (QoS) criteria. This provides the benefit of balancing work across different MVS™ images, providing enhanced scalability and failover in a z/OS Parallel Sysplex®.

3.4.2 High availability configuration

Figure 3-20 shows the recommended high availability configuration. CICSplex SM provides a dynamic routing program that supports the dynamic routing of transactions. This provides the ability for applications invoked by Web service requests to be dynamically routed across a CICSplex.

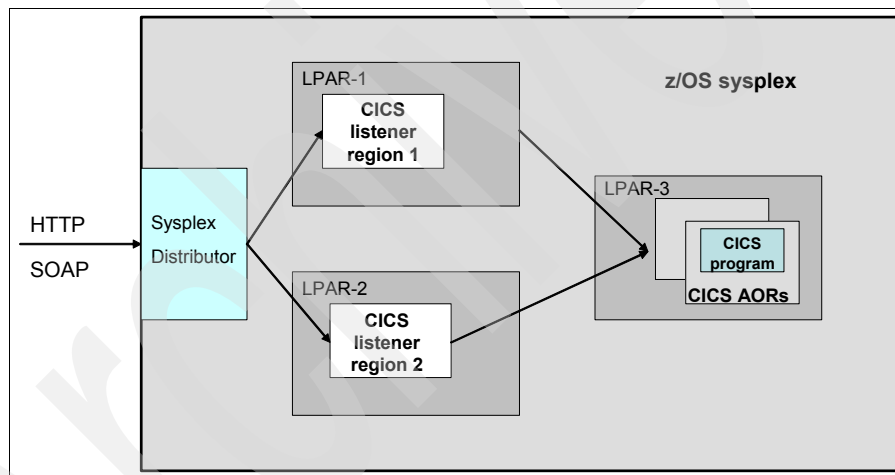


Figure 3-20 High scalability and availability configuration

3.4.3 Routing inbound Web service requests

Inbound Web service requests can be routed to a different CICS region than the one that receives the request using one of two routing models:

- ▶ Distributed routing
- ▶ Dynamic program routing

The distributed routing model

The transaction that runs the target application program is eligible for routing when one of the following is true:

- ▶ The content of the DFHWS-USERID container has been changed by a program in the pipeline.
- ▶ The content of the DFHWS-TRANID container has been changed by a program in the pipeline.
- ▶ The transaction is defined as DYNAMIC or with REMOTESYSTEM(sysid).

Figure 3-21 shows how the distributed routing model can be used to route requests for the ORDR transaction. The routing can be controlled by the routing program specified in the DSRTPGM system initialization parameter. CICSplex SM can be used to balance the routing requests across multiple AORs.

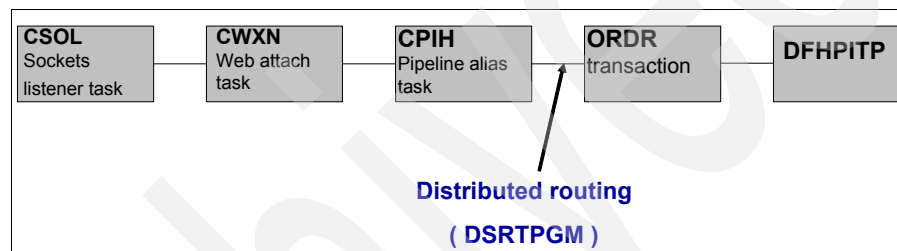


Figure 3-21 Web service provider - Distributed routing

Pipeline configuration

Special considerations have to be made when configuring a pipeline to be used in a distributed routing environment. Table 3-4 shows the resource definition requirements for both the listener region and AOR, and whether each resource definition can be shared between the regions.

Table 3-4 Pipeline resource definitions in dynamic routing configuration

Resource	Listener region	AOR
TCPIPSERVICE	Required	Not required
PIPELINE	Required, shared	Required, shared
WEBSERVICE	Automatically installed from PIPELINE, shared	Required, automatically installed from PIPELINE, shared
Pipeline configuration file	Required, shared	Required, shared
TRANSACTION definition	DYNAMIC(YES)	DYNAMIC(NO)

The dynamic routing model

An alternative way to dynamically route a Web service request, is at the point where CICS links to the user program, in our case DFH0XCMN. At this point (Figure 3-22) the request is routed using the dynamic routing model. In this scenario, the routing can be controlled by the program specified in the DTRPGM system initialization parameter. CICSplex SM can be used to balance the program link requests across multiple AORs.

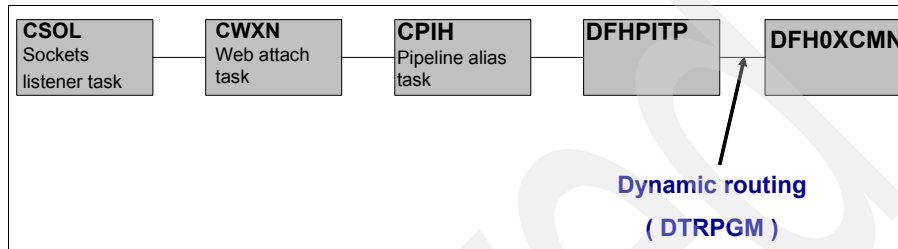


Figure 3-22 Web service provider - Dynamic routing

3.5 Problem determination

In this section we highlight different ways of diagnosing problems that occur when an incorrect URI is used in a Web services call.

3.5.1 Error calling dispatch service: INVREQ

During testing of the CICS service requester scenario we experienced the problem shown in Figure 3-23.

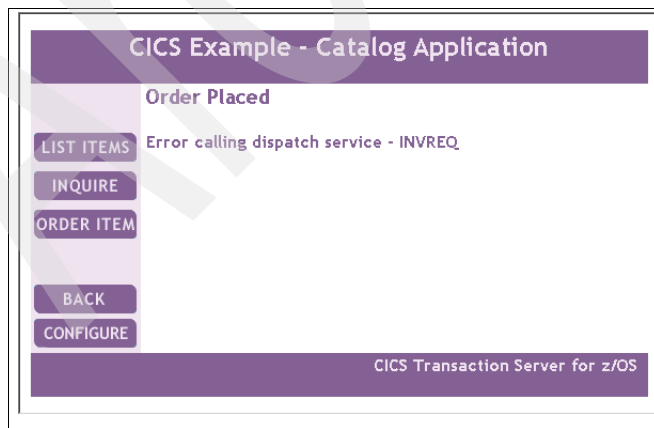


Figure 3-23 CICS - Catalog application INVREQ

Figure 3-24 shows the catalog manager configuration (including the URI for the outbound Web service).

```

CONFIGURE CICS EXAMPLE CATALOG APPLICATION

      Datastore Type ==> VSAM                STUB|VSAM
Outbound WebService? ==> YES                YES|NO
      Catalog Manager ==> DFHOXCMN
      Data Store Stub ==> DFHOXSDS
      Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> DFHOXWOD
      Stock Manager ==> DFHOXSSM
      VSAM File Name ==> EXMPCAT
Server Address and Port ==>
Outbound WebService URI ==> http://tx1.mop.ibm.com:13880/exampleApp/dispatchOrder
                        ==>
                        ==>
                        ==>

APPLICATION CONFIGURATION UPDATED

PF              3 END              12 CNCL

```

Figure 3-24 CICS - Catalog application ECFG screen

3.5.2 Diagnosing the problem

Next we discuss the troubleshooting techniques we used for the diagnosis.

CICS trace

To help diagnose the problem, we turned on CICS auxiliary trace using the CETR transaction. In trace entry 002142 of Example 3-16 we see the error returned from WebSphere Application Server:

“Error 404: SRVE0190E: File not found /services/dispatchOrder.”

Example 3-16 Sample DispatchOrder CICS trace

```

PI 0A31 PIIS EVENT - REQUEST_CNT - TASK-00182 KE_NUM-008C TCB-L8003/00ADCOA8 RET-9753E4F2
                        TIME-12:13:58.4376239562 INTERVAL-00.0000008750          =002137=
1-0000 3C534F41 502D454E 563A456E 76656C6F 70652078 6D6C6E73 3A534F41 502D454E
                                                *<SOAP-ENV:Envelope xmlns:SOAP-EN*
0020 563D2268 7474703A 2F2F7363 68656D61 732E786D 6C736F61 702E6F72 672F736F
                                                *V="http://schemas.xmlsoap.org/so*

```

```

0040 61702F65 6E76656C 6F70652F 22203E3C 534F4150 2D454E56 3A426F64 793E3C64
      *ap/envelope/" ><SOAP-ENV:Body><d*
0060 69737061 7463684F 72646572 52657175 65737420 786D6C6E 733D2268 7474703A
      *ispatchOrderRequest xmlns="http:*
0080 2F2F7777 772E6578 616D706C 65417070 2E646973 70617463 684F7264 65722E52
      *//www.exampleApp.dispatchOrder.R*
00A0 65717565 73742E63 6F6D223E 3C697465 6D526566 6572656E 63654E75 6D626572
      *equest.com"><itemReferenceNumber*
00C0 3E31303C 2F697465 6D526566 6572656E 63654E75 6D626572 3E3C7175 616E7469
      *>10</itemReferenceNumber><quanti*
00E0 74795265 71756972 65643E31 3C2F7175 616E7469 74795265 71756972 65643E3C
      *tyRequired>1</quantityRequired><*
0100 63757374 6F6D6572 49643E54 6F6D6D79 2020203C 2F637573 746F6D65 7249643E
      *customerId>Tommy </customerId>*
0120 3C636861 72676544 65706172 746D656E 743E4954 534F2020 20203C2F 63686172
      *<chargeDepartment>ITSO </char*
0140 67654465 70617274 6D656E74 3E3C2F64 69737061 7463684F 72646572 52657175
      *geDepartment></dispatchOrderRequ*
0160 6573743E 3C2F534F 41502D45 4E563A42 6F64793E 3C2F534F 41502D45 4E563A45
      *est></SOAP-ENV:Body></SOAP-ENV:E*
0180 6E76656C 6F70653E
      *nvelope>
PI 0A32 PIIS EVENT - RESPONSE_CNT - TASK-00182 KE_NUM-008C TCB-L8003/00ADC0A8 RET-9753E4F2
      TIME-12:13:58.4376326594 INTERVAL-00.0000006562 =002142=
1-0000 4572726F 72203430 343A2053 52564530 31393045 3A204669 6C65206E 6F742066
      *Error 404: SRVE0190E: File not f*
0020 6F756E64 3A202F73 65727669 6365732F 64697370 61746368 4F726465 720A
      *ound: /services/dispatchOrder. *

```

Using SNIFFER

The user-written SNIFFER handler program is a simple program that browses through the containers available in the pipeline. It can be used as a message handler program or a header processing program.

It browses the containers by issuing a STARTBROWSE CONTAINER command followed by GETNEXT CONTAINER until all containers have been browsed. It then issues an ENDBROWSE CONTAINER command. For each container browsed, it writes the container name and contents to the CICS transient data queue CESE.

We added the SNIFFER message handler program to the requester pipeline EXPIPE02. Example 3-17 shows the pipeline configuration file with SNIFFER added as a message handler program.

Example 3-17 Pipeline configuration file with SNIFFER

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/http/cics/pipeline
  requester.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
  <default_transport_handler_list>
    <handler>
      <program>SNIFFER</program>
      <handler_parameter_list/>
    </handler>
  </default_transport_handler_list>
</requester_pipeline>
```

Example 3-18 shows the containers in the requester pipeline as listed by SNIFFER. The container of interest is DFHWS-URI:

<http://tx1.mop.ibm.com:13880/exampleApp/dispatchOrder>

Example 3-18 sample SNIFFER output

```
SNIFFER : *** Start ***
SNIFFER : >=====
SNIFFER : Container Name : DFHFUNCTION
SNIFFER : Content length : 00000016
SNIFFER : Container content: SEND-REQUEST
SNIFFER : Containers on channel: List starts.
SNIFFER : >=====
SNIFFER : Container Name : DFHHEADER
SNIFFER : Content length : 00000000
SNIFFER : Container EMPTY
SNIFFER : >=====
SNIFFER : Container Name : DFHWS-XMLNS
SNIFFER : Content length : 00000059
SNIFFER : Container content: xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
envelope/"
SNIFFER : >=====
SNIFFER : Container Name : DFHWS-SOAPLEVEL
SNIFFER : Content length : 00000004
SNIFFER : Container content:
SNIFFER : >=====
SNIFFER : Container Name : DFH-HANDLERPLIST
SNIFFER : Content length : 00000000
SNIFFER : Container EMPTY
```

```

SNIFFER : >=====<
SNIFFER : Container Name   : DFHRESPONSE
SNIFFER : Content length   : 00000000
SNIFFER : Container EMPTY
SNIFFER : >=====<
SNIFFER : Container Name   : DFHFUNCTION
SNIFFER : Content length   : 00000016
SNIFFER : Container content: SEND-REQUEST
SNIFFER : >=====<
SNIFFER : Container Name   : DFH-SERVICEPLIST
SNIFFER : Content length   : 00000000
SNIFFER : Container EMPTY
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-USERID
SNIFFER : Content length   : 00000008
SNIFFER : Container content: CICSUSER
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-TRANID
SNIFFER : Content length   : 00000004
SNIFFER : Container content: ORDR
SNIFFER : >=====<
SNIFFER : Container Name   : DFHREQUEST
SNIFFER : Content length   : 00000000
SNIFFER : Container EMPTY
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-BODY
SNIFFER : Content length   : 00000293
SNIFFER : Container content: <SOAP-ENV:Body><dispatchOrderRequest
xmlns="http://www.exampleApp.dispatchOrder.Request.com"><itemRefere
renceNumber>10</itemReferenceNumber><quantityRequired>1</quantityRequired><cust
omerId>Tommy </customerId><chargeDepartment>ITS0
</chargeDepartment></dispatchOrderRequest></SOAP-ENV:Body>
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-URI
SNIFFER : Content length   : 00000255
SNIFFER : Container content:
http://tx1.mop.ibm.com:13880/exampleApp/dispatchOrder
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-SOAPACTION
SNIFFER : Content length   : 00000002
SNIFFER : Container content: ""
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-OPERATION
SNIFFER : Content length   : 00000255
SNIFFER : Container content: dispatchOrder
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-PIPELINE
SNIFFER : Content length   : 00000008
SNIFFER : Container content: EXPIPE02

```



```
SNIFFER : >=====<
SNIFFER : Container Name   : DFHWS-DATA
SNIFFER : Content length   : 00000023
SNIFFER : Container content: 0010001Tommy  ITS0
SNIFFER : Containers on channel: List ends
SNIFFER : in a SOAP header processing program.....
SNIFFER : **** End ****
```

Checking the SOAP address in the WSDL

Next we checked the SOAP address in the WSDL file of the deployed EAR file. In the WebSphere administrative console we clicked **Applications** → **Enterprise applications** → **dispatchOrder** → **Publish WSDL file** → **DispatchOrder_WSDLFiles.zip** and saved the file to disk. We unzipped the file into the directory structure shown in Figure 3-25.

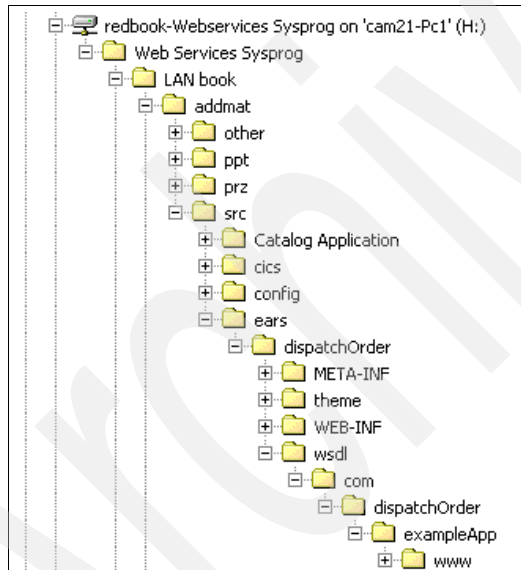


Figure 3-25 ExampleAppDispatchOrder path

In the WSDL file dispatchOrder.wsdl (Example 3-19) we noted the URI of the Web service as found in the soap: address location.

Example 3-19 dispatchOrder sample WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.exampleApp.dispatchOrder.com"
  xmlns:tns="http://www.exampleApp.dispatchOrder.com"
  xmlns:reqns="http://www.exampleApp.dispatchOrder.Request.com"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:resns="http://www.exampleApp.dispatchOrder.Response.com"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://www.exampleApp.dispatchOrder.Request.com"
      xmlns:reqns="http://www.exampleApp.dispatchOrder.Request.com"
      xmlns:tns="http://www.exampleApp.dispatchOrder.Request.com"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="dispatchOrderRequest" nillable="false">
      .
    .
  </xsd:schema>
  .
  Part of wsdl not included
  .
  <binding name="dispatchOrderSoapBinding" type="tns:dispatchOrderPort">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/
      soap/http"/>
    <operation name="dispatchOrder">
      <soap:operation soapAction="" style="document"/>
      <input name="DFHOXODSRequest">
        <soap:body parts="RequestPart" use="literal"/>
      </input>
      <output name="DFHOXODSResponse">
        <soap:body parts="ResponsePart" use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="dispatchOrderService">
    <port name="dispatchOrderPort" binding="tns:dispatchOrderSoapBinding">
      <soap:address location="http://tx1.mop.ibm.com:13880/
        exampleApp/services/dispatchOrderPort"/>
    </port>
  </service>
</definitions>
```

The error shown in Figure 3-24 on page 117 was caused by specifying an incorrect URI for the dispatchOrder Web service. In the catalog manager configuration (Figure 3-24) we specified the URI `/exampleApp/dispatchOrderPort` for the outbound Web service. This is the correct URI for the dispatchOrder service provider deployed inside CICS, but a URI of `/exampleApp/services/dispatchOrderPort` is the correct URI for our dispatchOrder service deployed in WebSphere Application Server.



Web services using WebSphere MQ

In this chapter, we describe how we configured our test CICS environment to support Web services using WebSphere MQ as the transport mechanism.

4.1 Preparation

After outlining our test configuration (Figure 4-1), we show how we enabled WebSphere MQ (WMQ) support in a CICS region. We next explain how we configured CICS as a service provider for incoming WMQ message requests. Finally, we show how we configured a CICS region to act as a service requester, sending requests in WMQ messages.

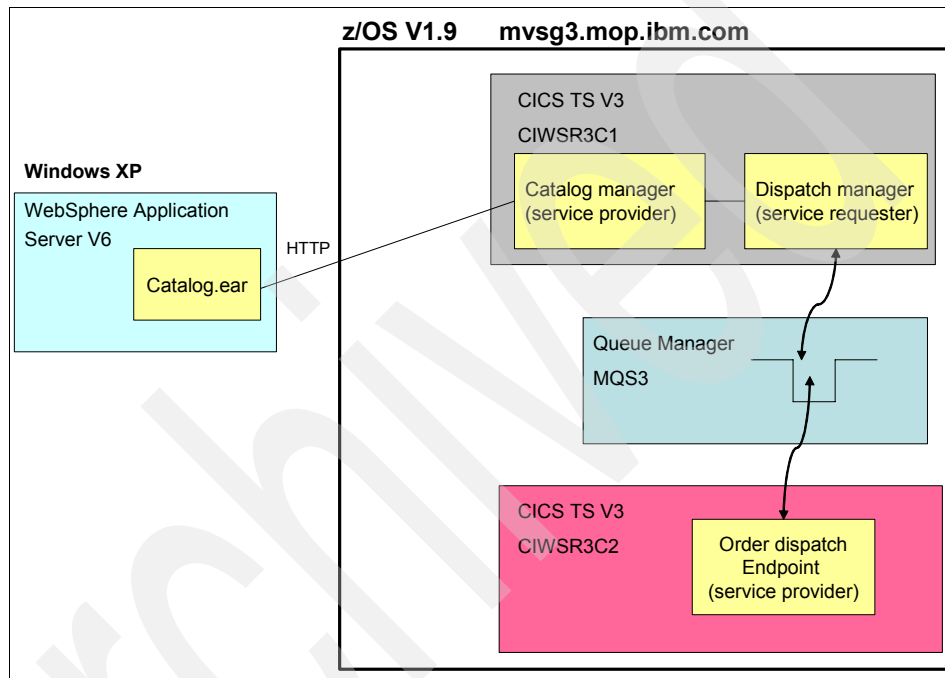


Figure 4-1 Software components: Web services using HTTP and WMQ

Note: We used a CICS-to-CICS scenario in order to demonstrate how WMQ can be used with a CICS service provider and a service requester. You can also use WMQ to pass SOAP messages between WebSphere Application Server and CICS.

We do not provide details on how to install the software components, and we also assume that the reader has a working knowledge of CICS and WebSphere MQ.

4.1.1 Software checklist

For the configuration shown in Figure 4-1 we used the levels of software shown in Table 4-1.

Table 4-1 Software used in the WebSphere MQ scenarios

Windows	z/OS
Windows XP SP2	z/OS V1.9
IBM WebSphere Application Server - ND V6.1.0.17	CICS Transaction Server V3
Internet Explorer V6.0	
	WebSphere MQ V6R0M0
Our J2EE applications ▶ Catalog.ear Catalog manager service requester application	Our user-supplied CICS programs ▶ CIWSMSGH (message handler program)

4.1.2 Definition checklist

The z/OS definitions we used to configure the scenarios are listed in Table 4-2.

Table 4-2 Definition checklist

Value	CICS region 1	CICS region 2
IP name	mvsg3.mop.ibm.com	mvsg3.mop.ibm.com
IP address	9.100.193.167	9.100.193.167
TCP/IP port	13301	
Job name	CIWSR3C1	CIWSR3C2
APPLID	A6POR3C1	A6POR3C2
TCPIP SERVICE	R3C1	
Provider PIPELINE	EXPIPE01	EXPIPEP03
Requester PIPELINE	EXPIPE02	
WMQ queue manager	MQS3	MQS3

The WMQ definitions we used to configure the scenarios are listed in Table 4-3.

Table 4-3 WMQ definition checklist

Value	Queue manager MQS3
Queues	V3G3.R3C2.PIPE3.REQUEST V3G3.R3C2.PIPE3.RESPONSE
Process	VSG3.R3C2.PROCESS

4.2 WebSphere MQ configuration

We completed the following tasks in order to enable WMQ support in the two CICS regions CIWSR3C1 and CIWSR3C2:

- ▶ Adding WMQ support to CICS
- ▶ Defining the queues
- ▶ Defining the trigger process

4.2.1 Adding WebSphere MQ support to CICS

We updated the CICS startup procedure for each CICS region by adding the WMQ libraries to the STEPLIB and DFHRPL as shown in Example 4-1.

Example 4-1 CICS startup JCL

```
//STEPLIB DD DSN=CICSTS31.CICS.SDFHAUTH,DISP=SHR
//          DD DSN=CICSTS31.CICS.SDFJAUTH,DISP=SHR
//          DD DSN=MQM.SCSQANLE,DISP=SHR
//          DD DSN=MQM.SCSQAUTH,DISP=SHR
//DFHRPL   DD DSN=CIWS.CICS.USERLOAD,DISP=SHR
//          DD DSN=CEE.SCEECICS,DISP=SHR
//          DD DSN=CEE.SCEERUN2,DISP=SHR
//          DD DSN=CEE.SCEERUN,DISP=SHR
//          DD DSN=CICSTS31.CICS.SDFHLOAD,DISP=SHR
//          DD DSN=MQM.SCSQLOAD,DISP=SHR
//          DD DSN=MQM.SCSQANLE,DISP=SHR
//          DD DSN=MQM.SCSQCICS,DISP=SHR
//          DD DSN=MQM.SCSQAUTH,DISP=SHR
```

- ▶ We updated the SIT parameters on CICS region CIWSR3C1:
 - MQCONN=YES
 - INITPARM=(DFHMQPRM='SN=MQS3,IQ=VSG3.R3C1.INITQ')
- ▶ We updated the SIT parameters on CICS region CIWSR3C2:
 - MQCONN=YES
 - INITPARM=(DFHMQPRM='SN=MQS3,IQ=VSG3.R3C2.INITQ')
- ▶ We added the CICS supplied WMQ RDO groups to the startup LIST on CICS region 1 using the following commands, and then we restarted the CICS region:


```
CEDA ADD GROUP(DFHMQ) TO LIST(LISTR3C2)
```
- ▶ We added the CICS supplied WMQ RDO groups to the startup LIST on CICS region 2 using the following commands, and then we restarted the CICS region:


```
CEDA ADD GROUP(DFHMQ) TO LIST(LISTR3C2)
```

4.2.2 Defining the queues

Example 4-2 shows the JCL that we used to define two QUEUE resources of type local in the MQS3 queue manager region. One queue is for incoming requests and the other is for responses.

Example 4-2 JCL for defining the queues

```
//CHIQUEUE JOB 1,CIWS,TIME=1440,NOTIFY=&SYSUID,REGION=4M,
//          CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//*
//CSQUTIL EXEC PGM=CSQUTIL,PARM='MQS3'
//STEPLIB DD DSN=MQM.SCSQLOAD,DISP=SHR
//          DD DSN=MQM.SCSQANLE,DISP=SHR
//          DD DSN=MQM.SCSQAUTH,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(CMDINP)
/*
//CMDINP DD *
*
DEFINE QLOCAL(VSG3.R3C2.PIPE3.REQUEST) -
DESCR('QUEUE SOAP INCOMING REQUEST') -
PROCESS(VSG3.R3C2.PROCESS) -
```

```
TRIGGER -  
TRIGTYPE(FIRST) -  
INITQ('VSG3.R3C2.INITQ') -  
*  
DEFINE QLOCAL(VSG3.R3C2.PIPE3.RESPONSE) -  
DESCR('QUEUE SOAP RESPONSE') -  
*  
/*
```

The INITQ VSG3.R3C2.INITQ is the same name as specified in the INITPARM parameter for the CIWSR3C2 region.

4.2.3 Defining the trigger process

Example 4-3 shows the command that we used to define a PROCESS.

Example 4-3 WMQ definition of PROCESS

```
DEFINE PROCESS(VSG3.R3C2.PROCESS)  
APPLTYPE(CICS)  
APPLICID(CPIL)
```

The process name is the same name specified when defining the request queue VSG3.R3C2.PIPE3.REQUEST in Example 4-2. APPLICID is specified as CPIL (the SOAP MQ inbound listener transaction), which means that this transaction is started in CICS when a service request arrives. CPIL matches an incoming URI to a URIMAP definition in order to match the URI to a WEBSERVICE, and attaches the CPIQ transaction (the SOAP MQ inbound router transaction).

4.3 Configuring CICS as a service provider using WMQ

In this section we discuss how we configured the CICS region CIWSR3C2 as a service provider using WMQ (Figure 4-2).

4.3.1 Interface for dispatching an order

The catalog manager application provides a dispatch manager program that provides an interface for dispatching an order to an external partner. In this scenario, we configured a remote order dispatch endpoint, such that the dispatch request is sent to a CICS service provider program DFH0XODE using WMQ.

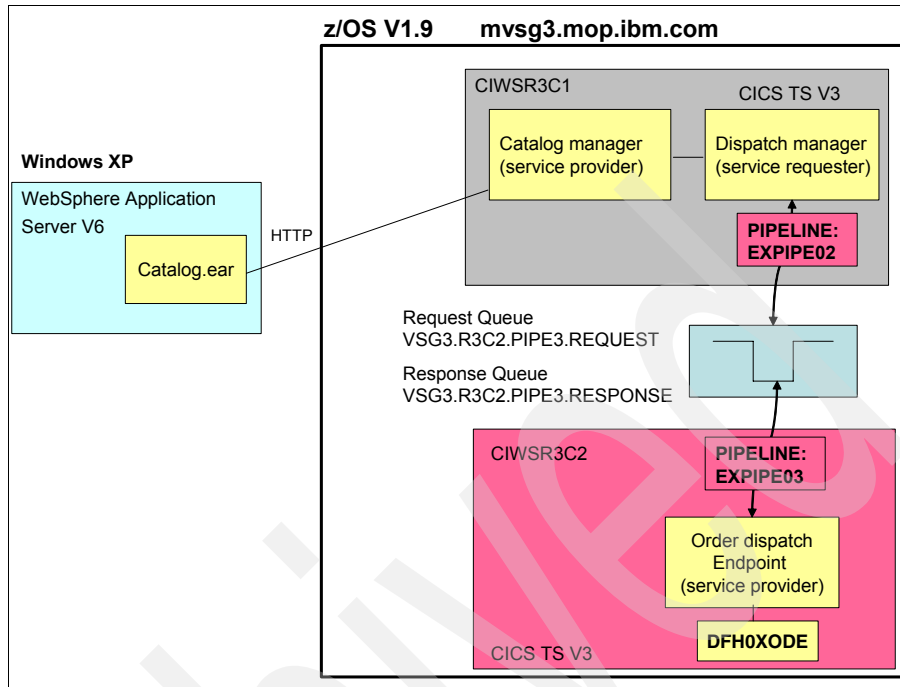


Figure 4-2 CICS as a service provider using WMQ

Note: On our system the two CICS regions are actually running on the same z/OS image. In practice they would normally be running on two different systems.

4.3.2 Configuring the service provider pipeline

To enable CICS to receive Web service requests using WMQ we performed the following tasks:

- ▶ Creating the HFS directories
- ▶ Configuring the pipeline configuration file
- ▶ Updating the message handler program CIWSMSGH
- ▶ Creating and installing the PIPELINE resource definition

Creating the HFS directories

We created the HFS directories shown in Example 4-4.

Example 4-4 HFS directories used in the PIPELINE definition

```
/CIWS/R3C2/config  
/CIWS/R3C2/shelf  
/CIWS/R3C2/wsbinding/provider
```

We copied the dispatchOrderEndPoint.wsbinding file from the CICS-supplied directory /usr/lpp/cicsts/cicsts31/samples/webservices/wsbinding/provider to our wsbinding directory /CIWS/R3C2/wsbinding/provider.

The config directory is used for the pipeline configuration file that we create in a subsequent step, while CICS uses the shelf directory to store installed wsbinding files.

We gave the CICS region user ID read permission to the config and wsbinding directories, and update permission to the shelf directory.

Configuring the pipeline configuration file

In order to add the CIWSMSGH message handler program to the pipeline for the service provider, we used the same pipeline configuration file that is described in “Customizing the pipeline configuration file” on page 92.

Updating the message handler program

The default transaction ID assigned to inbound WMQ Web services transactions is CPIQ. We wanted to assign a different transaction ID to the dispatch request. To do this, we updated the CIWSMSGH message handler program that we first introduced in “Writing the message handler program” on page 93. We replaced the transaction ID in the DFHWS-TRANID container with an ID based on the service requester (which can be found in the DFHWS-WEBSERVICE container).

Table 4-4 shows the relationship between the transaction ID and the Web service request.

Table 4-4 Transaction ID to Web services name relationship

Transaction ID	Web services request
DISP	dispatchOrderEndPoint

Before we activated the message handler program, we had to create the new TRANSACTION definition with the same characteristics as the CICS-supplied definition for CPIQ.

We used this CEDA COPY command to create the transaction definition:

```
CEDA COPY TRANSACTION(CPIQ) GROUP(DFHPIPE) TO(R3C2) AS(DISP)
```

Then we installed the definition.

Creating the PIPELINE resource definition

We then defined the PIPELINE for the CICS service provider using the following CICS command:

```
CEDA DEFINE PIPELINE(EXPIPE03) GROUP(R3C2)
```

We defined EXPIPE03 as shown in Figure 4-3.

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0650
CEDA DEFine PIpeline(EXPIPE03 )

  PIpeline      : EXPIPE03
  Group        : R3C2
  Description   ==>
  SStatus      ==> Enabled           Enabled | Disabled
  Respwait     ==> Deft             Default | 0-9999
  Configfile   ==> /CIWS/R3C2/config/basicsoap12provider.xml
  (Mixed Case) ==>
              ==>
              ==>
              ==>
  SHeIf        ==> /CIWS/R3C2/shelf
  (Mixed Case) ==>
              ==>
              ==>
              ==>
  Wsdir        : /CIWS/R3C2/wsbind/provider
  (Mixed Case) :
```

Figure 4-3 CEDA DEFINE PIPELINE EXPIPE03

- ▶ We set CONFIGFILE to the name of our pipeline configuration file:
/CIWS/R3C2/config/basicsoap12provider.xml
- ▶ We set SHELF to the name of the shelf directory:
/CIWS/R3C2/shelf
- ▶ We set WSDIR to the Web service binding directory that contains the wsbind files for the sample application:
/CIWS/R3C2/wsbind/provider

Installing the PIPELINE resource

We used CEDA to install the PIPELINE definition. When the PIPELINE is installed, CICS scans the wsdir directory and dynamically creates a WEBSERVICE and a URIMAP definition for each wsbind file that it finds.

Figure 4-4 shows a CEMT INQUIRE PIPELINE for EXPIPE03.

```
INQUIRE PIPELINE
RESULT - OVERTYPE TO MODIFY
  Pipeline(EXPIPE03)
  Enablestatus( Enabled )
  Mode(Provider)
  Mtomst(Nomtom)
  Sendmtomst(Nosendmtom)
  Mtomnoxopst(Nomtomnoxop)
  Xopsupportst(Noxopsupport)
  Xopdirectst(Noxopdirect)
  Soaplevel(1.2)
  Respwait(      )
  Configfile(/CIWS/C3C2/config/basicsoap12provider.xml)
  Shelf(/CIWS/C3C2/shelf/)
  Wsdir(/CIWS/C3C2/wsbind/provider/)
  Ciddomain(cicsts)
SYSID=R3C2 APPLID=A6P0R3C2
```

Figure 4-4 CEMT INQUIRE PIPELINE - EXPIPE03

After installing the pipeline, we used the CEMT INQUIRE WEBSERVICE command to view the dynamically installed Web service. In Figure 4-5, we noted that the name of the service (namely, dispatchOrderEndpoint) is taken from the wsbind file.

```
INQUIRE WEBSERVICE
STATUS: RESULTS - OVERTYPE TO MODIFY
  Webs(dispatchOrderEndpoint      ) Pip(EXPIPE03)
  Ins Ccs(00000) Uri(£439310 ) Pro(DFH0X0DE) Com
SYSID=R3C2 APPLID=A6P0R3C2
```

Figure 4-5 CEMT INQUIRE WEBSERVICE

Figure 4-6 shows the dynamically installed URIMAP that is associated with the Web service.

```
INQUIRE URIMAP
RESULT - OVERTYPE TO MODIFY
  Urimap(£439310)
  Usage(Pipe)
  Enablestatus( Enabled )
  Analyzerstat(Noanalyzer)
  Scheme(Http)
  Redirecttype( None )
  Tcpiptype()
  Host(*)
  Path(/exampleApp/dispatchOrder)
  Transaction(CPIH)
  Converter()
  Program()
  Pipeline(EXPIPE03)
  Webservice(dispatchOrderEndpoint)
  Userid()
  Certificate()
  Ciphers()
  Templatenamename()
```

SYSID=R3C2 APPLID=A6P0R3C2

Figure 4-6 CEMT INQUIRE URIMAP

4.4 Configuring CICS as service requester using WMQ

In this section we explain how we configured CICS region CIWSR3C1 as a service requester using WMQ. In this particular scenario CICS region CIWSR3C1 is both a service provider for the catalog manager application (as detailed in “Configuring CICS as a service provider” on page 88) and a service requester of the Dispatch manager application.

Figure 4-7 shows the Dispatch manager order dispatch program DFH0XWOD, which issues an EXEC CICS INVOKE WEBSERVICE command to make an outbound Web service call to the order dispatcher running in CICS region CIWSR3C2.

When communication between the service requester and service provider uses WMQ, the URI of the target is in a form that identifies the target as a queue, and includes information to specify how the request and response should be handled by WMQ.

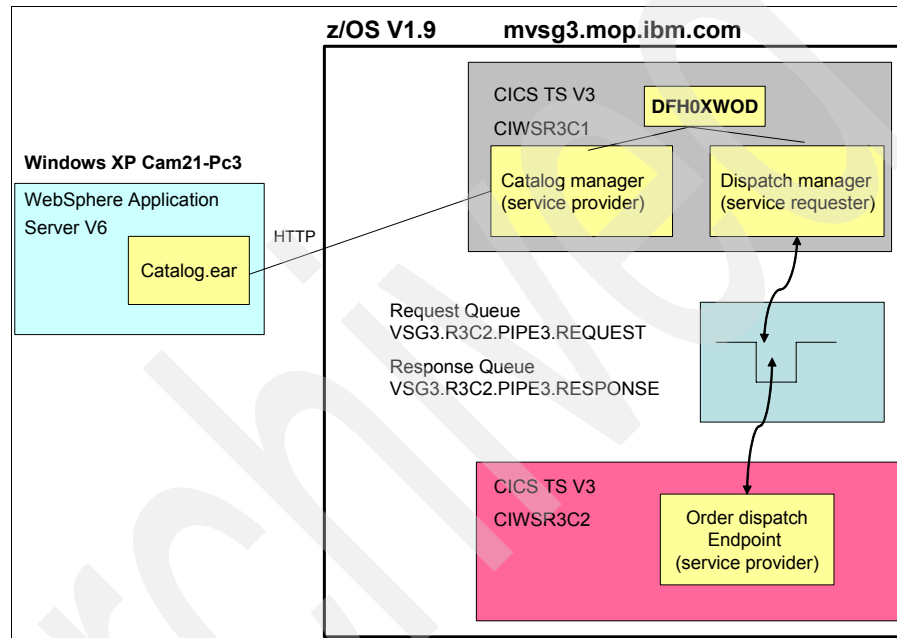


Figure 4-7 CICS as a service provider using WMQ

4.4.1 Configuring the catalog application

We configured the catalog application to activate the outbound Web services feature using WMQ. We used the CICS-supplied catalog manager configuration transaction ECFG to configure the example application (Figure 4-8).

```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION

      Datastore Type ==> VSAM                STUB|VSAM
Outbound WebService? ==> YES                YES|NO
      Catalog Manager ==> DFHOXCMN
      Data Store Stub ==> DFHOXSDS
      Data Store VSAM ==> DFHOXVDS
      Order Dispatch Stub ==> DFHOXSOD
Order Dispatch WebService ==> DFHOXWOD
      Stock Manager ==> DFHOXSSM
      VSAM File Name ==> EXMPCAT
Server Address and Port ==> 9.100.193.167:13301
Outbound WebService URI ==> jms:/queue?destination=VSG3.R3C2.PIPE3.REQUE
                          ==> ST@MQS3&targetService=/exampleApp/dispatchOr
                          ==> der&replyDestination=VSG3.R3C2.PIPE3.RESPONS
                          ==> E
                          ==>

PF                3  END                12  CNCL
```

Figure 4-8 Catalog application configuration screen for WMQ

We changed Outbound WebService to **Yes**, and entered the URI of the service provider for our outbound service request:

```
jms:/queue?destination=VSG3.R3C2.PIPE3.REQUEST@MQS3&targetService=/exampleApp/dispatchOrder&replyDestination=VSG3.R3C2.PIPE3.RESPONSE
```

We then pressed PF3 to save the configuration.

Tip: You must use the ampersand (&) character as a separator between options; otherwise CICS does not recognize the parameters.

The Dispatch manager module DFHOXWOD uses the value of the **Outbound WebService URI** parameter as the URI of the Web service to be invoked when it invokes the dispatch service with an EXEC CICS INVOKE WEBSERVICE command.

The main parameters for the Outbound WebService URI are as follows:

- ▶ **jms:/** : A specific URI format to use WMQ.
- ▶ **destination:** VSG3.R3C2.PIPE3.REQUEST@MQS3 is a concatenation of the target queue name and the queue manager name.
- ▶ **targetService:** /exampleApp/dispatchOrder is the target service in CIWSR3C2 (it matches the dynamically installed URIMAP shown in Figure 4-6 on page 133).

Tip: If you do not want to specify the targetService in URI data, you can pass the same information by setting /exampleApp/dispatchOrder as the TRIGDATA attribute of the receive queue VSG3.R3C2.PIPE3.REQUEST.

- ▶ **replyDestination:** VSG3.R3C2.PIPE3.RESPONSE is the reply queue name for the response.

Important: When the URI specified on a EXEC CICS INVOKE WEBSERVICE command begins with **jms:/**, CICS uses WMQ rather than HTTP to send the request. The application program itself does not have to be aware that WMQ is being used as the transport mechanism in place of HTTP.

Timeout considerations

It is not possible to manage timeout for a WMQ service requester application by specifying a timeout value on the URI. We tested different values for the timeout parameter and found that it always timed out after one minute.

For further information about using WMQ to transport SOAP messages, see *WebSphere MQ - Transport for SOAP*, SC34-6651.

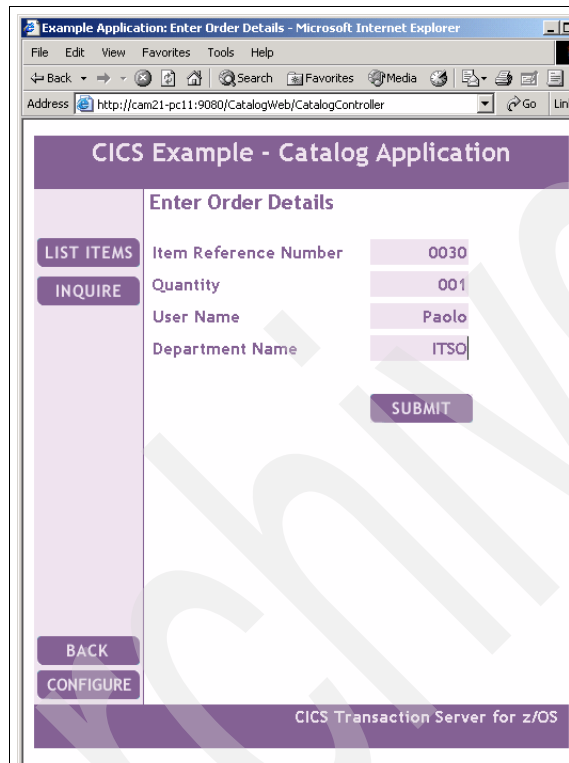
4.4.2 Configuring WebSphere Application Server on Windows

We deployed the catalog manager service requester application (catalog.ear) to WebSphere Application Server for Windows XP as documented in 3.2.3, “Configuring WebSphere Application Server on Windows” on page 99.

4.5 Testing the WMQ configuration

To test our WMQ setup, we used a Web browser to run the Catalog application as described in “Testing the configuration” on page 102.

Figure 4-9 shows order details for our request.



The screenshot shows a Microsoft Internet Explorer browser window titled "Example Application: Enter Order Details - Microsoft Internet Explorer". The address bar displays "http://cam21-pc11:9080/CatalogWeb/CatalogController". The main content area is titled "CICS Example - Catalog Application" and "Enter Order Details". On the left side, there are buttons for "LIST ITEMS", "INQUIRE", "BACK", and "CONFIGURE". The main form contains the following fields and values:

Item Reference Number	0030
Quantity	001
User Name	Paolo
Department Name	ITSO

Below the form is a "SUBMIT" button. At the bottom of the page, it says "CICS Transaction Server for z/OS".

Figure 4-9 Catalog application - ORDER function

From a CICS 3270 screen, we used the CICS Execution Diagnostic Facility (EDF) to intercept the DISP transaction on CICS region CIWSR3C2. We then used the CEMT INQUIRE TASK command to view the inflight transactions (Figure 4-10).

```

INQUIRE TASK
STATUS: RESULTS - OVERTYPE TO MODIFY
Tas(0000026) Tra(CKAM)          Sus Tas Pri( 255 )
    Sta(SD) Use(CIWS3D ) Uow(BE0772E8804C9C2B)
Tas(0000344) Tra(CKTI)          Sus Tas Pri( 001 )
    Sta(SD) Use(CICSUSER) Uow(BE0CE7454196FC8B) Hty(MQSeries)
Tas(0000386) Tra(CEMT) Fac(G350) Run Ter Pri( 255 )
    Sta(TO) Use(CICSUSER) Uow(BE0CE689E4ECE06F)
Tas(0000388) Tra(CPIL)          Sus Tas Pri( 001 )
    Sta(SD) Use(CICSUSER) Uow(BE0CE7454267F84B) Hty(MQSeries)
Tas(0000389) Tra(CPIQ)          Sus Tas Pri( 001 )
    Sta(S ) Use(CICSUSER) Uow(BE0CE745420A3860) Hty(RZCBNOTI)
Tas(0000390) Tra(DISP)          Sus Tas Pri( 001 )
    Sta(U ) Use(CICSUSER) Uow(BE0CE74542800740) Hty(EDF )
Tas(0000392) Tra(CEDF) Fac(G353) Sus Ter Pri( 001 )
    Sta(SD) Use(CICSUSER) Uow(BE0CE74542C74820) Hty(ZCIOWAIT)

SYSID=R3C2 APPLID=A6P0R3C2

```

Figure 4-10 CEMT INQUIRE TASK

Figure 4-10 shows the inflight transactions:

- ▶ The SOAP MQ inbound listener transaction (CPIL)
- ▶ The SOAP MQ inbound router transaction (CPIQ)
- ▶ The transaction used for running the business logic program (DISP)

After ending the EDF session, we received the ORDER SUCCESSFULLY PLACED response in the browser.

We noted the SYSPRINT messages by the CIWSMSGH message handler for CICS region CIWSR3C2 (Example 4-5).

Example 4-5 CICS CIWSR3C2 - SYSPRINT

```

CIWSMSGH: >=====<
CIWSMSGH: Container Name: : DFHWS-WEBSERVICE
CIWSMSGH: Container content: dispatchOrderEndpoint
CIWSMSGH: -----
CIWSMSGH: Container Name: : DFHWS-TRANID
CIWSMSGH: Container content: DISP
CIWSMSGH: -----
CIWSMSGH: Container Name: : DFHWS-URI
CIWSMSGH: Container content: wmq:VSG3.R3C2.PIPE3.REQUEST/exampleApp/dispatchOrder

```

Note that after the request for the `dispatchOrderEndpoint` service arrives, the message handler changes the transaction ID to `DISP`. In particular, the `DFHWS-URI` container shows the URI in WMQ format.

4.6 High availability with WMQ

In 3.4, “Configuring for high availability” on page 113 we outlined how HTTP Web service requests can be balanced across multiple CICS regions in order to provide a high availability configuration. Here we take a brief look at how WMQ Web service requests can be balanced across multiple CICS regions.

On page 113, we also discussed how, after a request is received by a specific CICS region, it can be dynamically routed within a CICSplex. These transaction and program routing mechanisms can be used irrespective of how the SOAP message is transported.

The principal areas for consideration when designing a high availability configuration for WebSphere MQ are:

- ▶ How to share access to queues across multiple CICS regions
- ▶ How to load balance WMQ connections across multiple queue managers

Figure 4-11 shows an example high availability configuration for WMQ, in which queues shared in the coupling facility can be accessed by CICS regions running on different LPARs, and WMQ connections are balanced across different queue managers using shared channels.

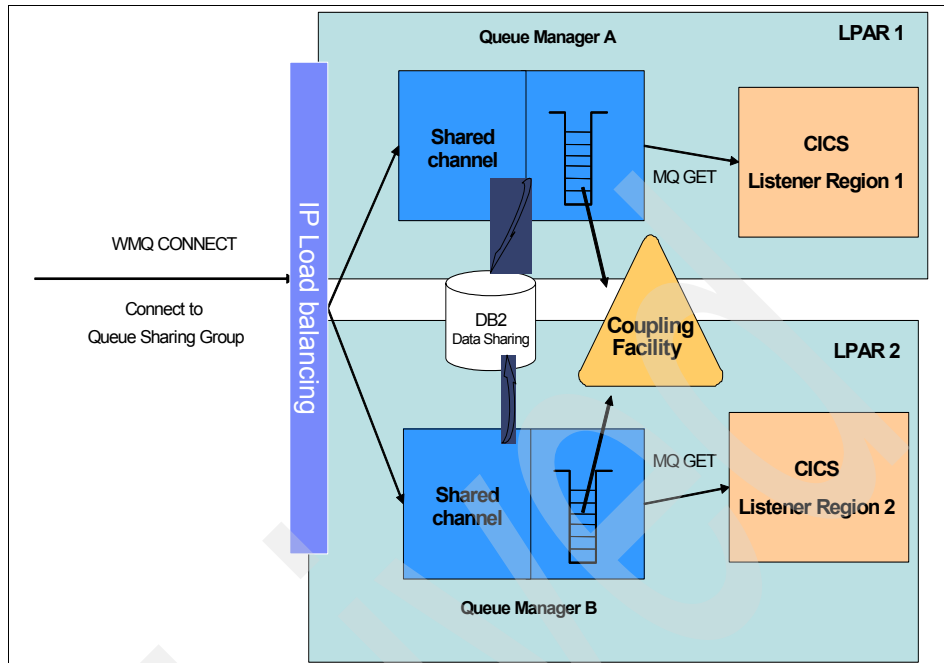


Figure 4-11 High availability configuration for WMQ

Figure 4-11 shows an example WMQ configuration that takes advantage of several parallel sysplex high availability capabilities, specifically:

► **Shared queues**

A *shared queue* is a type of queue in which messages on that queue can be accessed by one or more queue managers that are identified to the sysplex. The queue managers that can access the same set of shared queues form a group called a *queue-sharing group* (QSG).

A QSG controls which queue managers can access which coupling facility list structures and hence, which shared queues. Each coupling facility list structure is owned by a QSG and can only be accessed by queue managers in that QSG.

Multiple queue managers on multiple MVS images within the same queue-sharing group can put messages to and get messages from the same shared queue. This is achieved by storing all the messages in a shared queue in the same coupling facility list structure.

Multiple queue managers on multiple MVS images within the same queue-sharing group can access the same WebSphere MQ objects. This is achieved by storing the object definitions in tables of a DB2 data-sharing group.

The use of shared queues provides a highly available solution because the failure of a single MVS image does not prevent access to shared queues. Another benefit is a capability to implement pull workload balancing. It means that by defining the input queue of an application (such as a CICS service provider application) as a shared queue, you make any message put to that queue available to be retrieved by any queue manager in the queue-sharing group.

► **Shared channels**

The advantage of using *shared channels* is high availability when compared to being connected to a single queue manager. An inbound channel is classed as shared if it is connected to the queue manager through a group listener. A group listener is an additional task started on each channel initiator in the queue-sharing group. This task listens on an IP address/port combination, specific to that queue manager, known as its group address. Each group address can then be registered with an IP routing mechanism such as Sysplex Distributor.

► **Sysplex Distributor**

Sysplex Distributor is designed to address the requirement of one single network-visible IP address for a service. Sysplex distributor can be used to map a queue-sharing group-wide generic IP address/port to a specific group address.

For more information about configuring high availability with WMQ, refer to *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864.

MTOM/XOP optimization

In this chapter we describe how CICS can be configured to support MTOM/XOP optimization of binary data.

We use a modified version of the CICS catalog example application that has been extended to display the corresponding catalog item image on the Web client front end. See *CICS Transaction Server V3R1 Channels and Containers Revealed*, SG24-7227 for more details on this modified application.

5.1 Preparation

In this chapter we describe how we configure CICS to use MTOM/XOP optimization of binary data in a SOAP message. We describe the changes we have to make to the configuration we used in Chapter 3, “Web services using HTTP” on page 85. This includes:

- ▶ Changing the pipeline configuration file
- ▶ Modifying the PIPELINE definition
- ▶ Rebuilding the WSBIND file
- ▶ Modifying the application

5.1.1 Software checklist

We used the levels of software shown in Table 5-1.

Table 5-1 Software used in the MTOM/XOP scenario

Windows	z/OS
Windows XP SP3	z/OS V1.9
Our Java applications <ul style="list-style-type: none">▶ saz099.catalog.jar▶ Catalog manager service requester application	CICS Transaction Server V3.2

5.1.2 Definition checklist

The z/OS definitions we used to configure the scenarios are listed in Table 5-2.

Table 5-2 Definition checklist

Value	CICS region 1
IP name	wtsnet.itso.ibm.com
TCP/IP port	14306
Job name	A6POC3C3
APPLID	A6POC3C3
TCPIPSERVICE	C3C3
PIPELINE, configured for MTOM	MTOM

5.1.3 The sample application

For our tests we used the sample program described in 2.5, “Catalog manager example application” on page 53. We do not document how to install the sample application itself, because this is explained in detail in *CICS Web Services Guide V3.1*, SC34-6458.

5.1.4 Testing the scenario

We test the scenario using a modified catalog client example application, which runs as a Java application on Windows. In a Command prompt window we enter the command shown in Example 5-1. We run the first test without having customized CICS to support MTOM/XOP optimization.

Example 5-1 Command used to run sample catalog client java application

```
java -cp c:\saz099.catalog.jar  
com.ibm.itso.saz099.catalog.client.CatalogApplication
```

The application presents the window shown in Figure 5-1.

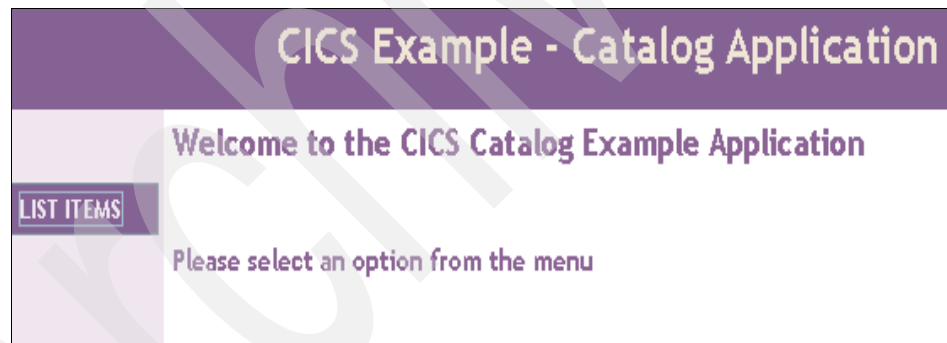


Figure 5-1 Modified catalog application welcome window

Next we click **LIST ITEMS** and the window in Figure 5-2 is presented.

The screenshot shows a web application window titled "CICS Example - Catalog Application". Below the title bar, there is a section titled "Enter Catalog Item Reference Number". On the left side, there is a vertical menu with a button labeled "LIST ITEMS". In the main area, there is a label "Start List From Item Number" followed by a text input field containing the value "0010". At the bottom right, there is a "SUBMIT" button.

Figure 5-2 Catalog application LIST ITEM window

Next we click **SUBMIT** and get the result shown in Figure 5-3.

The screenshot shows the same application window after clicking "SUBMIT". The title bar remains "CICS Example - Catalog Application". The main content area is titled "Item Details - Select Item to Place Order". On the left, the menu now has "LIST ITEMS" and "INQUIRE" buttons. The main area contains a table with the following data:

Item	Description	In Stock	On Order	Cost
10	Ball Pens Blac...	7982	2	002.90
20	Ball Pens Blue...	5	50	002.90
30	Ball Pens Red...	106	0	002.90
40	Ball Pens Gre...	80	0	002.90
50	Pencil with e...	82	0	001.78
60	Highlighters ...	13	40	003.89
70	Laser Paper 2...	100	20	007.44
80	Laser Paper 2...	25	0	033.54
90	Blue Laser Pa...	22	0	005.35
100	Green Laser ...	3	20	005.35
110	IBM Network ...	12	0	169.56
120	Standard Diar...	7	0	025.99
130	Wall Planner:...	3	0	018.85
140	70 Sheet Har...	84	0	005.89
150	Sticky Note...	26	45	005.35

At the bottom of the table, there is a "SUBMIT" button.

Figure 5-3 Catalog application list item result window

Next we highlight item no. 10 and click **SUBMIT**. On the window shown in Figure 5-4 we enter a value for User Name and Department Name and click **SUBMIT**.

CICS Example - Catalog Application

LIST ITEMS

INQUIRE

Enter Order Details

Item Reference Number	10
Quantity	1
User Name	Tommy
Department Name	ITSO
<div style="border: 1px solid #4b4b8b; background-color: #4b4b8b; color: white; padding: 5px 15px; display: inline-block;">SUBMIT</div>	

Click on image for larger view:




Figure 5-4 Catalog application, placeOrder window

Example 5-2 shows the SOAP response without MTOM/XOP optimization. We see that the image data itself is imbedded in the SOAP response.

Example 5-2 CICS trace of the SOAP message not using MTOM/XOP optimization.

```

<SOAP-ENV:Body>
<CR2010C0operationResponse
xmlns="http://www.CR2010C0.CR2.Response.com">
<ca-request-id></ca-request-id>
<ca-return-code>0</ca-return-code>
<ca-response-message>RETURNED ITEM: REF=0010</ca-response-message>
<ca-inquire-single>
<ca-item-ref-req>10</ca-item-ref-req>
<filler1></filler1><filler2></filler2>
<ca-single-item>
<ca-sngl-item-ref>10</ca-sngl-item-ref>
<ca-sngl-description>Ball Pens Black 24pk</ca-sngl-description>
<ca-sngl-department>10</ca-sngl-department>
<ca-sngl-cost>002.90</ca-sngl-cost>
<in-sngl-stock>7993</in-sngl-stock>
<on-sngl-order>2</on-sngl-order>
</ca-single-item>
<filler3></filler3>
</ca-inquire-single>
<imageData-length>0038768</imageData-length>

```

```
<imageData>R01G0DdhAgP/AvCAAAAAAAAAVQAAqgAA/wAkAAkVQAKqgAk/wBJAABJVQBJ
qgBJ/wBtAABtVQBtqgBt/wCSAACSVCsQqCS/wC2AAC2VQC2qgC2/wDbAADbVQDbqgDb/wD
/AAD/VQD/qgD//yQAACQAVSQaqiQA/yQkACQkVSQkqiQk/yRJACRJVSJRqiRJ/yRtACRtVS
RtqiRt/ySSACSSVSSqiSS/yS2ACS2VSS2qiS2/yTbACTbVSTbqiTb/yT/ACT/VST/qiT//
----- deleted lines -----
</imageData>
</CR2010C00operationResponse>
</SOAP-ENV:Body>
```

5.2 Configuring for MTOM/XOP optimization

Note: We had a lot of work to do to support xop and base64 decoding because we are writing a standalone client. If your client runs in an application server that has a framework for supporting Web services, such as axis, then the xop content is handled automatically for a base64 element.

Attention: In Appendix A, “Additional material” on page 315, we provide a copy of the `saz099.catalog.jar` file created from the foregoing changes.

We also ran the DFHWS2LS for each of the three options (inquire, list, and order) in the CICS example catalog application. Example 5-3 shows the parameters we use to modify the WSBIND file and the WSDL in order to support MTOM/XOP. We run the same job for the `catList` and the `catOrder` Web services.

Example 5-3 DFHWS2LS for CATINQ

```
LANG=COBOL
LOGFILE=/u/cicsrs2/wsbind/catInquire.log
PDSLIB=CICSR2.CHNLCON.COPY
PGMINT=CHANNEL
MAPPING-LEVEL=2.1
CHAR-VARYING=NO
PGMNAME=CATINQ
URI=/catalog/CATINQ
WSDL=/CIWS/C3C3/provider/wsd1/catInquire.wsd1
WSBIND=/CIWS/C3C3/wsbind/catInquire.wsbind
BINDING=catInquire
REQMEM=CATINQ
RESPMEM=CATINP
```

Note: We have highlighted the parameters in bold that have changed for using both the modified application and MTOM/XOP:

- ▶ PGMINT= CHANNEL (indicates that we are using channels and container)
- ▶ MAPPING-LEVEL=2.1
- ▶ CHAR-VARYING=NO

5.2.1 Configuration steps

Here we discuss the steps that we went through in our configuration.

Changing the PIPELINE configuration file

Next we create a PIPELINE configuration file, including support for MTOM/XOP. We name the configuration file MTOM.xml, and it is located in the /CIWS/C3C3/config directory. The configuration file is shown in Example 5-4.

Example 5-4 PIPELINE configuration file with MTOM/XOP support

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
xmlns="http://www.ibm.com/software/htp/cics/pipeline"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
provider
  <cics_mtom_handler>
    <dfhmtom_configuration version="1">
      <mtom_options send_mtom="yes" send_when_no_xop="yes"/>
    </dfhmtom_configuration>
  </cics_mtom_handler>
  <transport>
</transport>
  <service>
    <service_handler_list>
</service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler>
</cics_soap_1.1_handler>
    </terminal_handler>
  </service>
    <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Configuring the PIPELINE definition for MTOM

Next we configure the PIPELINE to support MTOM/XOP for the CICS service provider using the following CEDA command:

```
CEDA DEFINE PIPELINE(MTOM) GROUP(C3C3EXWS)
```

We define the MTOM pipeline as shown in Figure 5-5.

```

OVERTYPE TO MODIFY                                     CICS RELEASE = 0650
CEDA DEFine Pipeline(MTOM )
Pipeline       : MTOM
Group         : C3C3EXWS
Description    :
Status        : Enabled                               Enabled | Disabled
Respwait     : Deft                                 Default | 0-9999
Configfile    : /CIWS/C3C3/config/MTOM.xml
(Mixed Case)  :
:
:
:
SHeLF        : /CIWS/C3C3/shelf
(Mixed Case)  :
:
:
:
WsdIr        : /CIWS/C3C3/wsbind/
(Mixed Case)  :
:
:
:
SYSID=C3C3 APPLID=A6POC3C3
```

Figure 5-5 CEDA DEFINE PIPELINE

Installing the PIPELINE definition

We then used CEDA to install the PIPELINE definition. When the PIPELINE is installed, CICS scans the wsdir directory, and dynamically creates WEBSERVICE and URIMAP definitions for the wsbind files found.

Configuring the client to support MTOM/XOP

Next we discuss the additional configuration changes we made to enable MTOM/XOP to be used.

As stated previously, we used a modified version of the CICS catalog example application that enabled us to display images. However, when this application was originally modified, the images were passed in binary format. We now want to take advantage of MTOM/XOP optimization, therefore we want to change the format to support base64.

We followed these steps to change the client to support MTOM/XOP:

1. Add in base64 support. This came from classes downloaded from:
<http://iharder.sourceforge.net/current/java/base64/>
The change was to allow the client to be able to decode base64 data that has been received.
2. Change the client to expect the image being returned to be in base64 format. The data had to be decoded and transformed into the image.
3. Change the client to handle the image being received as either base64 or as an attachment indicated by the data being an `<xop>` element. That should be an `<xop:Include>` element.

If the `xop` element was present, then the data had to be searched to find the matching attachment and the binary data extracted. This data is the image so it can be used directly.

5.2.2 Testing the MTOM/XOP optimization

Next we run the same scenario as documented in “Testing the scenario” on page 145. The optimized SOAP response is shown in Example 5-5. We highlight the `<xop:Include>`, replacing the binary data in Example 5-2 on page 147.

Example 5-5 inquireSingle SOAP response using XTOM/XOP optimization

```
<SOAP-ENV:Body>
<CR2010C0operationResponse
xmlns="http://www.CR2010C0.CR2.Response.com">
<ca-request-id></ca-request-id>
<ca-return-code>0</ca-return-code>
<ca-response-message>RETURNED ITEM: REF=0010</ca-response-message>
<ca-inquire-single><ca-item-ref-req>10</ca-item-ref-req>
<filler1></filler1><filler2></filler2>
<ca-single-item><ca-sngl-item-ref>10</ca-sngl-item-ref>
<ca-sngl-description>Ball Pens Black 24pk</ca-sngl-description>
<ca-sngl-department>10</ca-sngl-department>
<ca-sngl-cost>002.90</ca-sngl-cost>
<in-sngl-stock>7993</in-sngl-stock>
<on-sngl-order>2</on-sngl-order>
</ca-single-item><filler3></filler3>
</ca-inquire-single>
<imageData-length>0038768</imageData-length>
<imageData>
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
href="cid:8cebd22c-ed32c173-8d136412-b8ca9300%40cicsts"/>
</imageData>
</CR2010C0operationResponse>
</SOAP-ENV:Body>
```

SOA governance and WSRR

In this chapter we provide an overview of SOA governance, why it is important, and where you can find out more about the subject. Then we discuss the facilities related to governance provided by the WebSphere Service Registry and Repository (WSRR) product. Finally we show how CICS can be involved in the management of Web Services by the use of CICS SupportPac CA1N and WebSphere Service Registry and Repository.

For a more in-depth discussion on SOA, governance, and the features provided by WebSphere Service Registry and Repository, see the *WebSphere Service Registry and Repository Handbook*, SG24-7386.

6.1 SOA governance

SOA is a compelling technique for developing software applications that best align with business models. However, SOA increases the level of cooperation and coordination required between business and information technology (IT), as well as among IT departments and teams. This cooperation and coordination is provided by SOA governance, which covers the tasks and processes for specifying and managing how services and SOA applications are supported.

6.1.1 What is SOA governance?

In general, governance means establishing and enforcing how a group agrees to work together. Specifically, there are two aspects to governance:

- ▶ Establishing chains of responsibility, authority, and communication to empower people, determining who has the rights to make what decisions.
- ▶ Establishing measurement, policy, and control mechanisms to enable people to carry out their roles and responsibilities.

Governance is distinct from management in the following ways:

- ▶ Governance determines who has the authority and responsibility for making the decisions.
- ▶ Management is the process of making and implementing the decisions.

To put it another way, governance says what should be done, while management makes sure that it gets done.

A more specific form of governance is IT governance, which:

- ▶ Establishes decision-making rights associated with IT.
- ▶ Establishes mechanisms and policies used to measure and control the way IT decisions are made and carried out.

That is, IT governance is about who is responsible for what in an IT department and how the department knows that those responsibilities are being performed.

SOA adds the following unique aspects to governance:

- ▶ It acts as an extension of IT governance that focuses on the lifecycle of services to ensure the business value of SOA.
- ▶ It determines who should monitor, define, and authorize changes to existing services within an enterprise.

Governance becomes more important in SOA than in general IT. In SOA, service consumers and service providers run in different processes, are developed and managed by different departments, and require a lot of coordination to work together successfully. For SOA to succeed, multiple applications have to share common services, which means that they have to coordinate on making those services common and reusable. These are governance issues, and they are much more complex than in the days of monolithic applications or even in the days of reusable code and components.

As companies use SOA to better align IT with the business, companies can ideally use SOA governance to improve overall IT governance. Employing SOA governance is key if companies are to realize the benefits of SOA. For SOA to be successful, SOA business and technical governance is not optional, it is required.

6.1.2 SOA governance in practice

In practice, SOA governance guides the development of reusable services, establishing how services must be designed and developed, and how those services change over time. It establishes agreements between the providers of services and the consumers of those services, telling the consumers what they can expect and the providers what they are obligated to provide.

SOA governance does not design the services, but guides how the services must be designed. It helps answer many thorny questions related to SOA: What services are available? Who can use them? How reliable are they? How long must they be supported? Can you depend on them to not change? What if you want them to change, for example, to fix a bug? Or to add a new feature? What if two consumers want the same service to work differently? Just because you decide to expose a service, does it mean that you are obligated to support it forever? If you decide to consume a service, can you be confident that it is not shut down tomorrow?

SOA governance builds on existing IT governance techniques and practices. A key aspect of IT governance when using object-oriented technologies such as Java 2 Platform, Enterprise Edition (J2EE) is code reuse. Code reuse also illustrates the difficulties of IT governance. Everyone thinks that reusable assets are good, but they are difficult to make work in practice: Who is going to pay to develop them? Do development teams actually strive to reuse them? Can everyone really agree on a single set of behavior for a reusable asset, or does everyone have their own customized version that is not really being reused after all? SOA and services make these governance issues even more important and thus, their consequences even more significant.

Governance is more of a political problem than a technological or business one. Technology focuses on matching interfaces and invocation protocols. Business focuses on functionality for serving customers. Technology and business focus on requirements. While governance gets involved in those aspects, it focuses more on ensuring that everyone is working together and that separate efforts are not contradicting each other. Governance does not determine what the results of decisions are, but what decisions must be made and who makes them.

The two parties, the consumers and the providers, have to agree on how they are going to work together. Much of this understanding can be captured in a service-level agreement (SLA), measurable goals that a service provider agrees to meet and that a service consumer agrees to live with. This agreement is like a contract between the parties, and can, in fact, be a legal contract. At the very least, the SLA articulates what the provider must do and what the consumer can expect.

SOA governance is enacted by a center of excellence (COE), a board of knowledgeable SOA practitioners who establish and supervise policies to help ensure an enterprise's success with SOA. The COE establishes policies for identification and development of services, establishment of SLAs, management of registries, and other efforts that provide effective governance. COE members then put those policies into practice, mentoring and assisting teams with developing services and composite applications.

After the governance COE works out the policies, technology can be used to manage those policies. Technology does not define an SLA, but it can be used to enforce and measure compliance. For example, technology can limit which consumers can invoke a service and when they can do so. It can warn a consumer that the service has been deprecated. It can measure the service's availability and response time.

A good place for the technology to enforce governance policies is through a combination of an enterprise service bus (ESB) and a service registry. A service can be exposed so that only certain ESBs can invoke it. Then the ESB/registry combination can control the consumers' access, monitor and meter usage, measure SLA compliance, and so on. This way, the services focus on providing the business functionality, and the ESB/registry focuses on aspects of governance.

6.1.3 Aspects of SOA governance

SOA governance is not just a single set of practices, but many sets of practices coordinated together. The sections that follow provide a brief overview of the various aspects of SOA governance.

Service definition

The most fundamental aspect of SOA governance is overseeing the creation of services. Services must be identified, their functionality described, their behavior scoped, and their interfaces designed. The governance COE might not perform these tasks, but it makes sure that the tasks are being performed. The COE coordinates the teams that are creating and requiring services, to make sure that requirements are being met and to avoid duplicate effort.

Often, it is not obvious what should be a service. The function should match a set of repeatable business tasks. The service's boundaries should encapsulate a reusable, context-free capability. The interface should expose what the service does, but hide how the service is implemented and allow for the implementation to change or for alternative implementations. When services are designed from scratch, they can be designed to model the business; when they wrap existing function, it can be more difficult to create and implement a good business interface.

An interesting example of the potential difficulties in defining service boundaries is where to set transactional boundaries. A service usually runs in its own transaction, making sure that its functionality either works completely or is rolled back entirely. However, a service coordinator might want to invoke multiple services in a single transaction (ideally through a specified interaction like WS-AtomicTransactions). This task requires the service interface to expose its transaction support so that it can participate in the caller's transaction. But such exposure requires trust in the caller and can be risky for the provider. For example, the provider can lock resources to perform the service, but if the caller never finishes the transaction (it fails to commit or roll back), the provider can have difficulty cleanly releasing the resource locks. As this scenario shows, the scope of a service and who has control is sometimes no easy decision.

Service deployment lifecycle

Services do not come into being instantaneously and then exist forever. Like any software, they must be planned, designed, implemented, deployed, maintained, and ultimately, decommissioned. The application lifecycle can be public and affect many parts of an organization, but a service's lifecycle can have even greater impact because multiple applications can depend on a single service.

The lifecycle of services becomes most evident when you consider the use of a registry. When should a new service be added to the registry? Are all services in a registry necessarily available and ready for use? Should a decommissioned service be removed from the registry?

While there is no one-size-fits-all lifecycle that is appropriate for all services and all organizations, a typical service development lifecycle has five main stages:

1. Plan

A new service that is identified and is being designed, but has not yet been implemented or is still being implemented.

2. Test

After being implemented, a service must be tested. Some testing might have to be performed in production systems, which use the service as though it were active.

3. Active

This is the stage for a service available for use and what we typically think of as a service. It is a service, it is available, it really runs and really works, and it has not been decommissioned yet.

4. Deprecate

This stage describes a service that is still active, but not for much longer. It is a warning for consumers to stop using the service.

5. Sunset

This is the final stage of a service, one that is no longer being provided. Registries might want to keep a record of services that had been active, but are no longer available. This stage is inevitable, and yet frequently is not planned for by providers or consumers.

Sunsetting effectively turns the service version off, and the sunset date should be planned and announced ahead of time. A service should be deprecated within a suitable amount of time before it is *sunsetting*, to programmatically warn consumers so that they can plan accordingly. The schedule for deprecation and sunsetting should be specified in the SLA.

One stage that might appear to be missing from this list is “maintenance.” Maintenance occurs while a service is in the active state; it can move the service back into test to reconfirm proper functionality, although this can be a problem for existing users depending on an active service provider.

Maintenance occurs in services much less than you might expect; maintenance of a service often involves not changing the existing service, but producing a new service version.

Service versioning

No sooner than a service is made available, the users of those services start requiring changes. Bugs have to be fixed, new functionality added, interfaces redesigned, and unnecessary functionality removed. The service reflects the business, so as the business changes, the service has to change accordingly.

With existing users of the service, however, changes have to be made judiciously so as not to disrupt their successful operation. At the same time, the requirements of existing users for stability cannot be allowed to impede those of users desiring additional functionality.

Service versioning meets these contradictory goals. It enables users satisfied with an existing service to continue using it unchanged, yet allows the service to evolve to meet the new requirements of users. The current service interface and behavior is preserved as one version, while the newer service is introduced as another version. Version compatibility can enable a consumer expecting one version to invoke a different but compatible version.

While versioning helps solve these problems, it also introduces new ones, such as the necessity to migrate.

Service migration

Even with service versioning, a consumer cannot depend on a service, or more specifically, a desired version of that service, to be available and supported forever. Eventually, the provider of a service is bound to stop providing it. Version compatibility can help delay this “day of reckoning” but does not eliminate it. Versioning does not obsolete the service development lifecycle, but it enables the lifecycle to play out over successive generations.

When a consumer starts using a service, it is creating a dependency on that service, a dependency that has to be managed. A management technique is for planned, periodic migration to newer versions of the service. This approach also enables the consumer to take advantage of additional features added to the service.

However, even in enterprises with the best governance, service providers cannot depend on consumer migration alone. For a variety of reasons, for example existing code, manpower, budget, priorities, some consumers might not migrate in a timely fashion. Does that mean the provider must support the service version forever? Can the provider simply disable the service version one day after everyone should have already migrated?

Neither of those extremes is desirable. A good compromise is a planned deprecation and sunset schedule for every service version, as described in “Service deployment lifecycle” on page 157.

Service registries

How do service providers make their services available and known? How do service consumers locate the services they want to invoke? These are the responsibilities of a service registry. It acts as a listing of the services available and the addresses for invoking them.

The service registry also helps coordinate versions of a service. Consumers and providers can specify which version they require or have, and the registry then makes sure only to enumerate the providers of the version desired by the consumer. The registry can manage version compatibility, tracking compatibility between versions, and enumerating the providers of a consumer's desired version or compatible versions. The registry can also support service states, such as test and deprecated, and only make services with these states available to consumers that want them.

When a consumer starts using a service, a dependency on that service is created. While each consumer clearly knows which services it depends on, globally throughout an enterprise these dependencies can be difficult to detect, much less manage. Not only can a registry list services and providers, but it can also track dependencies between consumers and services. This tracking can help answer the age-old question: Who is using this service? A registry aware of dependencies can then notify consumers of changes in providers, such as when a service becoming deprecated.

Service message model

In a service invocation, the consumer and provider must agree on the message formats. When separate development teams are designing the two parts, they can easily have difficulty finding agreement on common message formats. Multiply that by dozens of applications using a typical service and a typical application using dozens of services, and you can see how simply negotiating message formats can become a full-time task.

A common approach for avoiding message format chaos is to use a canonical data model. A canonical data model is a common set of data formats that is independent of any one application and shared by all applications. In this way, applications do not have to agree on message formats, they can simply agree to use existing canonical data formats. A canonical data model addresses the format of the data in the message, so you still require agreement around the rest of the message format, for example header fields, what data the message payload contains, and how that data is arranged, but the canonical data model goes a long way toward reaching agreement.

A central governance board can act as a neutral party to develop a canonical data model. As part of surveying the applications and designing the services, it can also design common data formats to be used in the service invocations.

Service monitoring

If a service provider stops working, how do you know? Do you wait until the applications that use those services stop working and the people that use them start complaining?

A composite application, one that combines multiple services, is only as reliable as the services it depends on. Because multiple composite applications can share a service, a single service failure can affect many applications. SLAs must be defined to describe the reliability and performance consumers can depend on. Service providers must be monitored to ensure meeting their defined SLAs.

A related issue is problem determination. When a composite application stops working, why is that? It might be that the application head, the UI that the users interface with, has stopped running. But it can also be that the head is running fine, but some of the services it uses, or some of the services that those services use, are not running properly. Thus it is important to monitor not just how each application is running, but also how each service (as a collection of providers) and individual providers are also running. Correlation of events between services in a single business transaction is critical.

Such monitoring can help detect and prevent problems before they occur. It can detect load imbalances and outages, providing warning before they become critical, and can even attempt to correct problems automatically. It can measure usage over time to help predict services that are becoming more popular so that they can run with increased capacity.

Service ownership

When multiple composite applications use a service, who is responsible for that service? Is that person or organization responsible for all of them? One of them; if so, which one? Do others think they own the service? Welcome to the ambiguous world of service ownership.

Any shared resource is difficult to acquire and care for, whether it is a neighborhood park, a reusable Java framework, or a service provider. Yet a necessary pooled resource provides value beyond any participant's cost: Think of a public road system.

Often an enterprise organizes its staff reporting structure and finances around business operations. To the extent that an SOA organizes the enterprise's IT around those same operations, the department responsible for certain operations can also be responsible for the development and run time of the IT for those operations. That department owns those services. Yet the services and composite applications in an SOA often do not follow an enterprise's strict hierarchical reporting and financial structure, creating gaps and overlap in IT responsibilities.

A related issue is user roles. Because a focus of SOA is to align IT and business, and another focus is enterprise reuse, many different people in an organization have a say in what the services must be, how they must work, and how they must be used. These roles include business analyst, enterprise architect, software architect, software developer, and IT administrator. All of these roles have a stake in making sure that the services serve the enterprise requirements and work correctly.

An SOA should reflect its business. Usually this means changing the SOA to fit the business, but in cases like this, it might be necessary to change the business to match the SOA. When this is not possible, increased levels of cooperation are required between multiple departments to share the burden of developing common services. This cooperation can be achieved by a cross-organizational standing committee that, in effect, owns the services and manages them.

Service testing

The service deployment lifecycle includes the test stage, during which the team confirms that a service works properly before activating it. If a service provider is tested and shown to work correctly, does the consumer have to retest it as well? Are all providers of a service tested with the same rigor? If a service changes, does it have to be retested?

SOA increases the opportunity to test functionality in isolation and increases the expectation that it works as intended. However, SOA also introduces the opportunity to retest the same functionality repeatedly by each new consumer who doesn't necessarily trust that the services it uses are consistently working properly. Meanwhile, because composite applications share services, a single buggy service can adversely affect a range of seemingly unrelated applications, magnifying the consequences of those programming mistakes.

To leverage the reuse benefits of SOA, service consumers and providers have to agree on an adequate level of testing of the providers and have to ensure that the testing is performed as agreed. Then a service consumer only has to test its own functionality and its connections to the service, and can assume that the service works as advertised.

Service security

Should anyone be allowed to invoke any service? Should a service with a range of users enable all users to access all data? Does the data exchanged between service consumers and providers have to be protected? Does a service have to be as secure as the requirements of its most paranoid users or as those of its most lackadaisical users?

Security is a difficult but necessary proposition for any application. Functionality has to be limited to authorized users and data must be protected from interception. By providing more access points to functionality (that is, services), SOA has the potential to greatly increase vulnerability in composite applications.

SOA creates services that are easily reusable, even by consumers who ought not to reuse them. Even among authorized users, not all users should have access to all data the service has access to. For example, a service for accessing bank accounts should only make a particular user's accounts available, even though the code also has access to other accounts for other users. Some consumers of a service have greater requirements than other consumers of the same service for data confidentiality, integrity, and *nonrepudiation*.

Service invocation technologies must be able to provide all of these security capabilities. Access to services has to be controlled and limited to authorized consumers. User identity must be propagated into services and used to authorize data access. Qualities of data protection have to be represented as policies within ranges. This enables consumers to express minimal levels of protection and maximum capabilities and to be matched with appropriate providers who might, in fact, include additional protections.

For more information, go to the following Web sites:

<http://www.ibm.com/software/solutions/soa/gov/>

<http://www.ibm.com/developerworks/ibm/library/ar-servgov/>

http://www.ibm.com/developerworks/rational/downloads/07/rmc_v7.2/soa_governance/index.html

<http://www.ibm.com/developerworks/wikis/display/woolf/SOA+Governance>

6.2 WebSphere Service Registry and Repository

IBM WebSphere Service Registry and Repository (WSRR) is a tool that enables better management and governance of your services. Through its registry and repository capabilities and its integration with IBM SOA Foundation, WebSphere Service Registry and Repository is an essential foundational component of a service-oriented architecture (SOA) implementation.

WSRR enables you to store, access, and manage information about services and service interaction endpoint descriptions (referred to as service *metadata*) in an SOA. This information is used to select, invoke, govern, and reuse services as part of a successful SOA.

This service information includes traditional Web services that implement Web Services Description Language (WSDL) interfaces with SOAP/HTTP bindings as well as a broad range of SOA services that can be described using WSDL, XML Schema Definition (XSD) and policy decorations, but might use a range of protocols and be implemented according to a variety of programming models.

Note: For more information about the IBM SOA programming model and how it relates to the notion of service, see the *Introduction to the IBM SOA Programming Model* at:

<http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel/>

You can use WebSphere Service Registry and Repository to store information about services in your systems, or in other organizations' systems, that you already use, that you plan to use, or that you want to be aware of. For example, an application can check with WSRR just before it invokes a service to locate the most appropriate service that satisfies its functional and performance requirements. This capability helps make an SOA deployment more dynamic and more adaptable to changing business conditions.

WebSphere Service Registry and Repository includes:

- ▶ A *service registry* that contains information about services, such as the service interfaces, its operations, and parameters
- ▶ A *metadata repository* that has the robust framework and extensibility to suit the diverse nature of service usage

The following section gives more detail on what WSRR is and what its functions are in a service-oriented architecture.

6.2.1 What is WebSphere Service Registry and Repository?

As the integration point for service metadata, WebSphere Service Registry and Repository establishes a central point for finding and managing service metadata acquired from a number of sources, including service application deployments and other service metadata and endpoint registries and repositories, such as Universal Description, Discovery, and Integration (UDDI). It is where service metadata that is scattered across an enterprise is brought together to provide a single, comprehensive description of a service. After this happens, visibility is controlled, versions are managed, proposed changes are analyzed and communicated, usage is monitored, and other parts of the SOA foundation can access service metadata with the confidence that they have found the copy of record.

6.2.2 WebSphere Service Registry and Repository in practice

WebSphere Service Registry and Repository does not manage all service metadata, and it does not manage service metadata across the whole SOA lifecycle. It focuses on a minimalist set of metadata that describes capabilities, requirements, and the semantics of deployed service endpoints. It interacts and federates with other metadata stores that play a role in managing the overall lifecycle of a service.

In summary, WebSphere Service Registry and Repository:

- ▶ Provides awareness of service associations and relationships while encouraging reuse of services to avoid duplication and reduce costs.
- ▶ Enhances connectivity with dynamic service selection and binding at runtime.
- ▶ Enables governance of services throughout the service lifecycle.
- ▶ Ensures interoperability of services with a registry and repository built on open standards. Use other standard registries and repositories to ensure a unified view across a variety of service information sources.
- ▶ Helps institute best practices and enforce policies in SOA deployments.

Registry and repository

In this section we consider the registry and repository capabilities of WSRR.

System of records for service information and metadata

Service metadata artefacts exist across an enterprise in a variety of heterogeneous development and runtime stores that often provide information about a service tailored towards use cases in a particular phase of the SOA life-cycle. Examples include Asset Management systems in the development space or Configuration Management systems in the runtime space.

Service metadata plays a key role in an SOA because it can be used to describe and enrich SOA concepts such as services, providers, consumers or contracts. These elements provide contextual as well as operational information about the overall system and can influence the way it is behaving. An example of such information is the definition of service endpoints to be used in a service interaction, indication of service ownership, or elements of service performance such as the average response time or the error rate for a particular service.

In this context, WSRR mostly focuses on handling the metadata management aspects of operational services and provides the system of record of these metadata artefacts—the place where anybody looking for a catalogue of all services deployed in or used by the enterprise would go first.

To fully support this role of a system of record for service related information and metadata, WSRR exhibits both registry and repository capabilities:

- ▶ It provides *registry* functions supporting publication and retrieval of metadata about services, their capabilities, requirements and semantics of services that enable service consumers to find services or to analyze their relationships.
- ▶ It provides *repository* functions to store, manage, and version service information and metadata. In that respect, WSRR really *owns* part of the service information and metadata and is able to play a key role in the overall SOA governance by governing and controlling the entities it manages.

Metadata and policy driven SOA

Part of the metadata stored and managed by WSRR can constitute policies that constrain and drive different aspects of the SOA. For example, you might want to define security policies and relate them to the services that are defined in WSRR. Another example would be to describe Service Level Agreements (SLA) and bind them to service providers and consumers as part of the overall service information model managed by WSRR.

Such policies would influence and drive the behavior or the overall SOA by constraining or relaxing some of the aspects of this system of services. For example, defining security policies to control access and usage of services and storing them in WSRR would help in the implementation of an overall security model applied to an ecosystem of services. Defining service providers' capabilities and service consumers' expectations in the form of Service Level Agreements and relating these definitions to service interactions would also contribute to the ability to control and manage the overall SOA.

As a system of record for service information and metadata, WebSphere Service Registry and Repository is a natural fit for the definition, the promotion, and possibly the application of service oriented policies ("service oriented" in the sense that these policies apply to services) across the different actors and components of the SOA. In that context WSRR might take care of different aspects:

- ▶ The definition of service-oriented metadata and policies by leveraging a rich and extensible information model
- ▶ The promotion of service-oriented policies and metadata across the multiple actors and components of an SOA by providing extended access capabilities to retrieve the pieces of information
- ▶ The enforcement of some service-oriented policies in relation with the overall governance model of the SOA

Integration point with other SOA components

WebSphere Service Registry and Repository provides extended capabilities to promote service information and metadata across all the components of the architecture.

The type of information exchanged depends on the context of the integration and the stage in the SOA Foundation lifecycle where it takes place.

To enable this integration and the promotion of service information and metadata among a broad community of components, environments, and stages, WSRR provides the following capabilities:

- ▶ Various interfaces and APIs supporting different interaction contexts, protocols and technologies such as EJB, Web Services, JMX™, GUI, and command line
- ▶ Complete security model to control these integration cases and restrict access to the managed resources
- ▶ Auditing mechanism to trace the operations taking place as part of these integration cases

Enabler for SOA governance

For a service-oriented architecture to be successful, it requires an expanded scope of governance centered around the reconciliation of multiple points of view and interest, such as business versus IT and development versus operations, and across different lines of business with different priorities. Coming up with a consistent service process model to meet these reconciled objectives is also key. SOA governance tasks serve the entire SOA lifecycle, insuring service cooperation, service technology compatibility, and proper service investment and reuse.

WebSphere Service Registry and Repository plays a key role in the end-to-end SOA governance. While the scope of SOA governance is definitively broader than what WSRR provides support for, it is important to stress the fact that WSRR must be considered as an enabler for SOA governance. WSRR supports governance of service metadata throughout the lifecycle of a service from its initial publication in the development space to deployment to service management.

The following is a brief summary of the main mechanisms WSRR provides to support SOA governance:

- ▶ WSRR is a registry/repository that allows customers to both store and register standards-based service metadata including WSDL, XSD, and policy. This gives customers the ability to have a copy of record for all service metadata and thus ensure consistency throughout their enterprise.

- ▶ The publication of service metadata through tooling, UI, or API allows customers to socialize their high value or standardized shared services, thus encouraging interoperability and reuse.
- ▶ The ability to query and find service metadata is used at development time through tooling, for reuse of interfaces and schemas as well as static binding of service endpoints. At runtime WSRR service queries through the API allow infrastructures such as the ESB to make dynamic endpoint selection decisions based on metadata in the registry.
- ▶ The capture of service dependencies allows the impact of changes to be assessed. Consumers of services that are impacted by the change can be notified and thus take advantage of improvements and ensure that no degradation in quality of service occurs.
- ▶ WSRR provides a basic mechanism for associating policies to services which allows the infrastructure (for example, ESB, Tivoli) or application code to interpret and enforce the policy. This gives agility because policy is configured in the registry which can be changed quickly (without redeployment) but is still subject to governance.
- ▶ WSRR allows client applications (and other middleware) to extend the metadata that can be represented beyond the out-of-the-box standards-based metadata. By providing user defined classifications, properties, and relationships, WSRR can be extended to support any enterprise service metadata model.
- ▶ WSRR supports validation plug-ins that can be configured to run whenever actions are performed against the registry. This means that service metadata updates can be validated ensuring metadata integrity and JMS events generated to allow external applications to respond to the changes.

6.3 Why interoperate CICS and WebSphere?

From a business viewpoint, interoperation greatly increases the range of commercial possibilities for customers who are using CICS and WebSphere by allowing the tools, applications, and runtimes in each environment to make use of resources held in the other ones.

From the CICS customer viewpoint, the transactional programs and applications that have been developed and maintained over the last 20 to 30 years represent a huge investment in time and money with many millions of lines of code usually being involved. By allowing WebSphere to access these valuable resources, they can be modernized, reused, and better exploited in modern, flexible SOA architectures.

There are two main scenarios where CICS and WebSphere can interoperate:

- ▶ **Application development:** Using tools such as Rational Developer for System z to develop new CICS applications and to modernize existing CICS applications. In addition, using the CICS SupportPac CA1N tools to reverse-engineer CICS applications from files held in WSRR.
- ▶ **Application runtime:** Enabling applications in CICS and WebSphere to invoke each other as service providers and service requesters within an SOA implementation.

Application development

The Rational product portfolio contains an important tool for System z application development called Rational Developer for System z.

Rational Developer for System z is an integrated application development workbench that runs in Eclipse. The product is specifically designed for developing System z applications.

When a developer uses Rational Developer for System z to work with a CICS application, the tool must be able to locate the resources (programs) that make up that application. This is possible because an Eclipse plug-in provides an interface that allows Rational Developer for System z to access service metadata held by WSRR in the form of WSDL files.

On the CICS side of the development fence, the CA1N SupportPac tools included with CICS Transaction Server enable developers to retrieve application service metadata files (WSDL files) from WSRR, bring them into the CICS z/OS environment, and “reverse engineer” them into new CICS High Level Language (HLL) applications.

Application runtime

In an SOA implementation that involves CICS and WebSphere, CICS applications must be able to invoke WebSphere applications as service providers (for example, applications running in WebSphere Application Server). WebSphere applications must be able to invoke CICS applications as service providers.

WSRR plays a key role in an SOA runtime environment because it holds the service metadata information (WSDL files) that enable CICS and WebSphere applications to discover and invoke (call) each other.

This is possible because WSDL files derived from CICS application HLL structures are published automatically from the CICS z/OS environment to WSRR. These files can then be accessed by other Web applications that want to invoke their associated applications.

6.4 CICS SupportPac CA1N

This SupportPac provides the tools to streamline the job of registering, discovering, and governing the Web services resources that are generated from CICS applications. Together, these tools provide additional support for organizations intending to move to a Service-Oriented Architecture (SOA). The tools also provide additional evidence of the IBM ongoing commitment to SOA as the environment for conducting today's business operations.

The new tools enable you to automatically publish the Web Service Description Language (WSDL) documents that are derived from CICS application interfaces, from the CICS z/OS environment to the IBM WebSphere Service Registry and Repository (WSRR), where other Web applications can access them. The tools also enable you to retrieve WSDL documents from WSRR into the z/OS environment where they can be modified and "reverse engineered" into new CICS application high-level language structures.

The SupportPac includes:

- ▶ DFHWS2SR: A z/OS batch utility to publish a WSDL document to WSRR.
- ▶ DFHSR2WS: A z/OS batch utility to retrieve a WSDL document from WSRR.

DFHWS2SR provides the ability to publish a WSDL document to WSRR. This can be done in a stand-alone fashion for WSDL documents that have already been created for CICS hosted Web services. Typically though, this would be used as part of a batch job to run the CICS Web Services Assistant DFHLS2WS. DFHLS2WS would have been used to generate a WSDL document and WSBIND file from a CICS language structure. DFHWS2SR could then immediately be called to publish the generated WSDL document into WSRR and make it available for Web clients to find and use.

DFHSR2WS provides the ability to retrieve WSDL documents from WSRR. Typically this would be used as part of a batch job to run the CICS Web Services Assistant DFHWS2LS. DFHSR2WS would run first to retrieve the specified WSDL file (and possibly other associated schema documents). DFHWS2LS would then be run to generate the language structure required for use in a CICS application.

There have been a couple of versions of this SupportPac. It is recommended that you use the latest version of the SupportPac. Currently that is version 2.1. This version has a few important changes from the previous version:

- ▶ It has been upgraded to a category 3 SupportPac. This means that it is now fully supported by the normal CICS support process, provided that the release of CICS it is being used with is still supported.

- ▶ The runtime part has been removed. Only batch utilities are now provided.
- ▶ The utility for retrieving a WSDL document from WSRR, DFHSR2WS, has been enhanced to retrieve imported documents stored in WSRR. The document to be retrieved is now identified using name, namespace, and version where previously a unique identifier supplied by WSRR was required.
- ▶ The utility for publishing a WSDL document into WSRR, DFHWS2SR, has been improved to make the setting of user properties more intuitive.

You can obtain the latest version from the CICS SupportPac Web site:

<http://www.ibm.com/support/docview.wss?rs=1083&uid=swg24013629>

6.5 Installation

This section describes the tasks you have to perform to download and install the CICS SupportPac, and the software required to run it.

This section is aimed at experienced CICS system programmers who are responsible for installing and customizing CICS. Knowledge of downloading files from the Internet, uploading files to z/FS, and UNIX commands is assumed. Options for the **cp**, **pax**, and **chmod** commands are detailed in the *z/OS UNIX System Services Command Reference* manual.

6.5.1 Software requirements

To use the SupportPac, the following software products are required at the version stated, or later:

- ▶ IBM SDK for z/OS Java 2 Technology Edition V1.4
- ▶ IBM WebSphere Service Registry and Repository V6

6.5.2 Downloading and installing the CICS SupportPac

The following steps guide you through downloading and installing the SupportPac.

1. From a workstation, download the ca1n.zip file from the IBM CICS SupportPacs Web site to your workstation:
 - Browse to the IBM CICS SupportPacs Web site:
www.ibm.com/software/ts/cics/txppacs
 - Click the link, **View all: By Category**

- Click the link, **CA1N**
 - Click the link, **HTTP**
 - Provided that you agree to the terms and conditions shown, click the link, **I agree**
 - Save the ca1n.zip file onto your workstation.
2. Extract the contents of ca1n.zip to a local directory. The directory should contain the following items:
 - \CA1N\ca1n.pdf: The SupportPac documentation
 - \CA1N\ca1n.pax.Z: The SupportPac code in the z/OS pax archive format
 - \CA1N\licenses: A directory containing the license in various languages
 3. Upload the file \CA1N\ca1n.pax.Z in binary to your z/OS system. For example, from a workstation command shell, enter the following information. Substitute “install-directory” with the directory to be used for the SupportPac files on z/FS, such as /usr/lpp/cicsts/ca1n.

```
ftp mvs.mysite.com (login with your hostname, userid, and password)
ftp> mkdir /install-directory
ftp> cd /install-directory
ftp> bin
ftp> put CA1N\ca1n.pax.Z
ftp> quit
```
 4. Expand the ca1n.pax.Z file. For example, from a z/OS UNIX shell, enter the following commands:

```
cd /install-directory
pax -rvf ca1n.pax.Z
```
 5. Ensure that the appropriate permissions and attributes are set for the files in the “install-directory” and its sub-directories. In particular, the files in “install-directory/bin” must have their execution permission set. For example, from a z/OS UNIX shell, enter the following command:

```
chmod 755 /install-directory/bin/*
```
 6. Copy the sample JCL files to a data set. For example, from a z/OS UNIX shell, enter the following command:

```
cp -T /install-directory/sampleJCL/* "'/'h1q.CA1N.JCL'"
```

Substitute h1q.CA1N.JCL for the data set where you want to store the SupportPac JCL files.

Important: This command requires the target dataset to already exist.

The following directories and dataset libraries should now exist:

```
/install-directory/ - high level installation
/install-directory/bin - z/OS UNIX shell scripts
/install-directory/classes - Java class libraries
/install-directory/sampleJCL - sample JCL files
hlq.CA1N.JCL() - JCL to launch the utilities and sample JCL library
```

6.6 Publishing WSDL files from z/OS batch to WSRR

This section describes how we used the z/OS cataloged procedure DFHWS2SR provided by the SupportPac to publish a Web Services Description Language document stored on z/FS to WSRR.

We decided to publish the placeOrder.wsdl file which is part of the CICS catalog manager sample. This file is found in the z/FS in `$(CICS_HOME)/samples/webservices/wsdl` directory.

It is possible to call DFHLS2WS followed by DFHWS2SR within the same z/OS batch JCL to keep the WSBIND, WSDL, and the WSDL stored in WSRR in step with the copybooks. Figure 6-1 illustrates the components involved in this scenario. We did NOT do this in our tests. We only ran DFHWS2SR.

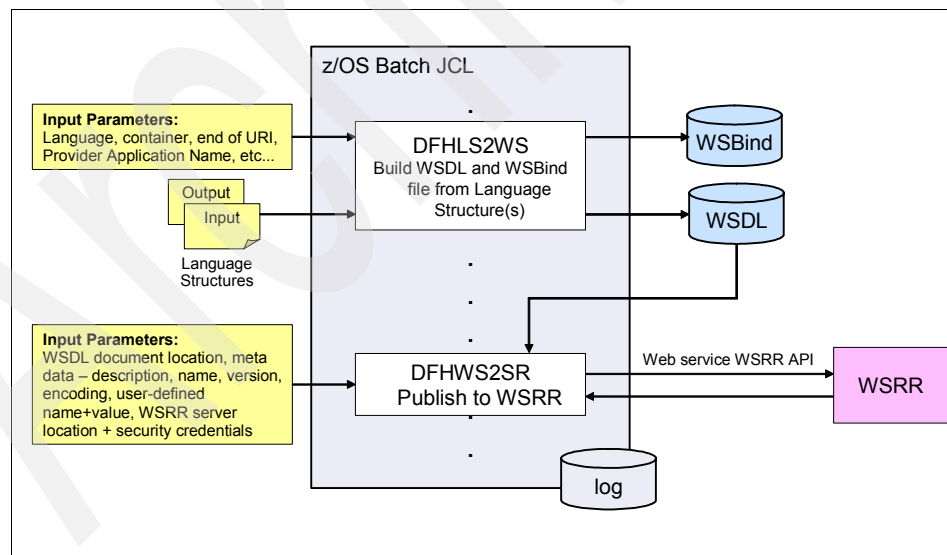


Figure 6-1 Publishing WSDL files from z/OS batch to WSRR

The userid under which the DFHWS2SR procedure ran required authority to:

- ▶ Read files from the SupportPac bin directory (WORKDIR symbolic parameter)
- ▶ Read files from the Java product directories (JAVADIR symbolic parameter)
- ▶ Read the WSDL file to be published (LOCATION parameter)
- ▶ Write access to the temporary directory (TMPDIR symbolic parameter)
- ▶ Write access the log file (LOGFILE parameter)
- ▶ Access to TCP/IP services to send Web services requests to the WSRR server (HOSTPORT parameter)

We did not use SSL, but if connectivity to WSRR is required to be via HTTPS, review 6.9, “Security considerations” on page 193 for details on how to enable it. When SSL is used, the userid running DFHWS2SR also requires access to the KEYSTORE and TRUSTSTORE files.

6.6.1 Job control statements for DFHWS2SR

The following parameters are used for DFHWS2SR:

- ▶ JOB
Initiates the job.
- ▶ EXEC
Specifies the procedure name (DFHWS2SR).
DFHWS2SR requires sufficient storage to run a Java virtual machine (JVM). You are advised to specify REGION=0M on the EXEC statement.
- ▶ INPUT.SYSUT1 DD
Specifies the input. The input parameters are usually specified in the input stream.
However, they can be defined in a data set, or in a member of a partitioned dataset.

6.6.2 Parameters for DFHWS2SR

In this section we describe the parameters used for DFHWS2SR.

Symbolic parameters

These symbolic parameters are defined in cataloged procedure DFHWS2SR:

- ▶ JAVADIR=path
Specifies the name of the Java directory that is used by DFHWS2SR. The value of this parameter is appended to /usr/lpp/ giving a complete path name of /usr/lpp/path.
- ▶ TMPDIR=tmpdir
Specifies the location of a directory in z/FS that DFHWS2SR uses as a temporary work space. The user ID under which the job runs must have read and write permission to this directory.

The default value is /tmp
- ▶ TMPFILE=tmpprefix
Specifies a prefix that DFHWS2SR uses to construct the names of the temporary workspace files.

The default value is WS2SR
- ▶ WORKDIR=path
Specifies the location of the directory in z/FS in which the SupportPac was installed. The user ID under which the job runs must have read permission to this directory.

Restriction: Keep the values for the foregoing parameters as short as possible. They get used as part of a command string for a BPXBATCH command and that is very limited in size. If these parameter values are too long, then the job fails to run.

The temporary work space

DFHWS2SR creates the following three temporary files during execution:

```
tmpdir/tmpprefix.in  
tmpdir/tmpprefix.out  
tmpdir/tmpprefix.err
```

Where:

tmpdir is the value specified in the TMPDIR parameter.
tmpprefix is the value specified in the TMPFILE parameter.

Default names for the files (when TMPDIR and TMPFILE are not specified):

```
/tmp/WS2SR.in  
/tmp/WS2SR.out  
/tmp/WS2SR.err
```

Important: DFHWS2SR does not lock access to the generated HFS file names. Therefore, if two or more instances of DFHWS2SR run concurrently, and use the same temporary workspace files, there is nothing to prevent one job overwriting the workspace files while another job is using them. This can lead to unpredictable failures.

Therefore, you are advised to devise a naming convention and operating procedures to avoid this situation. For example, you can use the system symbolic parameter SYSUID to generate workspace file names that are unique to an individual user.

These temporary files are deleted before the end of the job.

Input Parameters for DFHWS2SR

These are shown in Example 6-1.

Example 6-1 Input parameters to DFHWS2SR

```
.-CCSID=Cp037-.
>>+-----+-----+-----+-----+-----+----->
   '-CCSID=value-' '-DESC=value-' '-ENCODING=value-'
>----HOSTPORT=scheme://-+-domain name+--:port number----->
                               '-IP address--'
>--+-----+-----+-----+-----+-----+----->
   '-KEYSTORE=value-+-----+
                               '-KEYPWD=value-'
>--+-----+-----+-----+-----+-----+----->
   '-NAME=value-' '+-PropertyName=value--+
>--+-----+-----+-----+-----+-----+----->
   '-TRUSTSTORE=value-+-----+
                               '-TRUSTPWD=value-'
                                               .-VERSION=1-----.
>--+-----+-----+-----+-----+-----+-----><
   '-USERNAME=value-+-----+ '-VERSION=value-'
                               '-PASSWORD=value-'
```

Parameter use

Here we explain how to use the parameters:

- ▶ You can specify the input parameters in any order.
- ▶ Each parameter must start on a new line.
- ▶ A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.

- ▶ If a parameter is too long to fit on a single line, use an asterisk (*) character at the end of the line to indicate that the parameter continues on the next line. Everything (including spaces) before the asterisk is considered part of the parameter. For example:

```
LOCATION=/dir/*  
placeOrder.wsdl
```

This is equivalent to:

```
LOCATION=/dir/placeOrder.wsdl
```

- ▶ A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.
- ▶ A # character in the first character position of the line is a comment character. The line is ignored.
- ▶ All keywords and values are case sensitive unless otherwise specified.

Parameter description

Here we describe the parameters:

- ▶ **CCSID=Cp037|value**
The character encoding used to read the WSDL file from z/FS. A list of character encodings that can be specified is available in the “Supported Encodings” topic in the Java manuals. For Java 1.4, the character encodings are available from:
<http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html>
- ▶ **DESC=value**
The description of the WSDL used by WSRR to set the “Description” property.
- ▶ **ENCODING=value**
The character set encoding of the WSDL document used by WSRR to set the “Encoding” property. If ENCODING is not specified, WSRR obtains this value from the `<?xml encoding="encoding_value">` typically found at the beginning of the WSDL document.
- ▶ **HOSTPORT=scheme://{domain name|IP address}:port number**
The URI of the WSRR server. The scheme can be HTTP or HTTPS. The port number defaults to 80 for the scheme HTTP, and 443 for HTTPS.
- ▶ **KEYSTORE=value**
The fully qualified filename of the Keystore file.
- ▶ **KEYPWD=value**
The password to decrypt the keystore file.

- ▶ LOCATION=value
The fully qualified filename of the WSDL to publish.
- ▶ LOGFILE=value
The fully qualified filename into which DFHWS2SR writes its activity log and trace information. DFHWS2SR creates the file (but not the directory structure) if it does not already exist. Normally you do not have to use this file, but it might be helpful to diagnose problems.
- ▶ NAME=value
The name of the WSDL used by WSRR to set the “Name” property. If NAME is not specified, WSRR uses the unqualified filename of the WSDL.
- ▶ PASSWORD=value
The password to access WSRR.
- ▶ PropertyName=value
WSRR allows you to publish metadata with a WSDL document through the use of user-defined properties. Each property is a name and value pair. You can define up to 250 properties using the format *PropertyName=value*, where *PropertyName* is the user-defined name and *value* is the string to set it to; for example:
Host=IBM CICS Transaction Server
Avoid specifying duplicate property names.
- ▶ TRUSTPWD=value
The password to decrypt the truststore file.
- ▶ TRUSTSTORE=value
The fully qualified filename of the truststore file.
- ▶ USERNAME=value
The username to access WSRR, and is used by WSRR to set the “Owner” property.
- ▶ VERSION=1|value
The version number of the WSDL used by WSRR to set the “Version” property.

Completion codes

Table 6-1 shows the possible completion codes for DFHWS2SR.

Table 6-1 Completion codes for DFHWS2SR

Completion Code	Meaning
0	OK. The job completed successfully and only Informational messages have been issued.
4	Input Error. The job did not complete successfully. One or more error messages were issued to SYSOUT whilst validating the input parameters.
8	Input Error. The job did not complete successfully. One or more error messages were issued to SYSOUT whilst validating the input parameters.
12	Error. The job did not complete successfully. One or more error messages were issued to SYSOUT during execution.

6.7 Retrieving WSDL files from WSRR using z/OS batch

This section describes how to use the z/OS cataloged procedure DFHSR2WS provided by the CICS SupportPac to read a WSDL document stored in WSRR and place a copy of it on z/FS.

If you have to create high level language structures (copybooks) from the WSDL document, use the DFHWS2LS procedure provided with CICS TS V3.1 and above.

It is possible to call DFHSR2WS followed by DFHWS2LS within the same z/OS batch JCL to keep the copybooks, WSBind and WSDL on z/FS, in step with the WSDL stored in WSRR. To keep the CICS application that makes use of the generated copybooks in step, would take additional steps. Figure 6-2 illustrates the components involved in this scenario.

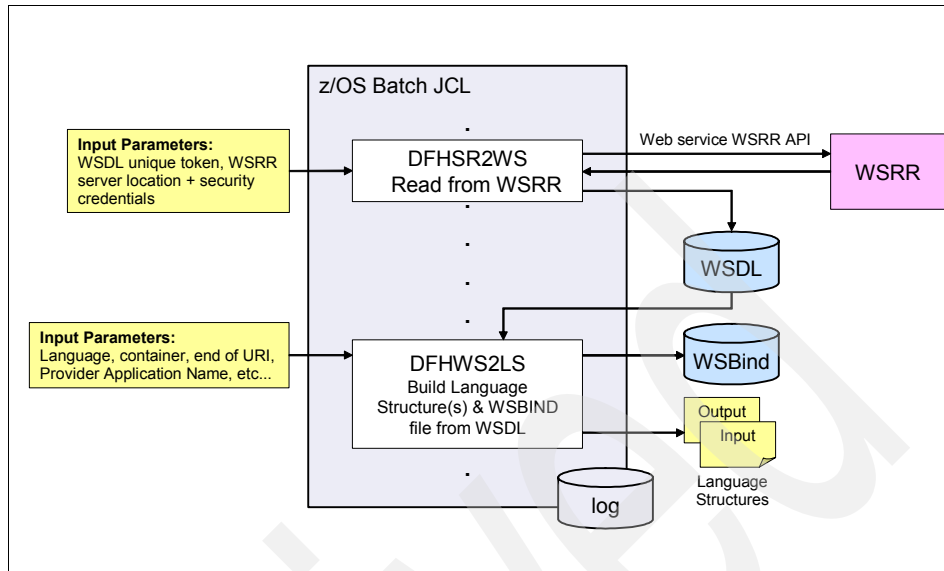


Figure 6-2 Reading WSDL files stored in WSRR from z/OS batch

This section is aimed at experienced CICS system programmers who are responsible for installing and customizing CICS. Knowledge of basic UNIX commands and the Java configuration on your system is required.

The userid under which the DFHWS2SR procedure is run requires authority to:

- ▶ read files from the SupportPac bin directory (WORKDIR symbolic parameter)
- ▶ read files from the Java product directories (JAVADIR symbolic parameter)
- ▶ optionally, read the trust store and key store files (TRUSTSTORE and KEYSTORE parameters)
- ▶ write the WSDL file to z/FS (LOCATION parameter)
- ▶ write access to the temporary directory (TMPDIR symbolic parameter)
- ▶ write access the log file (LOGFILE parameter)
- ▶ access to TCP/IP services to send Web services requests to the WSRR server (HOSTPORT parameter)

If connectivity to WSRR is required to be via HTTPS, review 6.9, “Security considerations” on page 193.

6.7.1 Job control statements for DFHSR2WS

The following job control statements are used for DFHSR2WS:

- ▶ JOB
Initiates the job.
- ▶ EXEC
Specifies the procedure name (DFHSR2WS).
DFHSR2WS requires sufficient storage to run Java virtual machine (JVM).
You are advised to specify REGION=0M on the EXEC statement.
- ▶ INPUT.SYSUT1 DD
Specifies the input. The input parameters are usually specified in the input stream.
However, they can be defined in a data set, or in a member of a partitioned data set.

6.7.2 Parameters for DFHSR2WS

The following parameters are used for DFHSR2WS:

Symbolic parameters

The following symbolic parameters are defined in cataloged procedure DFHSR2WS:

- ▶ JAVADIR=path
Specifies the name of the Java directory that is used by DFHSR2WS. The value of this parameter is appended to /usr/lpp/ giving a complete path name of /usr/lpp/path
- ▶ TMPDIR=tmpdir
Specifies the location of a directory in z/FS that DFHSR2WS uses as a temporary work space. The user ID under which the job runs must have read and write permission to this directory.
The default value is /tmp
- ▶ TMPFILE=tmpprefix
Specifies a prefix that DFHSR2WS uses to construct the names of the temporary workspace files.
The default value is SR2WS

► **WORKDIR=path**

Specifies the location of the directory in z/FS in which the SupportPac was installed. The user ID under which the job runs must have read permission to this directory.

Restriction: Keep the values for the above parameters as short as possible. They get used as part of a command string for a BPXBATCH command and that is very limited in size. If these parameter values are too long, then the job fails to run.

The temporary work space

DFHWS2SR creates the following three temporary files during execution:

```
tmpdir/tmpprefix.in  
tmpdir/tmpprefix.out  
tmpdir/tmpprefix.err
```

Where:

`tmpdir` is the value specified in the `TMPDIR` parameter
`tmpprefix` is the value specified in the `TMPFILE` parameter.

The default names for the files (when `TMPDIR` and `TMPFILE` are not specified), are:

```
/tmp/SR2WS.in  
/tmp/SR2WS.out  
/tmp/SR2WS.err
```

Important: DFHSR2WS does not lock access to the generated HFS file names. Therefore, if two or more instances of DFHSR2WS run concurrently, and use the same temporary workspace files, there is nothing to prevent one job overwriting the workspace files while another job is using them. This can lead to unpredictable failures.

Therefore, to avoid this situation, we recommend that you devise a naming convention and operating procedures. For example, you can use the system symbolic parameter `SYSUID` to generate workspace file names that are unique to an individual user.

These temporary files are deleted before the end of the job.

Input parameters for DFHSR2WS

These are shown in Example 6-2.

Example 6-2 Input parameters for DFHSR2WS

```
.-CCSID=Cp037-.
>>+-----+--HOSTPORT=scheme://-+domain name-+:-port number-->
   '-CCSID=value-' '-IP address--'
>+-----+--LOCATION=value--LOGFILE=value-->
   '-KEYSTORE=value-+-----+
                        '-KEYPWD=value-'
>+-----+-----+----->
   '-TRUSTSTORE=value-+-----+
                        '-TRUSTPWD=value-'
>+-----+-----+--NAME=value-----><
   '-USERNAME=value-+-----+
                        '-PASSWORD=value-'
```

Parameter use

Here we explain how to use the parameters:

- ▶ You can specify the input parameters in any order.
- ▶ Each parameter must start on a new line.
- ▶ A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.
- ▶ If a parameter is too long to fit on a single line, use an asterisk (*) character at the end of the line to indicate that the parameter continues on the next line. Everything (including spaces) before the asterisk is considered part of the parameter.

For example:

```
LOCATION=/dir/*
placeOrder.wsd1
```

This is equivalent to:

```
LOCATION=/dir/placeOrder.wsd1
```

A parameter (and its continuation character, if you use one) must not extend beyond column 72; columns 73 to 80 should contain blanks.

A # character in the first character position of the line is a comment character. The line is ignored.

All keywords and values are case sensitive unless otherwise specified.

Parameter description

Here we describe the parameters:

- ▶ **CCSID=Cp037|value**

The character encoding used to write the WSDL file to z/FS. A list of character encodings that can be specified is available in the “Supported Encodings” topic in the Java manuals. For Java 1.4, the character encodings are available from:

<http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html>
- ▶ **HOSTPORT=scheme://{domain name|IP address};port number**

URI of the WSRR server. The scheme should be HTTP or HTTPS. The port number defaults to 80.
- ▶ **KEYSTORE=value**

Only required if the scheme specified in HOSTPORT is HTTPS. The fully qualified filename of the Keystore file.
- ▶ **KEYPWD=value**

The password to decrypt the keystore file.
- ▶ **LOCATION=value**

Fully qualified filename for the WSDL document read from WSRR. DFHSRWS creates the file (but not the directory structure) if it does not already exist.
- ▶ **LOGFILE=value**

The fully qualified filename into which DFHSR2WS writes its activity log and trace information. DFHSR2WS creates the file (but not the directory structure) if it does not already exist. Normally you do not have to use this file, but it might be helpful to diagnose problems.
- ▶ **NAME=value**

The name of the WSDL document stored in WSRR.
- ▶ **NAMESPACE=value**

Optionally specify the namespace qualification for the WSDL document.
- ▶ **PASSWORD=value**

The password to access WSRR.
- ▶ **TRUSTPWD=value**

The password to decrypt the truststore file.
- ▶ **TRUSTSTORE=value**

The fully qualified filename of the truststore file.

- ▶ **USERNAME=value**
The username to access WSRR.
- ▶ **VERSION=value**
Optionally specify the version of the WSDL document stored in WSRR.

Completion code

Table 6-2 shows the possible completion codes for DFHSR2WS.

Table 6-2 Completion codes for DFHSR2WS

Completion Code	Meaning
0	OK. The job completed successfully and only Informational messages have been issued.
4	Input Error. The job did not complete successfully. One or more error messages were issued to SYSOUT whilst validating the input parameters.
8	Input Error. The job did not complete successfully. One or more error messages were issued to SYSOUT whilst validating the input parameters.
12	Error. The job did not complete successfully. One or more error messages were issued to SYSOUT during execution.

6.8 Setup for testing the SupportPac with WSRR 6.1

This section describes the system setup that was used for testing the CA1N SupportPac with WSRR. We do not go into detail on how to install and configure WebSphere Application Server or WSRR. The following two sections show the results of the tests we did publishing WSDL files in to WSRR and retrieving them back.

6.8.1 Software checklist

For the configurations shown in Figure 6-1 on page 173 and Figure 6-2 on page 180, we used the levels of software shown here in Table 6-3.

Table 6-3 Software used in the WSRR test scenarios

Windows	z/OS
Windows XP SP2	z/OS V1.8
IBM WebSphere Application Server - ND V6.1.0.13	CICS Transaction Server V3.2
IBM WebSphere Service Registry and Repository V6.1.0.3	CICS SupportPac CA1N
Mozilla Firefox 2.0.0.15	

WSRR was configured with its supplied sample configuration. The only other change made was to disable security on the Service Integration Bus within WebSphere Application Server. This was necessary because when WSRR is installed it enables bus security by default and we were not going to use security in these tests.

6.8.2 Running DFHWS2SR

To illustrate the use of DFHWS2SR we performed three different publishes. They all used the same sample placeOrder.wsdl file that is supplied with CICS.

First publish

Our first test was just to do a basic publish of the placeOrder.wsdl file. We used the minimum set of required parameters. We modified the supplied WSDLPUB JCL and Example 6-3 shows the final JCL we submitted.

Example 6-3 Customized version of hlq.CA1N.JCL(WSDLPUB)

```
//WSDLPUB JOB (MYSYS,AUSER),MSGCLASS=H,  
// CLASS=A,NOTIFY=&SYSUID,REGION=0M  
//WSDLPUB JCLLIB ORDER=(CICS.CICSTS.CA1N.JCL)  
// EXEC DFHWS2SR,  
// JAVADIR='java/IBM/J1.4',  
// WORKDIR='/usr/lpp/cicsts/ca1n',  
// TMPDIR='/u/username/tmp',  
// TMPFILE='WS2SR'  
//INPUT.SYSUT1 DD *
```

```
LOGFILE=/u/mpocock/cics650/wsrr/placeOrder.log
LOCATION=/usr/lpp/cicsts/cics650/samples/webservices/wsd1/*
placeOrder.wsd1
NAME=placeOrder
DESC=CICS TS 3.2 Catalog sample - Place Order sample
VERSION=1
ENCODING=EBCDIC-CP-US
HOSTPORT=http://9.143.31.49:9081
*/
```

Example 6-4 shows the contents of SYSOUT after submitting the JCL shown above in Example 6-3. The LOGFILE contains the same information but also shows the full SOAP messages being sent and received.

Example 6-4 DFHWS2SR SYSOUT

```
Reading parameter file /tmp/ws2sr.in
Writing log to '/u/mpocock/cics650/wsrr/placeOrder.log'
Release: CICS TS WSRR SupportPac v2.1 (CA1N)
Build-Level: cicswsrr2
INFORMATIONAL: 'USERNAME' parameter not set.
INFORMATIONAL: 'PASSWORD' parameter not set.
INFORMATIONAL: 'KEYSTORE' parameter not set.
INFORMATIONAL: 'KEYPWD' parameter not set.
INFORMATIONAL: 'TRUSTSTORE' parameter not set.
INFORMATIONAL: 'TRUSTPWD' parameter not set.
INFORMATIONAL: 'CCSID' parameter not set. Cp037 is used.
INFORMATIONAL: User properties (name value pairs) can be provided by
specifying them in the form 'name=value', one pair per line, up to 250.
Starting publish of WSDL document to WSRR
Invoking Web service
Success! WSDL document is now stored in WSRR with unique id
80b6b380-90a4-4450.be52.71840f715264
Program 'DFHWS2SR' completed SUCCESSFULLY.
```

We verified that the publish was indeed successful by using the WSRR console to see what documents were stored in the repository. Figure 6-3 shows what was seen in WSRR.

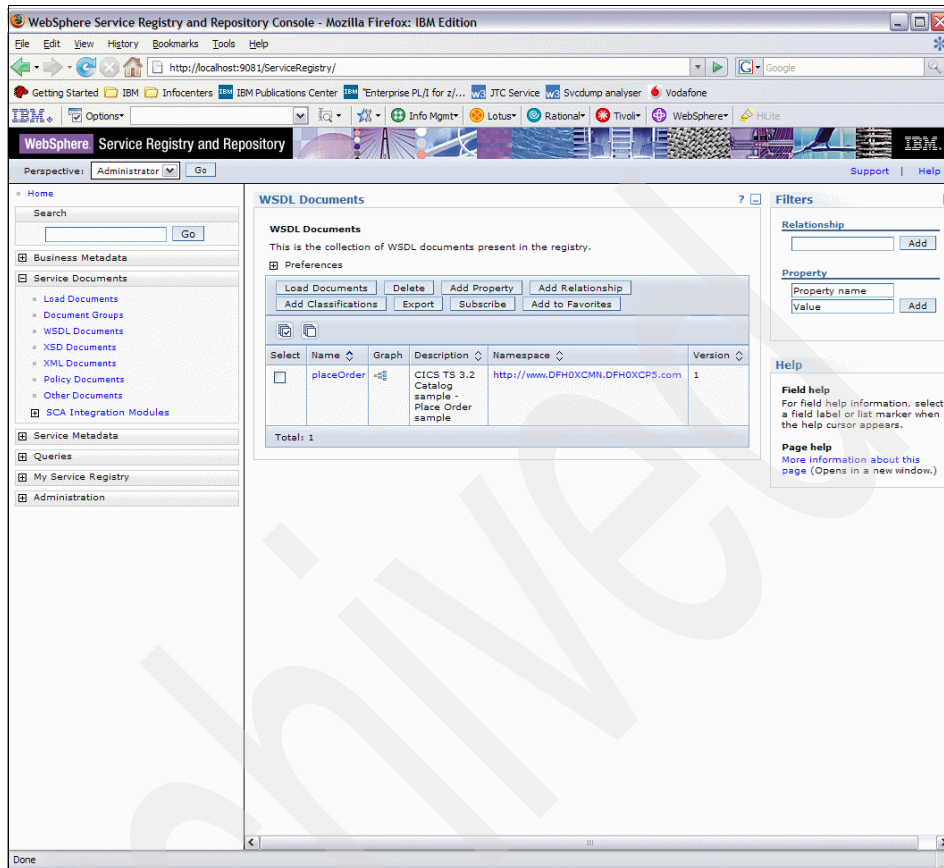


Figure 6-3 Screen capture showing WSRR repository after first publish

Second publish

We then decided to publish a second version of the file in to WSRR. We changed the DESC and VERSION parameters and resubmitted the JCL. Example 6-5 shows the updated set of parameters used.

Example 6-5 Parameters for second publish

```
LOGFILE=/u/mpocock/cics650/wsrr/placeOrder.log
LOCATION=/usr/lpp/cicsts/cics650/samples/webservices/wsd1/*
placeOrder.wsd1
NAME=placeOrder
DESC=CICS TS 3.2 Catalog sample - Place Order sample (2)
VERSION=2
ENCODING=EBCDIC-CP-US
HOSTPORT=http://9.143.31.49:9081
```

As before, we verified in WSRR that the new WSDL document had been stored successfully. This time there were two documents in the repository. Both had the same name and namespace but they had different version numbers. Figure 6-4 shows the view in WSRR.

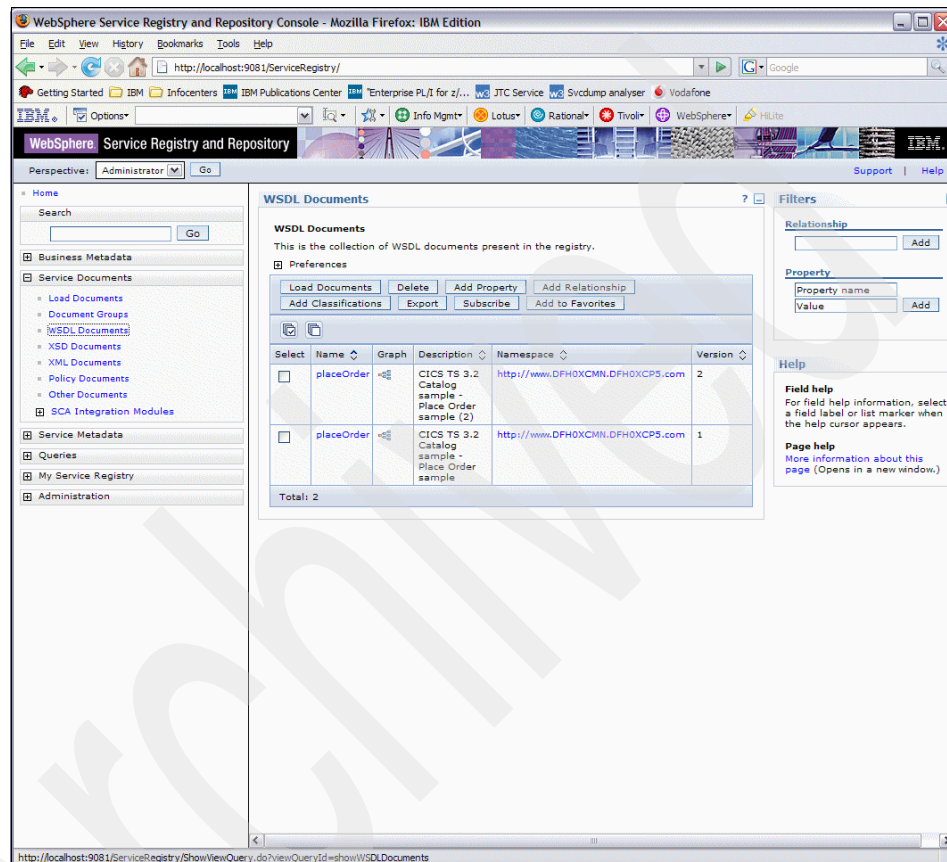


Figure 6-4 Screen capture of WSRR after the second publish

Third publish

To illustrate the use of custom properties, we performed a third publish of the placeOrder.wsdl file. We reused the same JCL, changed the parameters, and submitted it again. Example 6-6 shows the updated parameters we used including our custom property PUBLISHED_BY, which we gave a value of “ITSO Redbooks.”

Example 6-6 Parameters for third publish

```
LOGFILE=/u/mpocock/cics650/wsrr/placeOrder.log
LOCATION=/usr/lpp/cicsts/cics650/samples/webservices/wsd1/*
placeOrder.wsd1
NAME=placeOrder
DESC=CICS TS 3.2 Catalog sample - Place Order sample (3)
VERSION=3
ENCODING=EBCDIC-CP-US
HOSTPORT=http://9.143.31.49:9081
PUBLISHED_BY=ITSO Redbooks
```

We again checked in WSRR to see that the WSDL file was successfully published in the repository. This time we also checked that our custom property had been set. Figure 6-5 shows the properties page for this WSDL document, which includes our PUBLISHED_BY property.

The screenshot displays the WebSphere Service Registry and Repository Console in Mozilla Firefox. The main content area shows the 'Properties' page for the WSDL document 'placeOrder'. The page includes a table of metadata properties with columns for 'Name' and 'Value'. The 'PUBLISHED_BY' property is highlighted in blue, indicating it is the current selection. The table lists 18 properties, including 'name', 'description', 'owner', 'lastModified', 'creationTimestamp', 'lastModifiedBy', 'encoding', 'location', 'contentSize', 'xmlns', 'xmlns_tns', 'xmlns_soap', 'PUBLISHED_BY', 'xmlns_resns', and 'xmlns_reqns'. The 'PUBLISHED_BY' property has the value 'ITSO Redbooks'. The 'Total: 18' is shown at the bottom of the table.

Select	Name	Value
<input type="checkbox"/>	bsrURI	84a63b84-c9b8-487f-97b5-f78cc6f7b50c
<input type="checkbox"/>	name	placeOrder
<input type="checkbox"/>	namespace	http://www.DFHOXCMN.DFHOXCP5.com
<input type="checkbox"/>	version	3
<input type="checkbox"/>	description	CICS TS 3.2 Catalog sample - Place Order sample (3)
<input type="checkbox"/>	owner	UNAUTHENTICATED
<input type="checkbox"/>	lastModified	Friday, July 11, 2008 10:49:52 AM BST
<input type="checkbox"/>	creationTimestamp	Friday, July 11, 2008 10:49:52 AM BST
<input type="checkbox"/>	lastModifiedBy	UNAUTHENTICATED
<input type="checkbox"/>	encoding	EBCDIC-CP-US
<input type="checkbox"/>	location	/usr/lpp/cicsts/cics650/samples/webservices/wsd1/placeOrder.wsd1
<input type="checkbox"/>	contentSize	15,032
<input type="checkbox"/>	xmlns	http://schemas.xmlsoap.org/wsd/
<input type="checkbox"/>	xmlns_tns	http://www.DFHOXCMN.DFHOXCP5.com
<input type="checkbox"/>	xmlns_soap	http://schemas.xmlsoap.org/wsd/soap/
<input type="checkbox"/>	PUBLISHED_BY	ITSO Redbooks
<input type="checkbox"/>	xmlns_resns	http://www.DFHOXCMN.DFHOXCP5.Response.com
<input type="checkbox"/>	xmlns_reqns	http://www.DFHOXCMN.DFHOXCP5.Request.com

Figure 6-5 Properties page for WSDL document

6.8.3 Running DFHSR2WS

To illustrate the use of DFHSR2WS, we performed two retrieves. The first was to retrieve one of the WSDL files we just published. The second was to show the error you get when trying to retrieve a WSDL file that cannot be found in the repository.

First retrieve

Our first test was to retrieve version 2 of the placeOrder.wsdl document. We used the minimum set of required parameters. We modified the supplied WSDLREAD JCL and Example 6-7 shows the final JCL we submitted.

Example 6-7 Customized version of hlq.CA1N.JCL(WSDLREAD)

```
//WSDLREAD JOB (MYSYS,AUSER),MSGCLASS=H,  
// CLASS=A,NOTIFY=&SYSUID,REGION=OM  
//WSDLREAD JCLLIB ORDER=(CICS.CICSTS.CA1N.JCL)  
// EXEC DFHSR2WS,  
// JAVADIR='java/IBM/J1.4',  
// WORKDIR='/usr/lpp/cicsts/ca1n',  
// TMPDIR='/u/username/tmp',  
// TMPFILE='SR2WS'  
//INPUT.SYSUT1 DD *  
LOGFILE=/u/mpocock/cics650/wsrr/placeOrder.log  
LOCATION=/u/mpocock/cics650/wsrr/retrieved/  
NAME=placeOrder  
VERSION=2  
HOSTPORT=http://9.143.31.49:9081  
*/
```

Important: When running DFHSR2WS the value specified for the LOCATION parameter **MUST** be a directory. The filename of the retrieved document is appended to whatever you specify. The directory must also exist.

Example 6-8 shows the output written to SYSOUT after submitting the above JCL. The LOGFILE contains the same information as well as showing the SOAP messages sent to and received from WSRR.

Example 6-8 DFHSR2WS SYSOUT

```
Reading parameter file /tmp/SR2WS.in  
Writing log to '/u/mpocock/cics650/wsrr/placeOrder.log'  
Release: CICS TS WSRR SupportPac v2.1 (CA1N)  
Build-Level: cicswsrr2  
INFORMATIONAL: 'USERNAME' parameter not set.
```

```
INFORMATIONAL: 'PASSWORD' parameter not set.
INFORMATIONAL: 'NAMESPACE' parameter not set.
INFORMATIONAL: 'KEYSTORE' parameter not set.
INFORMATIONAL: 'KEYPWD' parameter not set.
INFORMATIONAL: 'TRUSTSTORE' parameter not set.
INFORMATIONAL: 'TRUSTPWD' parameter not set.
INFORMATIONAL: 'CCSID' parameter not set. Cp037 is used.
Starting read of WSDL document from WSRR
Writing WSDL to /u/mpocock/cics650/wsrr/retrieved/placeOrder.wsdl
Program 'DFHSR2WS' completed SUCCESSFULLY.
```

Second retrieve

In this test we tried to retrieve a WSDL document that did not exist in the WSRR repository. We used the same WSDLREAD JCL with slightly changed parameters as shown in Example 6-9.

Example 6-9 Parameters for second retrieve

```
LOGFILE=/u/mpocock/cics650/wsrr/unknownName.log
LOCATION=/u/mpocock/cics650/wsrr/retrieved/
NAME=unknownName
VERSION=1
HOSTPORT=http://9.143.31.49:9081
```

As expected, the retrieve failed. SYSOUT showed the messages seen in Example 6-10.

Example 6-10 SYSOUT from second retrieve

```
Reading parameter file /tmp/SR2WS.in
Writing log to '/u/mpocock/cics650/wsrr/unknownName.log'
Release: CICS TS WSRR SupportPac v2.1 (CA1N)
Build-Level: cicswsrr2
INFORMATIONAL: 'USERNAME' parameter not set.
INFORMATIONAL: 'PASSWORD' parameter not set.
INFORMATIONAL: 'NAMESPACE' parameter not set.
INFORMATIONAL: 'KEYSTORE' parameter not set.
INFORMATIONAL: 'KEYPWD' parameter not set.
INFORMATIONAL: 'TRUSTSTORE' parameter not set.
INFORMATIONAL: 'TRUSTPWD' parameter not set.
INFORMATIONAL: 'CCSID' parameter not set. Cp037 is used.
Starting read of WSDL document from WSRR
WARNING: Specified WSDL not found in WSRR.
Program 'DFHSR2WS' completed with WARNINGS.
```

6.9 Security considerations

When the HTTPS transport is used, SSL has to be configured between either the z/OS batch utilities or CICS and WSRR.

6.9.1 Configuring SSL between the z/OS batch utilities and WSRR

The z/OS batch utilities require a key store and a trust store to be provided. These can be created using a key configuration program such as **ikeyman**. WebSphere Application Server, which hosts WSRR, provides some sample key stores called `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`, and trust stores called `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`. These can be used as-is for testing the z/OS batch utilities with WSRR by specifying the client key and trust files in the z/OS batch utilities and the server key and trust files in WebSphere. Because the keys in these stores are shipped with WebSphere, they should be replaced.

6.9.2 Example of using self-signed certificates

As an example of using self-signed certificates, you could change the keys by opening the client key file using **ikeyman**, delete the “websphere dummy client” certificate and create a new self-signed certificate. Then extract the certificate and add it into the server's trust file. Next, open the servers' key file, delete the “websphere dummy server” certificate and create a new self-signed certificate, then extract the certificate and add it into the client's trust file. The client and server are then able to use SSL with the new certificates.

Archived



Part 3

Transaction management

In this part of the book, we begin by providing an introduction to Web services atomic transactions. We then outline the steps for enabling WS-Atomic Transaction (WS-AT) support in CICS. Finally, we show several scenarios that demonstrate how you can synchronize WebSphere and CICS updates using the WS-AT standard.

Archived

Introduction to Web services: Atomic transactions

We begin this chapter by describing an example of a classic transaction. We expect most of our readers to be familiar with examples similar to ours. Then we map our classical transaction to a Web services *atomic transaction* as a means of introducing the new terminology that we use in this and succeeding chapters: *coordinator*, *transactional context*, *activation service*, *registration service*, and *completion protocol*. We continue by providing an overview of the flows and message contents prescribed by the three specifications upon which CICS TS V3.2 support for atomic transactions is built:

- ▶ Web Services - Addressing (WS-Addressing or WS-A)
- ▶ Web Services - Coordination (WS-Coordination or WS-C)
- ▶ Web Services - Atomic Transaction (WS-Atomic Transaction or WS-AT)

7.1 Beginner's guide to atomic transactions

We begin by describing an example of a classic transaction. For this discussion we borrow freely from the paper *Tour Web Services Atomic Transaction operations: Beginner's guide to classic transactions, data recovery, and mapping to WS-Atomic Transactions*, which Thomas Freund and Daniel House published on September 2, 2004, on the IBM developerWorks® Web site at:

<http://www-128.ibm.com/developerworks/webservices/library/ws-introwsat>

Not losing money is quite important. Just ask Waldo. Waldo's situation typifies the requirement for a transaction. Waldo uses a Web browser or an Automatic Teller Machine (ATM) to move some money from one account to another account. These accounts can be in different branches of the same financial institution, or they can be in different institutions.

It is never acceptable to Waldo for his money to disappear. Should Waldo ever doubt the safety of his money, he would probably switch financial institutions.

Waldo's money is represented by data in two databases that cooperate to ensure that the data they contain is always in a known and consistent state. That is, these two databases allow actions or tasks between them to be within a common activity or work scope as shown in Figure 7-1. Put yet another way, a single transaction can manipulate data in both databases and *something* guarantees that only one of two possible outcomes occurs: *all* the changes are successfully made or *none* of them is made at all.

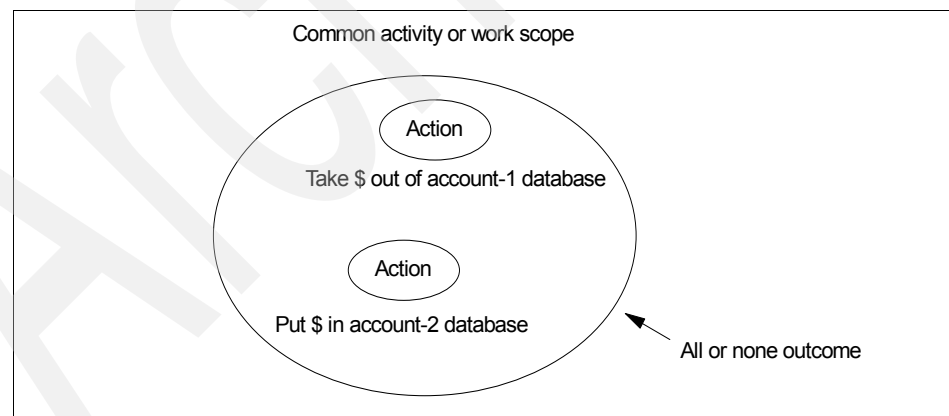


Figure 7-1 Common activity encompasses various recoverable actions

The *something* that guarantees the common outcome of all the actions is a protocol supported by both databases, and some supporting middleware. The protocol that the databases use to keep data (such as Waldo's balances)

coordinated is called *two phase commit*, or simply 2PC. Our example uses a common variation of 2PC called *presumed abort*, where the default behavior in the absence of a successful outcome is to rollback or undo all the actions in the activity.

From a programming perspective, there are different ways to specify that multiple actions should be within the scope of a single transaction. One particularly clear way to specify transactional behavior is shown in Example 7-1. The code is the small piece of logic running somewhere behind the ATM Waldo is using—perhaps in the data center of one of the financial institutions involved.

Example 7-1 Pseudo-code for Waldo's transaction

```
TransferCash(fromAcct, toAcct, amount)
  BeginTransaction
  fromAcct = fromAcct - amount
  toAcct = toAcct + amount
  CommitTransaction
Return
```

7.1.1 What is a classic transaction?

For our simple purposes, a *recoverable* action is anything that modifies protected data. For example, taking money out of one of Waldo's accounts ($\text{fromAcct} = \text{fromAcct} - \text{amount}$) is a recoverable action that can be reversed up to the end of the transaction. A *classic* transaction, then, is just a grouping of recoverable actions, the guaranteed outcome of which is that either all the actions are taken, or none of them is taken (see Figure 7-1).

In Waldo's case, his transaction is composed of two actions: taking money out of one account and putting money into another account. It is all right for both of these actions to occur, and it is even all right if neither of these actions occurs. However, it is never permissible for one action to occur without the other also occurring, which would result in corrupt data and either Waldo's net worth or the bank's assets disappearing or appearing from nowhere. Hence, both actions have to be within a single transaction with a single outcome: Either both actions occur (a *commit* outcome), or neither action occurs (a *rollback* outcome).

Assuming that no errors happen, the code in Example 7-1 shows that a commit outcome is desired. The code could just as easily have specified rollback instead of commit (for when Waldo presses the Cancel key on the ATM), which means to reverse all actions in the transactional work scope (between beginning and end). The transaction monitor, which is the underlying middleware helping the code in Example 7-1 that supports transaction processing, would automatically specify rollback if the program suffered an unhandled exception.

Such an automatic rollback on the part of the transaction monitor is a protection mechanism to make sure that data is not corrupted. For example, even if the ATM application fails unexpectedly, the middleware “cleans up” and guarantees the outcome.

Now let us see how one common variant of 2PC, *presumed abort*, can be used to effect Waldo’s transaction and move money from one account to another in a recoverable way. A key part of this illustration is to see that no matter what kind of failure occurs, data integrity is preserved and Waldo remains a loyal customer.

Figure 7-2 shows Waldo’s transaction on a time line with all of the interacting components required to execute the logic shown in Example 7-1.

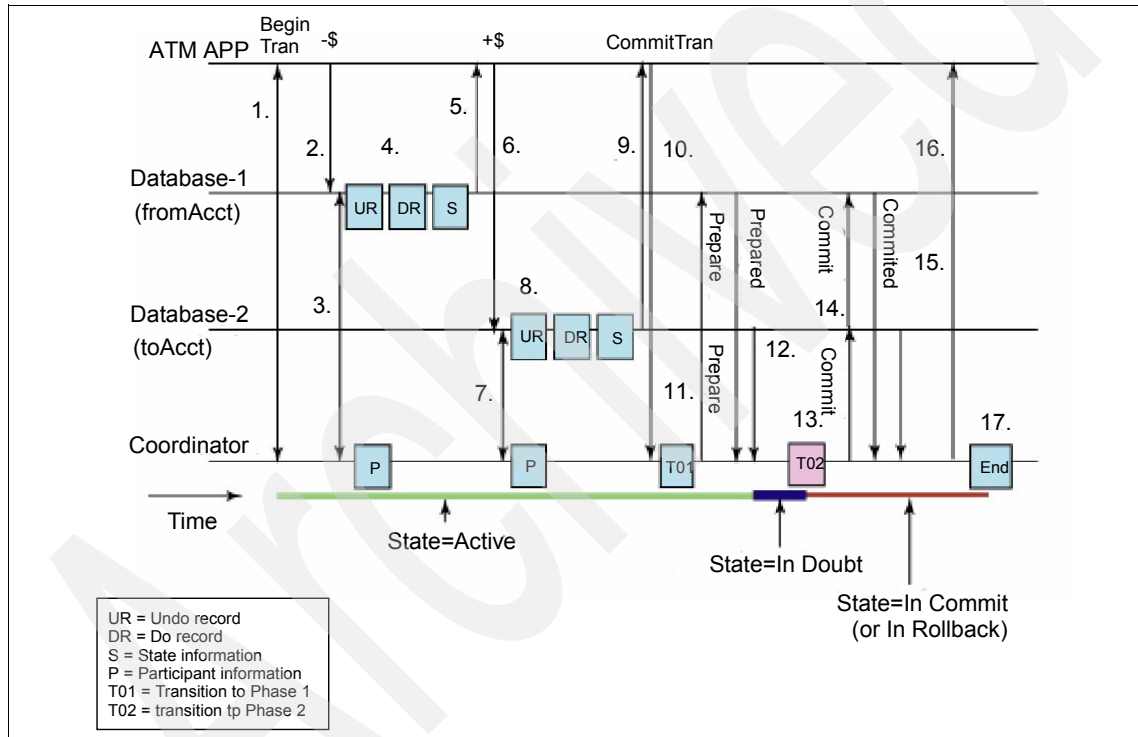


Figure 7-2 Behind the scenes of Waldo’s ATM transaction

The top line represents the ATM application itself. The next two lines represent the account databases that the application manipulates. The databases are transactional participants. The next line is a transactional coordinator, or middleware, which orchestrates the 2PC protocol. The line at the very bottom indicates the state of Waldo’s transaction at different points in time. The state of the transaction dictates recovery processing in the event of a failure.

The lines for Database-1, Database-2, and Coordinator represent both time (flowing left to right) and also some key records recorded onto a recovery log. These records include images of the data before it is modified (Undo records), images of the data after it has been modified (Do records), and state information. The recovery log is used to insure data integrity during recovery processing.

Now let us walk through Waldo's transaction. In the following discussion, when we talk about the *ATM application*, take it to mean either the application itself, or some middleware supporting the application. For example, when we say that the application begins a transactional scope, it could be that middleware begins the transactional scope on behalf of the application.

Here is our narration to explain the numbered steps shown in Figure 7-2:

1. The ATM application indicates the beginning of a transactional scope. The Coordinator creates a context for this transaction; the context includes a unique identifier and some other information about the transaction. Importantly, this transaction context flows back to the application. The context flows with other interactions between the application and resource managers; it is the context that helps glue together a whole set of actions into one transactional activity.
2. The application takes money out of Database-1. The context (from step 1) is inserted into this flow.
3. Database-1 sees the request for action, but also sees the transactional context. Database-1 uses this context to contact the transactional Coordinator and register interest in this transaction or activity (so that the Coordinator helps Database-1 through 2PC processing later to guarantee a commit or rollback outcome of all actions). The Coordinator remembers that Database-1 is a participant in the transaction.
4. Database-1 looks at the request to modify recoverable data. It writes records to a recovery log, plus transaction state information. One record describes the database change to be made if the decision later is to commit (the Do record). The other record describes the database change to be made if the decision is to rollback (the Undo record).
 - In this case, the Undo record says *make Waldo's balance = x* and the Do record says *make Waldo's balance = x - \$*. (X is the amount of the balance before this transaction ever started and \$ is the amount to transfer). Notice that we are only looking at the recovery log – not database files.
 - The Do records are not strictly required if Database-1 makes database file updates when the application requests it to, instead of waiting. However, waiting to write the data can have advantages for performance and concurrency. In addition, the Do records can be used for audit or other advanced reasons. Since they are so useful, our example databases use them.

5. Return to the application.
6. Similarly to step 2, the application makes a request to manipulate the other database, Database-2. The application wants to add in the amount taken out of Database-1.
7. Database-2 registers interest in the transaction with the Coordinator the same way Database-1 did. The Coordinator remembers that Database-2 is a participant in the transaction.
8. Database-2 writes Undo and Do records and state information to its recovery log, again just as Database-1 did.
9. Return to the application.
10. The application chooses to commit the transaction. The Coordinator now takes over. When **Commit** is received, the Coordinator writes a log record indicating that Phase 1 of 2PC has begun.
11. In Phase 1, the Coordinator goes down the list of all participants (Database-1 and Database-2 in this example) who expressed interest in this transaction, asking each one to **Prepare**. Prepare means get ready to receive the order to either commit or rollback.
12. Database-1 and Database-2 both respond with **Prepared**, meaning that they are ready to be told the final outcome (commit or rollback all the changes made) and support it.
 - They must have committed something (at least on their logs) by this point, because responding **Prepared** means they guarantee being able to commit or rollback when told; actions up to this point were just tentative.
 - If either Database had some kind of failure preparing, it would respond **Aborted** instead of **Prepared**, and the Coordinator would broadcast **Rollback** to all participants.
13. The Coordinator forces a log record indicating a Transition to Phase 2 (T02).
 - After this record is hardened on a log, we know and have recorded that:
 - All participants are prepared to go either way (commit or rollback).
 - The ultimate outcome of the transaction is known (commit, in our example).
 - The outcome is guaranteed by recovery processing.

- If this record fails to make it to the log for any reason, the ultimate outcome is to rollback (we are using *presumed abort* in this example). The recovery processing enforces the outcome.
14. The Coordinator informs each participant that the decision is to commit the changes. The participants can then do whatever they have to do, such as perhaps writing the results to the real database data.
 15. The participants return to the Coordinator with **Committed**. After the Coordinator knows that all the participants acknowledged the **Commit** order with **Committed**, it can forget about this transaction because the transaction was acknowledged by all to be done.
 16. Return to the application.
 17. At some point, because it knows that the participants have succeeded in the 2PC flow by acknowledging the common outcome, the Coordinator writes an *End* indicator on its log.

7.1.2 Mapping from classic transactions to WS-Atomic Transaction

In Figure 7-2 on page 200 we did not mention how Database-1 contacted the Coordinator, nor did we specify how the application called the databases. In fact, we did not specify the mechanisms for anything to contact anything else. In the past, these were mostly non-universal mechanisms that sometimes only worked between certain combinations of entities (applications, resource managers, and coordinators or transaction monitors).

The combination of Web services, WS-Coordination (WS-C), and WS-Atomic Transaction (WS-AT) maps all of the flows shown in Figure 7-2 on page 200 and specifies precise communications mechanisms for achieving the same results. However, instead of only working between certain combinations, the Web services based flows can work with just about anything.

Figure 7-3 illustrates how the classic flows are converted to Web services. Significantly changed steps are described following the figure. As before, when we say *application*, take it to mean the application or some helper middleware. Likewise, when we say *database*, it might mean the actual database, or some helper middleware.

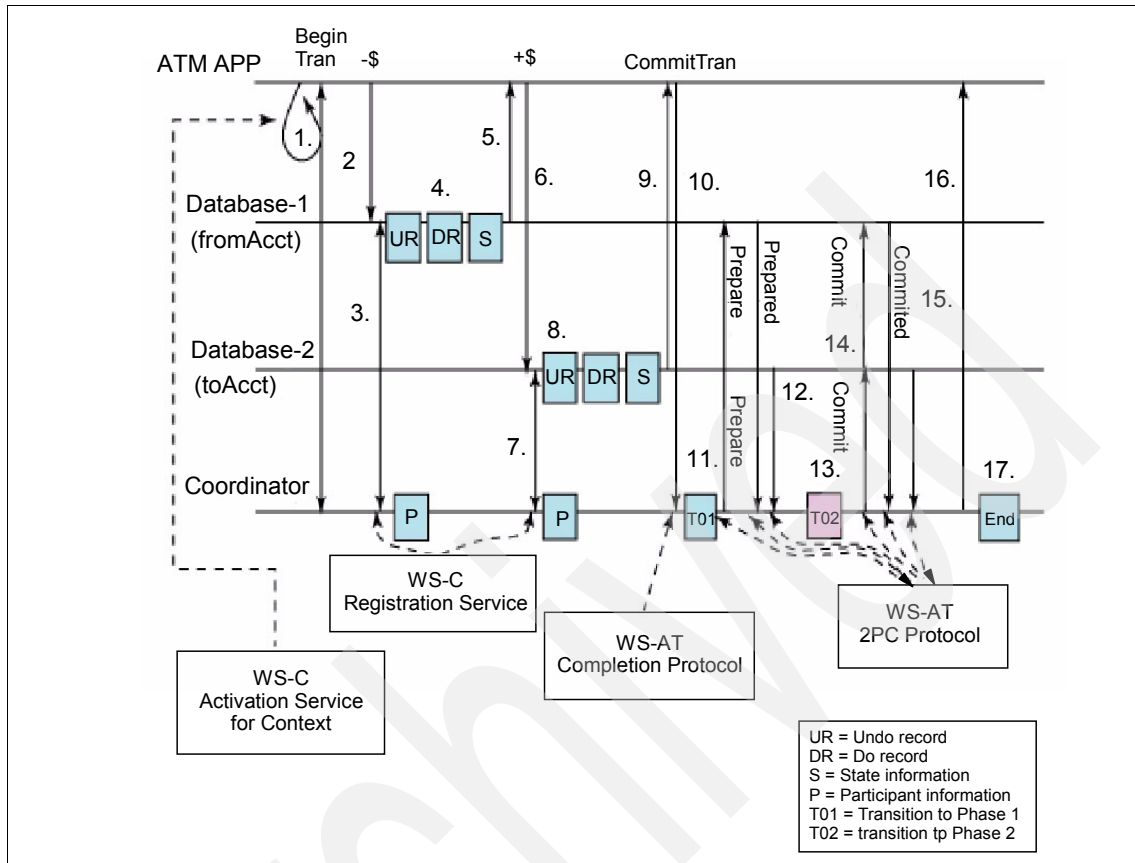


Figure 7-3 Waldo's transaction revisited

1. The application uses the Activation Service defined in WS-C to obtain a transactional context.
2. The application invokes a Web service exposed by Database-1 (alternatively, exposed by an application server that then talks to Database-1) to subtract money from Waldo's balance. The context flows along with the Web service invocation, although the application is not aware of that.
3. Database-1 uses information in the context to invoke the Registration Service defined in WS-C to register interest in this transaction.
4. No change.
5. No change.
6. The application invokes a Web service exposed by Database-2 to add money to Waldo's balance. Just like in step 2, the context flows along with the Web service invocation.

7. Just like step 3, Database-2 uses information in the context to invoke the Registration Service and register interest in this transaction.
8. No change.
9. No change.
10. The application uses the Completion Protocol defined in WS-AT to indicate that it wishes to commit the transaction.
11. to 15. Databases and the Coordinator participate in 2PC flows as defined in the WS-AT 2PC Protocol.

From Figure 7-3 it is clear that atomic transactions using Web services (WS-C and WS-AT) are substantially the same as without Web services (Figure 7-2 on page 200, for example). The primary differences are almost cosmetic from the outside and involve *how entities communicate with each other, not the substance of what they communicate*. However, these differences in how the entities communicate have a big impact on flexibility and interoperability.

You can achieve universal interoperability with Web services because instead of changing resource manager X to interoperate with transaction monitor Y, you can change both X and Y to use Web services and then interoperate with many other resource managers and transaction monitors. So instead of two-at-a-time interoperability, or interoperability only within a specific kind of domain, n-way universal interoperability is possible.

Recovery processing using Web services between the interested parties is the same as before Web services. Resource managers are the only ones who know their resources and how to commit them or roll them back.

As an example, suppose that Database-1 fails between steps 5 and 6 in Figure 7-3. Database-1 comes back up and, just like before Web services, it reads its log, notices that it has an incomplete transaction, and realizes that it has to contact the Coordinator. Information about how to contact the Coordinator is in the state saved on its recovery log; with Web services it is an endpoint reference (as defined in WS-Addressing; see “Endpoint references” on page 207).

Database-1 contacts the Coordinator at that endpoint reference with a message defined in WS-AT called **Replay**. Replay causes the Coordinator to resend the last protocol message to Database-1, which lets Database-1 deduce the transaction state and then apply the appropriate recovery rule. In our example the Coordinator tells Database-1 that it has no knowledge of this transaction. Database-1 therefore applies its Undo record, making the data consistent again.

Important: WS-AT is a two-phase commit transaction protocol that is suitable for short duration transactions only. WS-AT is well suited for distributed transactions within a single enterprise, but it is generally not recommended that WS-AT transactions be distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit.

7.2 WS-Addressing

Figure 7-3 on page 204 shows several messages flowing:

- ▶ The application sends a message to the Activation Service asking for a transactional context.
- ▶ The Activation Service sends a response containing a transactional context to the application.
- ▶ Database-1 and Database-2 each send a **Register** message to the Registration Service and receives a reply.
- ▶ The application sends a **Commit** message to the Coordinator.
- ▶ The Coordinator sends **Prepare** and **Commit** messages to Database-1 and Database-2.
- ▶ Database-1 and Database-2 each send **Prepared** and **Committed** messages to the Coordinator.

The application, Database-1, Database-2, the Activation Service, the Registration Service, and the Coordinator are endpoints for these messages. As with messages in the everyday business world, we require a way to identify the recipient of each message, the sender of the message, what previous message (if any) the message relates to, what action we want the recipient to take, and where the recipient should send the reply (if any) to the message. Furthermore, we want to be able to do this in a way that does not depend on the transport mechanism (such as HTTP or WebSphere MQ) that we use to send the message.

To this end IBM, Microsoft, Sun™ Microsystems, BEA, and SAP® formally submitted the *Web Services- Addressing (WS - Addressing)* specification to the World Wide Web Consortium (W3C) on 10 August 2004. You can find this specification at:

<http://www.w3.org/Submission/ws-addressing>

Note: As the specification has moved through the W3C standards process, it has been divided into three parts:

- ▶ Web Services Addressing 1.0 - Core
The latest version is a W3C Recommendation dated 9 May 2006:
<http://www.w3.org/TR/ws-addr-core>
- ▶ Web Services Addressing 1.0 - SOAP Binding
The latest version is a W3C Recommendation dated 9 May 2006:
<http://www.w3.org/TR/ws-addr-soap>
- ▶ Web Services Addressing 1.0 - Metadata
The latest version is a W3C Recommendation dated 4 September 2007:
<http://www.w3.org/TR/ws-addr-metadata>

However, the WS-Coordination and WS-Atomic Transaction specifications that we discuss later in this chapter are based on the original 10 August 2004 specification, and, therefore, so is CICS TS V3.2 support for Web services.

All information items defined by the 10 August 2004 specification are identified by the XML namespace URI:

`http://schemas.xmlsoap.org/ws/2004/08/addressing`

We associate the namespace prefix *wsa* with this namespace by using the attribute

`xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"`

The specification defines two constructs:

- ▶ Endpoint references
- ▶ Message information headers

7.2.1 Endpoint references

A Web service *endpoint* is a (referenceable) entity, processor, or resource to which Web services messages can be addressed.

An *endpoint reference* conveys the information required to address a Web service endpoint. As we shall see later:

- ▶ An endpoint reference for the Registration Service forms part of the coordination context.

- ▶ An endpoint reference for the participant's Protocol Service forms part of the Register request.
- ▶ An endpoint reference for the coordinator's Protocol Service forms part of the response to a Register request.

Example 7-2 shows the pseudo schema for an EndpointReference element.

Example 7-2 Pseudo schema for EndpointReference element

```

<wsa:EndpointReference>
  <wsa:Address>.....</wsa:Address>
  <wsa:ReferenceProperties>.....</wsa:ReferenceProperties>
  <wsa:ReferenceParameters>.....</wsa:ReferenceParameters>
  <wsa:PortType>.....</wsa:PortType>
  <wsa:ServiceName PortName="...">.....</wsa:ServiceName>
  <wsp:Policy>.....</wsp:Policy>
</wsa:EndpointReference>

```

For each child element of EndpointReference, Table 7-1 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Table 7-1 Children of the Endpoint Reference element

Element	Description	Min	Max
Address	Contains an address URI that identifies the endpoint. This can be a network address or a logical address.	1	1
ReferenceProperties	Contains child elements each of which represents an individual reference property. The number of child elements is not limited.	0	1
ReferenceParameters	Contains child elements each of which represents an individual reference parameter. The number of child elements is not limited.	0	1
PortType	Contains the name of the primary portType of the endpoint being conveyed.	0	1
ServiceName	Contains the name of the WSDL <service> element that contains a WSDL description of the endpoint being referenced. The service name provides a link to a full description of the service endpoint. The ServiceName element can optionally have a PortName attribute which specifies the name of the specific WSDL <port> definition in that service which corresponds to the endpoint being referenced.	0	1

Element	Description	Min	Max
Policy	Contains an XML policy element as described in WS-Policy that describes the behavior, requirements, and capabilities of the endpoint.	0	No limit

Here is the difference between a reference property and a reference parameter:

- ▶ A *reference property* is required to identify the endpoint. It is required to properly dispatch a message to an endpoint at the endpoint side of the interaction.
- ▶ A *reference parameter* is required to facilitate a particular interaction with the endpoint. It is required to properly interact with the endpoint.

Tip: In “Transaction scenarios” on page 251 we look at the messages that flow between CICS TS V3.1 and WebSphere Application Server V6.0. In these messages we see only the following children of the EndpointReference element:

- ▶ Address
- ▶ ReferenceProperties

Example 7-3 shows an endpoint reference for the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MMSG3.mop.ibm.com. The endpoint reference has two reference properties: UOWID and PublicID. We replaced the ending characters of the PublicID property with ... for brevity.

Example 7-3 Sample EndpointReference element for Registration Service in CICS

```

<wsa:EndpointReference
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline">
  <wsa:Address>
    http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
  </wsa:Address>
  <wsa:ReferenceProperties>
    <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
    <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
  </wsa:ReferenceProperties>
</wsa:EndpointReference>

```

Example 7-4 shows an endpoint reference for the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156. The endpoint reference has two reference properties: txID and instanceID. We replaced the ending characters of these properties with ... for brevity.

Example 7-4 Sample EndpointReference element for Registration Service in WAS

```

<wsa:EndpointReference
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">
  <wsa:Address>
    http://9.100.199.156:9080/_IBMSYSAPP/wscor/services/Registration
    CoordinatorPort
  </wsa:Address>
  <wsa:ReferenceProperties>
    <websphere-wsat:txID>
      com.ibm.ws.wstx:00000107d70ad26c0000000700000005cbb17a7d5f7e...
    </websphere-wsat:txID>
    <websphere-wsat:instanceID>
      com.ibm.ws.wstx:00000107d70ad26c0000000700000005cbb17a7d5f7e...
    </websphere-wsat:instanceID>
  </wsa:ReferenceProperties>
</wsa:EndpointReference>

```

7.2.2 Message information headers

Example 7-5 shows a pseudo schema for the message information headers defined in the WS-Addressing specification.

Example 7-5 Pseudo schema for message information header elements

```

<wsa:MessageID>.....</wsa:MessageID>
<wsa:RelatesTo RelationshipType="...">...</wsa:RelatesTo>
<wsa:To>.....</wsa:To>
<wsa:Action>.....</wsa:Action>
<wsa:From>.....</wsa:From>
<wsa:ReplyTo>.....</wsa:ReplyTo>
<wsa:FaultTo>.....</wsa:FaultTo>

```

Table 7-2 describes what each message information header element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Table 7-2 Message information header elements

Element	Description	Min	Max
MessageID	Contains a URI that uniquely identifies this message in space and time.	0 (but see Note 1)	1
RelatesTo	Contains a URI that corresponds to a related message's MessageID property. The RelatesTo element has an optional RelationshipType attribute that indicate the type of relationship this message has to the related message. The specification defines one relationship type, namely wsa:Reply. When absent, the implied value of this attribute is wsa:Reply.	0	No limit
To	Contains a URI that specifies the address of the intended receiver of this message.	1	1
Action	Contains a URI that uniquely identifies the semantics implied by this message.	1	1
From	Contains an endpoint reference that identifies the endpoint from which the message originated.	0	1
ReplyTo	Contains an endpoint reference that identifies the intended receiver for replies to this message.	0 (but see Note 2)	1
FaultTo	Contains an endpoint reference that identifies the intended receiver for faults related to this message.	0 (see also Note 3)	1

Table notes:

1. If ReplyTo or FaultTo is present, MessageID must be present.
2. If a reply is expected, ReplyTo *must* be present.
3. When formulating a fault message, the sender *must* use the contents of the FaultTo element of the message to which the Fault reply is being sent. If the FaultTo element is absent, the sender *can* use the contents of the ReplyTo element to formulate the fault message. If both the FaultTo element and the ReplyTo element are absent, the sender *can* use the contents of the From element to formulate the fault message. The FaultTo element can be absent if the sender cannot receive fault messages.

Example 7-6 shows an example of a set of message information headers.

Example 7-6 Sample message information header elements

```
<wsa:To>
  http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/Registration
  CoordinatorPort
```

```
</wsa:To>
<wsa:Action>
  http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register
</wsa:Action>
<wsa:MessageID>PIAT-MSG-A6P0T3C1-003342266297785C</wsa:MessageID>
<wsa:ReplyTo>
  <wsa:Address>
    http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
  </wsa:Address>
  <wsa:ReferenceProperties>
    <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
    <cicswsat:PublicId>
      310FD7E2E2...
    </cicswsat:PublicId>
  </wsa:ReferenceProperties>
</wsa:ReplyTo>
```

The To header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156.

The Action header indicates that the sender wishes to register with the Registration Coordinator Port.

The ID of the message, PIAT-MSG-A6P0T3C1-003342266297785C, uniquely identifies the message in space and time:

- ▶ A6P0T3C1 is the VTAM® APPLID of the CICS TS V3.1 region that sent the message.
- ▶ 003342266297785C is the abstime value returned by an EXEC CICS INQUIRE TIME issued in that CICS TS V3.1 region.

The ReplyTo header shows that the reply to this message should be sent to the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com.

7.2.3 SOAP binding for endpoint references

When a SOAP message must be addressed to an endpoint, the information contained in the endpoint reference is mapped to the SOAP message by the following two rules:

- ▶ The contents of the Address element in the endpoint reference is copied to the To message information header of the SOAP message.
- ▶ Each reference property or reference parameter is added as a header block in the new message.

Example 7-7 shows how we use these rules to address a message to the CICS Registration Service endpoint whose endpoint reference is given in Example 7-3 on page 209.

Example 7-7 SOAP message addressed to CICS RegistrationService endpoint

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:cicswsat="...">
  <S:Header>
    ...
    <wsa:To>
      http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
    </wsa:To>
    <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
    <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
    ...
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

7.3 WS-Coordination

In 7.1, “Beginner’s guide to atomic transactions” on page 198 and 7.2, “WS-Addressing” on page 206, we see that the application, Database-1, Database-2, the Activation service, the Registration service, and the Coordinator are endpoints for messages. We also see that:

- ▶ The application sends a message to the Activation service asking for a transactional context.
- ▶ The Activation service sends a response containing a transactional context to the application.
- ▶ Database-1 and Database-2 each sends a **Register** message to the Registration service and receives a reply.

We still have to define what these messages should contain.

For this purpose IBM, Microsoft, and BEA published the *Web Services-Coordination (WS - Coordination)* specification in September of 2003; they updated it in November of 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services-Coordination (WS-Coordination) Version 1.0* in August of 2005. You can find these at:

<http://www-128.ibm.com/developerworks/library/specification/ws-tx>

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

<http://schemas.xmlsoap.org/ws/2004/10/wscor>

We associate the namespace prefix *wscor* with this namespace by using the attribute:

```
xmlns:wscor="http://schemas.xmlsoap.org/ws/2004/10/wscor"
```

The specification defines:

- ▶ A coordination service
- ▶ The following messages:
 - CreateCoordinationContext
 - CreateCoordinationContextResponse
 - Register
 - RegisterResponse

7.3.1 Coordination service

As shown in Figure 7-4, a *Coordination service* (or *Coordinator*) is an aggregation of the following services:

- ▶ Activation service:

When the application sends a `CreateCoordinationContext` element, the Activation service creates a new activity and returns its coordination context in a `CreateCoordinationContextResponse` element.

The Coordination service can, but does not have to, support the Activation service.

Note: Some products provide this as an external service, for others to call. CICS does not do this, and only supports the creation of coordination contexts internally, for use by the workloads that it manages.

- ▶ Registration service:

The Registration service defines a **Register** operation that allows a Web service to register to participate in a coordination protocol.

The Coordination service must support the Registration service.

- ▶ A set of coordination Protocol services for each supported coordination type:

These are defined in the specification that defines the coordination type (for example, in the WS-Atomic Transaction specification).

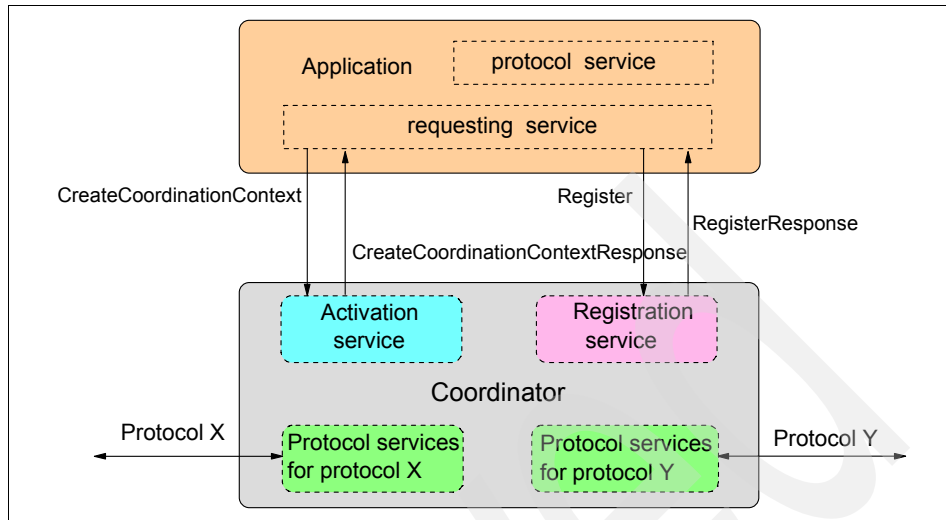


Figure 7-4 A Coordination service (or Coordinator)

7.3.2 CreateCoordinationContext

Example 7-8 shows the pseudo schema for the `CreateCoordinationContext` element.

Example 7-8 Pseudo schema for `CreateCoordinationContext` element

```

<wscoor:CreateCoordinationContext...>
  <wscoor:CoordinationType>.....</wscoor:CoordinationType>
  <wscoor:Expires>.....</wscoor:Expires>
  <wscoor:CurrentContext>.....</wscoor:CurrentContext>
</wscoor:CreateCoordinationContext>

```

For each child element of `CreateCoordinationContext`, Table 7-3 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Table 7-3 Children of the `CreateCoordinationContext` element

Element	Description	Min	Max
CoordinationType	The unique identifier for the desired coordination type for the activity.	1	1

Element	Description	Min	Max
Expires	The expiration for the returned CoordinationContext expressed as an unsigned integer in milliseconds. Specifies the earliest point in time at which a transaction can be terminated solely due to its length of operation.	0	1
CurrentContext	The current Coordination Context.	0	1

Currently there are two specifications that define coordination types:

► **WS-Atomic Transaction:**

This specification defines the coordination type for atomic transactions where the results of operations are not made visible until the completion of the unit of work:

<http://schemas.xmlsoap.org/ws/2004/10/wsat>

► **WS-Business Activity:**

This specification defines two coordination types for business activities:

<http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome>

<http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome>

Business activities have the following characteristics:

- A business activity can consume many resources over a long duration.
- There can be a significant number of atomic transactions involved.
- Individual tasks within a business activity can be seen prior to the completion of the business activity; their results can have an impact outside of the computer system.
- Responding to a request can take a very long time. Human approval, assembly, manufacturing, or delivery might have to take place before a response can be sent.
- In the case where a business exception requires an activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms might require business logic, for example, in the form of a compensation task, to reverse the effects of a previously completed task.

For example, selling a vacation is a business activity that might involve the travel agent in actions such as recording customer details, booking seats on an aircraft, booking a hotel, booking a rental car, invoicing the customer, checking for receipt of payment, processing the payment, and arranging foreign currency.

Table 7-4 compares an atomic transaction with a business activity.

Table 7-4 Comparison of features of atomic transaction and business activity

Atomic transaction	Business activity
Short duration	Longer duration
Locks	Avoid locks
Suited for a more controlled environment	Suited for a loosely coupled environment
Classical resource manager mapping - think database (not business processes crossing business boundaries)	Business process mapping
Easier to think about and program <ul style="list-style-type: none"> ▶ Rollback or Commit ▶ Automatic rollback in case of abnormal termination 	More complex <ul style="list-style-type: none"> ▶ Compensate
All resource managers move in one direction (everybody commits or rolls back in unison)	More flexible resource manager participation. They do not have to trust applications so much.

Note: CICS TS V3.2 does not support the WS-Business Activity specification. At this time there are no plans for future releases of CICS to support it either.

Example 7-9 shows an example of a CreateCoordinationContext element in which the coordination type is WS-Atomic Transaction.

Example 7-9 Sample CreateCoordinationContext element

```

<wscoor:CreateCoordinationContext>
  <wscoor:CoordinationType>
    http://schemas.xmlsoap.org/ws/2004/10/wsat
  </wscoor:CoordinationType>
  <wscoor:Expires>5000</wscoor:Expires>
</wscoor:CreateCoordinationContext>

```

7.3.3 CreateCoordinationContextResponse

The `CreateCoordinationContextResponse` element contains the `CoordinationContext` element.

The `CoordinationContext` element contains four elements:

- ▶ A URI that identifies the `CoordinationContext`.
- ▶ The `CoordinationType`.
- ▶ The expiration period for the `CoordinationContext`.
- ▶ An endpoint reference for the Registration Service which is part of this `Coordination` service. Recall from “Endpoint references” on page 207 that this means that the `CoordinationContext` must contain the address of the Registration Service and can contain reference properties for the Registration Service.

Example 7-10 shows the pseudo schema for the element, `CreateCoordinationContextResponse`.

Example 7-10 Pseudo schema for CreateCoordinationContext Response element

```
<wscoor:CreateCoordinationContextResponse>
  <wscoor:CoordinationContext>
    <wscoor:Identifier>.....</wscoor:Identifier>
    <wscoor:CoordinationType>.....</wscoor:CoordinationType>
    <wscoor:Expires>.....</wscoor:Expires>
    <wscoor:RegistrationService>.....</wscoor:RegistrationService>
  </wscoor:CoordinationContext>
</wscoor:CreateCoordinationContextResponse>
```

The application places the `CoordinationContext` element within an application message to pass the coordination information to other parties. Conveying a `CoordinationContext` on an application message is commonly referred to as *flowing* the context. When a context is flowed as a SOAP header, the header must have the `mustUnderstand` attribute and the value of the `mustUnderstand` attribute must be `true`.

Note: When an application flows the context, it passes the address of its Registration Service and the coordination type.

Example 7-11 shows a `CoordinationContext` created by a CICS TS V3.1 region.

Example 7-11 Sample CoordinationContext created by CICS

```
<wscoor:CoordinationContext>
  <wscoor:Identifier>
    PIAT-CCON-A6POT3C1-003322404576825C
  </wscoor:Identifier>
  <wscoor:CoordinationType>
    http://schemas.xmlsoap.org/ws/2004/10/wsat
  </wscoor:CoordinationType>
  <wscoor:RegistrationService>
    <wsa:Address>
      http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
    </wsa:Address>
    <wsa:ReferenceProperties>
      <cicswsat:NetName>A6POT3C1</cicswsat:NetName>
      <cicswsat:Token>FOF0FOF0</cicswsat:Token>
      <cicswsat:UOWID>BCDB8F2E852B924C</cicswsat:UOWID>
    </wsa:ReferenceProperties>
  </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Example 7-12 shows a CoordinationContext created by WebSphere Application Server V6.0.

Example 7-12 Sample CoordinationContext created by WebSphere

```
<wscoor:CoordinationContext>
  <wscoor:Expires>Never</wscoor:Expires>
  <wscoor:Identifier>
    com.ibm.ws.wstx:00000107d70ad2...
  </wscoor:Identifier>
  <wscoor:CoordinationType>
    http://schemas.xmlsoap.org/ws/2004/10/wsat
  </wscoor:CoordinationType>
  <wscoor:RegistrationService>
    <wsa:Address>
      http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
    </wsa:Address>
    <wsa:ReferenceProperties>
      <websphere-wsat:txID xmlns:websphere-wsat=".....">
        com.ibm.ws.wstx:00000107d70ad2...
      </websphere-wsat:txID>
      <websphere-wsat:instanceID xmlns:websphere-wsat=".....">
        com.ibm.ws.wstx:00000107d70ad2...
      </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
  </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Note: We noted from our tests that the content of the Expires element (that is, Never) is not an unsigned integer as required by the specification. This is planned to be changed in a future release of WebSphere Application Server.

7.3.4 Register

Before we provide the details of the Register request, we consider some concepts in this section so that you do not lose sight of the “big picture.”

The Coordinator provides the application with the Endpoint reference of its Registration service in the CreateCoordinationContextResponse. The application then knows where and how to send a Register request.

Figure 7-5 shows how Endpoint references are used during and after registration:

1. The Register message targets the Endpoint reference of the Coordinator's Registration Service and includes the Endpoint reference of the application's Protocol service as a parameter.
2. The Register Response includes the Endpoint reference of the Coordinator's Protocol service.
3. At this point, both sides have the Endpoint Reference of the other's Protocol service, so the protocol messages can target the other side.

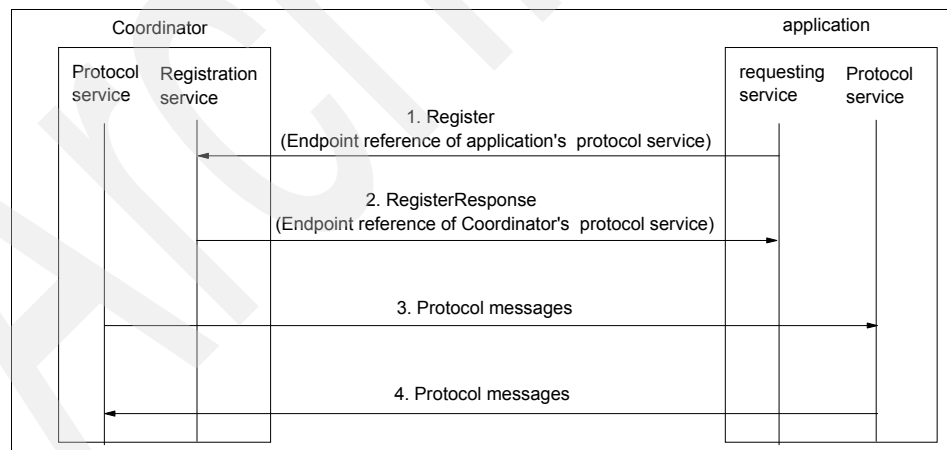


Figure 7-5 Using Endpoint references during and after registration

Now that you understand the big picture, we provide the details. Example 7-13 shows the pseudo schema for the Register element.

Example 7-13 Pseudo schema for the Register element

```
<wscoor:Register>
  <wscoor:ProtocolIdentifier>.....</wscoor:ProtocolIdentifier>
  <wscoor:ParticipantProtocolService>.....</wscoor:ParticipantProtocolService>
</wscoor:Register>
```

The ProtocolIdentifier element contains a URI that provides the identifier of the coordination protocol selected for registration. The contents of the CoordinationType element of the CoordinationContext element determine the possible choices for the ProtocolIdentifier element as follows:

- ▶ If the CoordinationType is atomic transaction, then the ProtocolIdentifier must be one of the following possibilities:
 - <http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion>
 - <http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC>
 - <http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC>
- ▶ If the CoordinationType is business activity (either AtomicOutcome or MixedOutcome), then the ProtocolIdentifier must be one of these:
 - <http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithParticipantCompletion>
 - <http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithCoordinatorCompletion>

The ParticipantProtocolService element contains the EndpointReference that the registering participant wants the Coordinator to use for the Protocol service.

Note: As we noted earlier, CICS TS V3.2 does not support the WS-Business Activity specification. Therefore, it does not support either the BusinessAgreementWithParticipantCompletion or the BusinessAgreementWithCoordinatorCompletion protocol identifiers.

Example 7-14 shows a Register element created by a CICS TS V3.1 region.

Example 7-14 Sample Register element

```
<wscoor:Register>
  <wscoor:ProtocolIdentifier>
    http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
  </wscoor:ProtocolIdentifier>
  <wscoor:ParticipantProtocolService>
    <wsa:Address>
      http://MVS3.mop.ibm.com:15301/cicswsat/RegistrationService
    </wsa:Address>
  </wscoor:ParticipantProtocolService>
</wscoor:Register>
```

```
<wsa:ReferenceProperties>
  <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
  <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
</wsa:ReferenceProperties>
</wscoor:ParticipantProtocolService>
</wscoor:Register>
```

Note: CICS TS V3.2 uses a single endpoint address for its Registration service, Protocol service, and fault messages. Other products, such as WebSphere Application Server V6.0, use separate addresses for each of these.

7.3.5 Register response

Example 7-15 shows the pseudo schema for the RegisterResponse element.

Example 7-15 Pseudo schema for the RegisterResponse element

```
<wscoor:RegisterResponse>
  <wscoor:CoordinatorProtocolService>.....</wscoor:CoordinatorProtocolService>
</wscoor:RegisterResponse>
```

The CoordinatorProtocolService element contains the EndpointReference that the Coordination service wants the registered participant to use for the Protocol service.

Again, we note that when the application has received the RegisterResponse, each side has the EndpointReference of the other's Protocol Service.

7.3.6 Two applications with their own coordinators

In this section we see how two application services (App1 and App2) with their own coordinators (Coordinator A and Coordinator B) interact as an activity propagates between them. Coordinator A provides Activation service ASa, Registration service RSa, and Protocol service Pa. Coordinator B provides Activation service ASb, Registration service RSb, and Protocol service Pb.

Figure 7-6 shows the two applications with their own coordinators.

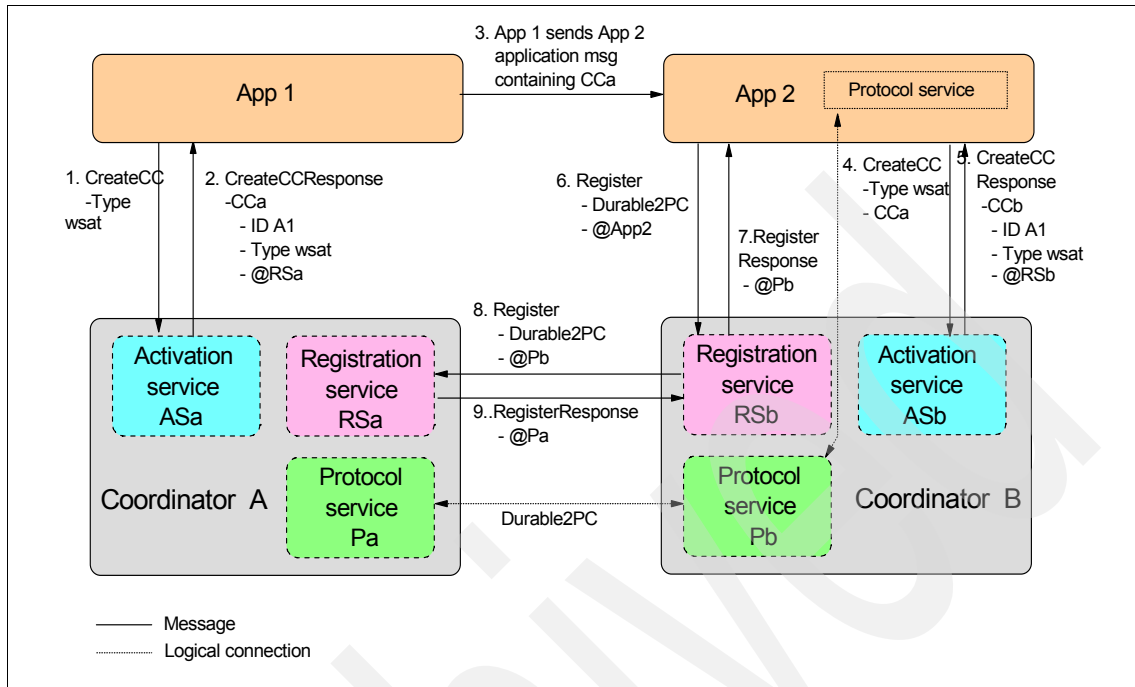


Figure 7-6 Two applications with their own coordinators

In Figure 7-6, the following actions take place:

1. App1 sends to ASa a CreateCoordinationContext element that specifies a CoordinationType of wsat.
2. ASa sends to App1 a CreateCoordinationContextResponse element that includes CoordinationContext CCa. CCa contains an Identifier element whose content is A1, a CoordinationType element whose content is wsat, and a RegistrationService element that contains an EndpointReference to Coordinator A's Registration service RSa.
3. App1 then sends to App2 a SOAP message that has a SOAP header that contains CCa and that has a mustUnderstand attribute with a value of True.
4. App2 prefers Coordinator B, so it sends to ASb a CreateCoordinationContext element that specifies a CurrentContext of CCa.
5. Coordinator B creates its own CoordinationContext element CCb that contains the same Identifier and CoordinationType as CCa but with an EndpointReference to its own RegistrationService RSb.

The *WS-Atomic Transaction* specification, which we discuss in detail in “WS-Atomic Transaction” on page 224, states that:

- If the `CreateCoordinationContext` request includes the `CurrentContext` element, then the target coordinator is interposed as a subordinate to the coordinator stipulated inside the `CurrentContext` element.
 - If the `CreateCoordinationContext` request does not include a `CurrentContext` element, then the target coordinator creates a new transaction and acts as the root coordinator.
6. App2 determines the coordination protocols supported by the `wsat` coordination type and then sends a `Register` element to RSb. This `Register` element contains a `ProtocolIdentifier` of `Durable2PC` and an `EndpointReference` to App2’s Protocol service.
 7. RSb sends back to App2 a `RegisterResponse` element that contains an `EndpointReference` to Protocol service Pb. This forms a logical connection (which the `Durable2PC` protocol can use) between the `Endpoint` reference for App2’s Protocol service and the `Endpoint` reference for Coordinator B’s Protocol service Pb.
 8. This registration causes Coordinator B to forward the registration on to Coordinator A’s Registration service RSa.
 9. RSa sends back to Coordinator B a `RegisterResponse` element that contains an `EndpointReference` to Protocol Service Pa. This forms a logical connection between the `EndpointReferences` for Pa and Pb that the `Durable2PC` protocol can use.

7.3.7 Addressing requirements for WS-Coordination message types

The `CreateCoordinationContext` and `Register` messages:

- ▶ *Must* include a `wsa:MessageID` header
- ▶ *Must* include a `wsa:ReplyTo` header

The `CreateCoordinationContextResponse` and `RegisterResponse` messages:

- ▶ *Must* include a `wsa:RelatesTo` header that specifies the `MessageID` from the corresponding request message

7.4 WS-Atomic Transaction

As we discussed in Section 7.3, “WS-Coordination” on page 213, the WS-Coordination specification defines an extensible framework for defining coordination types. The WS-Atomic Transaction specification builds on

WS-Coordination by providing the definition of the atomic transaction coordination type.

Atomic transactions have an *all-or-nothing* property. The actions taken prior to commit are only tentative (that is, not persistent and not visible to other activities). When an application finishes, it requests the Coordinator to determine the outcome for the transaction. The Coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the Coordinator commits all actions taken. If a participant votes that it has to abort or a participant does not respond at all, the Coordinator aborts all actions taken. Commit makes the tentative actions visible to other transactions. Abort makes the tentative actions appear as though the actions never happened.

IBM, Microsoft, and BEA published the *Web Services- Atomic Transaction (WS - Atomic Transaction)* specification in September, 2003; they updated it in November, 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services - Atomic Transaction (WS - Atomic Transaction) Version 1.0* in August, 2005. You can find these at:

<http://www-128.ibm.com/developerworks/library/specification/ws-tx>

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

<http://schemas.xmlsoap.org/ws/2004/10/wsat>

We associate the namespace prefix *wsat* with this namespace by using the attribute:

```
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
```

The WS-AT specification defines the following protocols for atomic transactions:

- ▶ Completion
- ▶ Volatile Two-Phase Commit
- ▶ Durable Two-Phase Commit

7.4.1 Completion protocol

The Completion protocol is used by an application to tell the Coordinator to try to either commit or abort an atomic transaction. The Completion protocol initiates commitment processing. Based on each protocol's registered participants, the Coordinator begins with Volatile 2PC and then proceeds through Durable 2PC. After the transaction has completed, a status (Committed or Aborted) is returned to the application.

An initiator registers for this protocol by specifying the following URI for the contents of the ProtocolIdentifier element in the Register element:

<http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion>

Figure 7-7 illustrates the protocol abstractly.

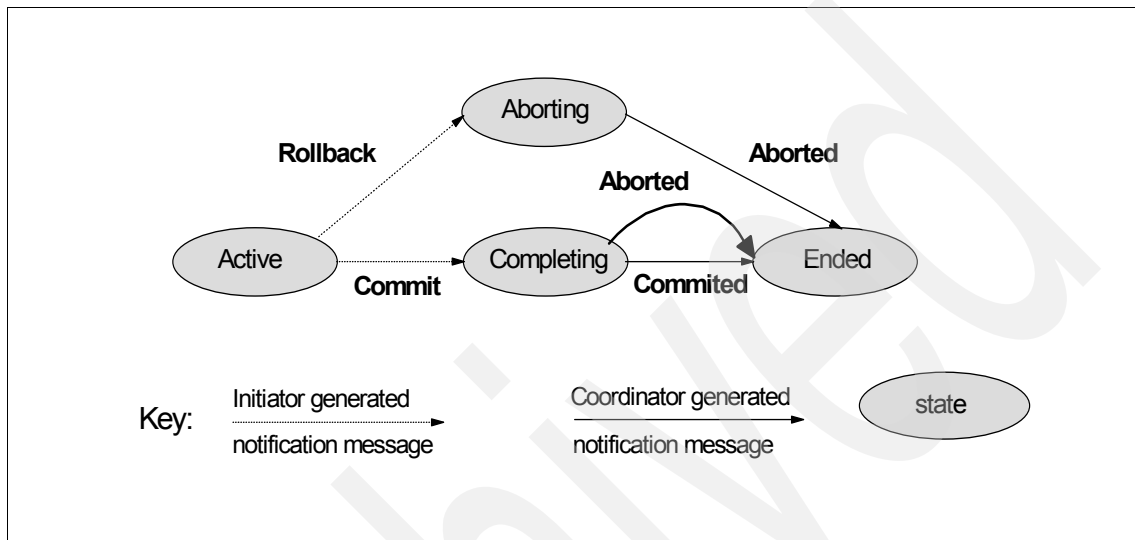


Figure 7-7 Completion protocol

The initiator generates:

▶ **Commit:**

Upon receipt of this notification, the Coordinator knows that the initiator has *completed* application processing and that it should attempt to commit the transaction.

▶ **Rollback:**

Upon receipt of this notification, the Coordinator knows that the initiator has *terminated* application processing and that it should abort the transaction.

The Coordinator generates:

▶ **Committed:**

Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to commit.

▶ **Aborted:**

Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to abort.

7.4.2 Two-Phase Commit protocol

The Two-Phase Commit (2PC) protocol defines how multiple registered participants reach agreement on the outcome of an atomic transaction. The 2PC protocol has two variants: Volatile 2PC and Durable 2PC.

Participants managing volatile resources such as a cache should register for this protocol by using the following protocol identifier:

`http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC`

Participants managing durable resources such as a database should register for this protocol by using the following protocol identifier:

`http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC`

Note: When CICS TS V3.2 is a participant in an atomic transaction, it always requests the Durable2PC protocol when it sends a Register request. When CICS TS V3.2 is the coordinator of an atomic transaction, it tolerates a Register request for Volatile2PC but it treats it as a Durable2PC request.

After receiving a **Commit** notification in the Completion protocol, the root Coordinator begins the Prepare phase of all participants registered for the Volatile 2PC protocol. All participants registered for this protocol must respond before a Prepare is issued to a participant registered for the Durable 2PC protocol. We illustrate this in Figure 7-8, where participants P1 and P3 registered for the Volatile 2PC protocol and participant P2 registered for the Durable 2PC protocol. Both P1 and P3 must respond to the **Prepare** notification before the Coordinator can send **Prepare** to P2.

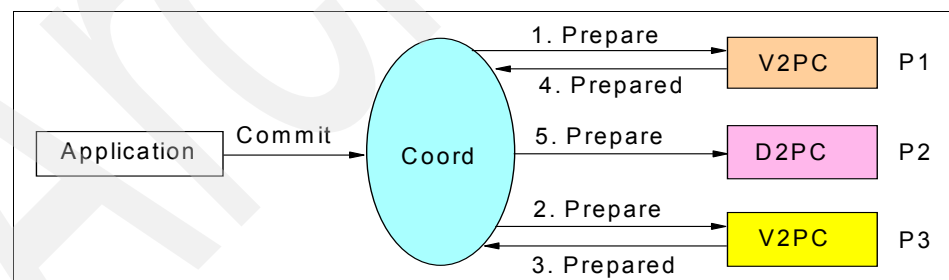


Figure 7-8 Mixture of participants registered for Durable 2PC and Volatile 2PC

Upon successfully completing the prepare phase for Volatile 2PC participants, the root Coordinator begins the Prepare phase for Durable 2PC participants. All participants registered for this protocol must respond Prepared or ReadOnly before a Commit notification is issued to a participant registered for either protocol. A volatile participant is not guaranteed to receive a notification of the transaction's outcome.

Figure 7-9 illustrates the 2PC protocol abstractly.

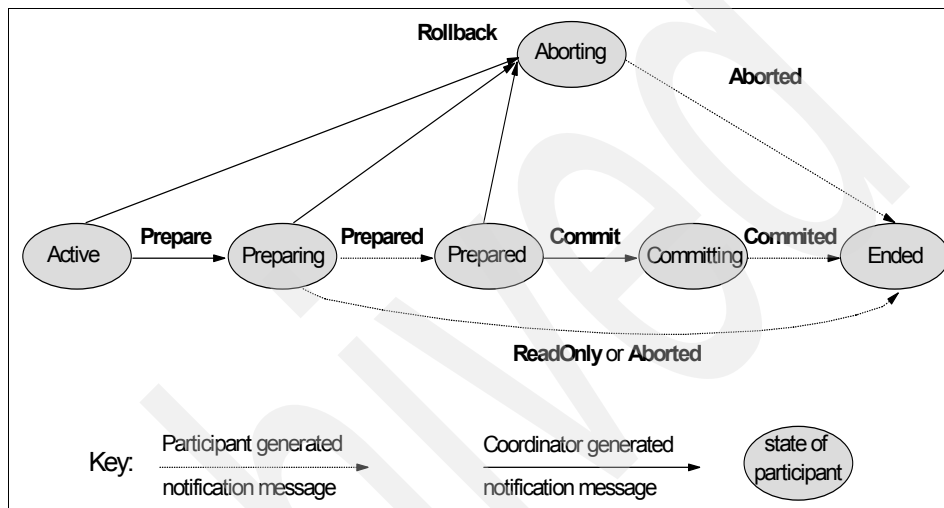


Figure 7-9 Two-Phase Commit protocol

The Coordinator generates:

► **Prepare**

Upon receipt of this notification, the participant should enter phase 1 and vote on the outcome of the transaction.

- If the participant has already voted, it should resend the same vote.
- If the participant does not know of the transaction, it must vote to abort.

► **Rollback**

Upon receipt of this notification, the participant should abort, and forget, the transaction. This notification can be sent in either phase 1 or phase 2. After being sent, the Coordinator can forget all knowledge of this transaction.

▶ **Commit**

Upon receipt of this notification, the participant should commit the transaction. This notification can only be sent after phase 1, and if the participant voted to commit. If the participant does not know of the transaction, it must send a **Committed** notification to the Coordinator.

The participant generates:

▶ **Prepared**

The participant is prepared and votes to commit the transaction.

▶ **ReadOnly**

The participant votes to commit the transaction and has forgotten the transaction. The participant does not wish to participate in phase two.

Suppose, for example, that the participant received an account number that it could not match to an entry in a database. It might return an error to the requesting application, but, having registered as a participant in the atomic transaction, it would then go on to be coordinated during 2PC processing. When the Coordinator sends **Prepare**, the participant replies **ReadOnly** and then terminates without waiting for the **Commit**. The Coordinator, on receipt of the **ReadOnly**, would then delete its own record of the interaction with the participant and would not attempt to send a **Commit** to it.

▶ **Aborted**

The participant has aborted, and forgotten, the transaction.

▶ **Committed**

The participant has committed the transaction. The Coordinator can safely forget that participant.

▶ **Replay**

The participant has suffered a recoverable failure. The Coordinator should resend the last appropriate protocol notification.

7.4.3 Two applications with their own coordinators (continued)

In 7.3.6, “Two applications with their own coordinators” we saw how two application services (App1 and App2) with their own coordinators (Coordinator A and Coordinator B) used the WS-Coordination specification to interact as an activity propagated between them. Figure 7-10 shows how they use the WS-AT specification to complete their global unit of work.

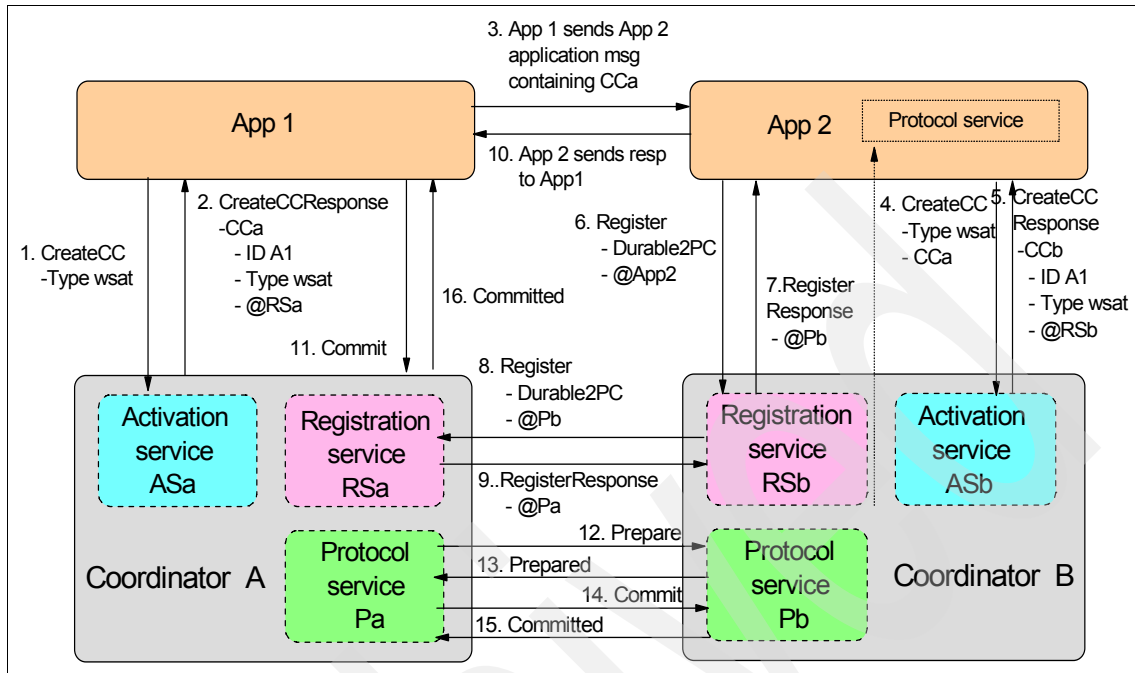


Figure 7-10 Two applications with their own coordinators (continued)

After completing its work, App2 sends its response back to App1 (step 10 in Figure 7-10). When App1 completes its work, it sends a **Commit** message to Coordinator A (step 11). This causes Coordinator A's Durable 2PC Protocol service Pa to send a **Prepare** notification to Coordinator B's Durable 2PC Protocol service Pb (step 12). We assume that Pb responds with a **Prepared** notification. If Pa encounters no other errors, it makes the decision to commit the atomic transaction and sends a **Commit** notification to Pb (step 14). Pb returns a **Committed** notification and then completes its updates before terminating. When Coordinator A receives this notification, it commits its own updates and notifies App1 of the outcome (step 16).

7.4.4 Addressing requirements for WS-AT message types

The messages defined in the WS-AT specification are *notification* messages; that is, they are *one-way* messages. There are two types of notification messages:

- ▶ A notification message is a *terminal* message when it indicates the end of a coordinator/participant relationship. **Committed**, **Aborted**, and **ReadOnly** are terminal messages.

- ▶ A notification message is a *non-terminal* message when it does not indicate the end of a coordinator/participant relationship. **Commit**, **Rollback**, **Prepare**, **Prepared**, and **Reply** are non-terminal messages.

Non-terminal notification messages *must* include a `wsa:ReplyTo` header.

7.4.5 CICS TS V3.2 and resynchronization processing

We have completed our discussion of the WS-AT specification. Unfortunately, the current version of the specification does not completely cover all of the issues surrounding the use of the 2PC protocol. In particular, it does not completely describe the resynchronization processing that should take place following a failure in one of the systems involved in the 2PC protocol or in the network connections that link the systems together. The only thing that the specification mentions relating to resynchronization is the **Reply** notification. Therefore, in this section we describe some aspects of how CICS TS V3.2 handles resynchronization processing for transactions that use the 2PC protocol.

Note: For the sake of brevity we do not describe all of the possible issues. For example, we do not describe:

- ▶ What happens when resync processing is driven from both sides and a *race* condition results
- ▶ What happens when a resync request fails

Network failures can result in messages not being delivered in a timely manner. System failures prevent processing altogether until a restart takes place.

Within the 2PC processing sequence there is a period of time, known as the *in-doubt* window, during which one system is unable to complete processing because it does not know what the other system has done. The distributed UOW is said to be in-doubt when:

- ▶ A participant Protocol service has replied **Prepared** in response to a **Prepare** notification, *and*
- ▶ Has written a log record of its response to signify that it has entered the in-doubt state, *and*
- ▶ Does not yet know the decision of its coordinator (to **Commit** or to **Rollback**).

Barring system or network failures, the UOW remains in-doubt until the coordinator issues either the **Commit** or **Rollback** request as a result of responses received from all UOW participants. If a failure occurs that causes loss of connectivity between a participant and its coordinator, the UOW remains in-doubt until either:

- ▶ Recovery from the failure has taken place and synchronization can resume, *or*
- ▶ The in-doubt waiting period is terminated by some built-in control mechanism, and an arbitrary (heuristic) decision is then taken (to commit or back out).

Note that while the UOW remains in-doubt, the recoverable resources that it owns remain locked.

If a system or network failure occurs during the in-doubt window, additional steps must be taken to ensure that the updates are completed in a consistent manner by both systems. This is known as *resynchronization processing*.

Previous releases of CICS provided a Recovery Manager that dealt with resynchronization processing for distributed workloads that made use of *VTAM* networks or that used *MRO* connections. These releases dealt with failures during the in-doubt window in one of three ways:

- ▶ Automatic heuristic decision:
You could cause CICS to make an automatic heuristic decision by specifying the *WAIT*, *WAITTIME*, and *ACTION* attributes on a *TRANSACTION* definition as follows:
 - If you set the *WAIT* attribute to *NO*, then CICS took whatever action was specified on the *ACTION* attribute (either *COMMIT* or *BACKOUT*) *immediately*.
 - If you set the *WAIT* attribute to *YES* and the *WAITTIME* attribute to a non-zero value, then CICS took whatever action was specified on the *ACTION* attribute after waiting for the amount of time specified in *WAITTIME* (assuming normal recovery and resynchronization had not already taken place).
- ▶ Manual heuristic decision:
You could force an in-doubt UOW to complete by issuing a *CEMT SET UOW(uowid) [COMMIT | BACKOUT]* command or its *EXEC CICS* equivalent.
- ▶ Automatic resynchronization:
If you set the *WAIT* attribute to *YES* and the *WAITTIME* attribute to *00.00.00*, the transaction would wait until it could communicate with its partner system, after which it could either explicitly request that the message it was waiting for be sent again, or it could resend the last message that it generated.

CICS TS V3.1 extended the Recovery Manager for use by *WS-AT* workloads. CICS applications that form part of a *WS-AT* workload can be controlled by any of these mechanisms. However, automatic resynchronization is somewhat different for *WS-AT* workloads.

The principle difference arises from the fact that WS-AT processing takes place over a TCP/IP network.

- ▶ Other forms of distributed transactions make use of communication mechanisms such as VTAM, and resynchronization across a VTAM network can be triggered when the connection between a pair of systems is re-established.
- ▶ CICS does not currently support TCP/IP connections in the same way that it does its VTAM connections, and so CICS can only drive resynchronization of WS-AT requests when a region starts.

Note: APARs PK56777 (CICS TS V3.2) and PK60532 (CICS TS V3.1) provided the capability to perform resynchronization processing while CICS is still active. This is achieved by running the CPIA transaction.

During any type of startup except an initial start, CICS reads the system log to discover any units of work that were in-doubt when the region previously shut down or failed. While reading through the log, CICS might find that it has outstanding units of work that indicate they were involved in an atomic transaction. These log records also indicate whether the UOW was acting as a coordinator or a participant.

▶ **Coordinator:**

If it is a coordinator and the log record indicates that the UOW was waiting for a **Committed** or **Aborted** response from a participant when CICS shut down, then the UOW is reactivated (unshunted) and sends out its decision message (**Commit** or **Rollback**) to the participant identified in the log record.

- If a response is received, then the UOW completes its processing and terminates.
- If a response is not received before the coordination UOW times out, then CICS shunts the UOW (moves it aside for processing later on). The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete. (The coordination UOW times out after 30 seconds, a value set internally by CICS).

▶ **Participant:**

If it is a participant and the log record indicates that the UOW had voted in response to a **Prepare** message and was waiting for a **Commit** or **Rollback** decision from its coordinator when communication was lost, then the UOW is reactivated, sends a **Replay** message to its coordinator, and once again waits for the decision message to arrive.

- If the decision message is then received, the participant acts on it and sends a **Committed** or **Aborted** message back to the coordinator before terminating.
- If the decision message does not arrive before the participant UOW times out, then CICS shunts the participant UOW. The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete.



Enabling atomic transactions

We begin this chapter by showing you how to enable atomic transactions in CICS. We start with the simple case where we have a service requester application running in CICS AOR1 invoking a service provider application running in CICS AOR2. Then we move to the more elaborate case where two CICSplexes are working together, one acting as service requester and the other as service provider. We conclude the chapter by showing you how to enable atomic transactions in WebSphere Application Server.

8.1 Enabling atomic transactions in CICS

We recognize that it is not likely for many customers to choose to use WS-AT for workloads distributed entirely within CICS. More typically, CICS would participate in an atomic transaction as a Web service provider with the service requester running under the control of another product such as WebSphere Application Server V6 or later. Customers might also use a CICS transaction acting as a Web service requester to participate in an atomic transaction with a service provider running under the control of another product.

Nevertheless, we start with the special case of one CICS region acting as a service requester participating in an atomic transaction with another CICS region acting as a service provider. This illustrates the capacity of CICS to undertake both the role of a coordinator of an atomic transaction and the role of a participant in an atomic transaction.

Important: If you intend to use WS-AT in CICS, then ensure that you have APAR PK56777 (CICS TS V3.2) or PK60532 (CICS TS V3.1) on your system. These APARs provide significant enhancements to the internal WS-AT processing in CICS.

8.1.1 CICS to CICS configuration

Figure 8-1 shows two CICS regions: AOR1 and AOR2. A service requester application running in AOR1 invokes a service provider application running in AOR2.

In AOR1 the request passes through a pipeline that contains a CICS-provided message handler module (either DFHPISN1, if SOAP 1.1 is being used; or DFHPISN2, if SOAP 1.2 is being used). DFHPISNx invokes the CICS-provided header processing program DFHWSATH. DFHWSATH adds a SOAP header containing a `CoordinationContext` to each message that it sends out.

In AOR2 the request passes through a pipeline that supports the Web service that AOR1's application is calling and also invokes DFHPISNx. DFHPISNx invokes the header processing program DFHWSATH when it detects a SOAP header that contains a `CoordinationContext`.

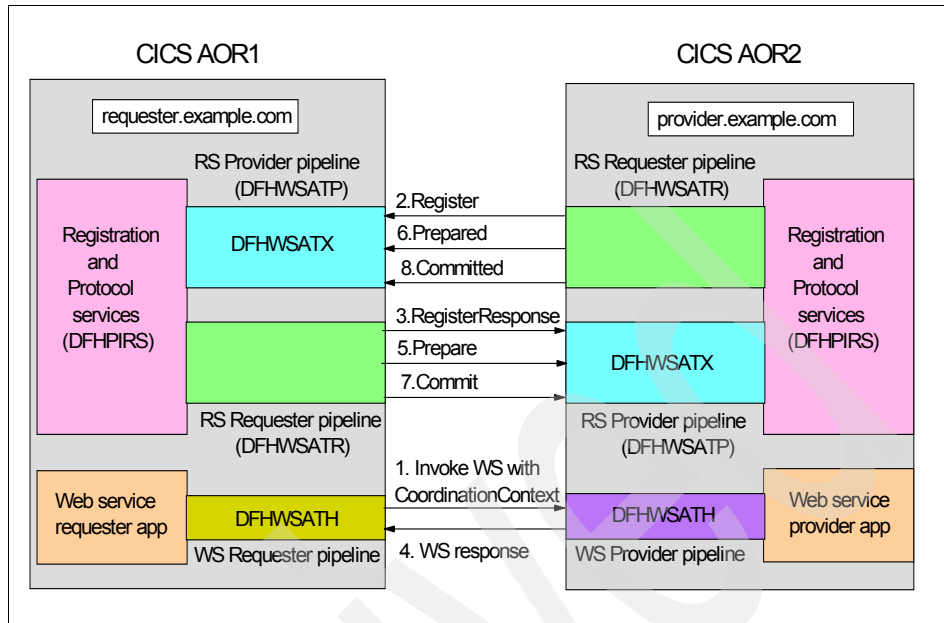


Figure 8-1 The special case: CICS to CICS

Both regions have a requester pipeline named DFHWSATR and a provider pipeline named DFHWSATP for registration and protocol processing. The DFHWSATP pipeline invokes the CICS-supplied message handler DFHWSATX as the last message handler in the pipeline.

For the workload we have shown, AOR1's DFHWSATP pipeline receives registration requests and protocol notifications, while its DFHWSATR pipeline sends registration responses and protocol instructions.

AOR2's DFHWSATP pipeline receives registration responses and protocol instructions, while its DFHWSATR pipeline sends registration requests and protocol notifications.

Note: The DFHWSATP pipeline acts as the registration endpoint for CICS.

The resource group DFHWSAT has been provided to assist customers with setting up WS-AT support in CICS. The DFHWSAT group contains the resources shown in Table 8-1.

Table 8-1 CICS supplied resource definitions for WS-AT

Resource	Resource name	Description
Pipeline	DFHWSATP	Registration services provider PIPELINE
Pipeline	DFHWSATR	Registration services requester PIPELINE
Urimap	DFHRSURI	URIMAP used by the Registration services provider
Program	DFHPIRS	Registration and protocol services program
Program	DFHWSATH	SOAP header processing program
Program	DFHWSATR	Registration and coordination services handler program
Program	DFHWSATX	CICS message handler program

Since DFHLIST does not include the DFHWSAT group and you cannot add the DFHWSAT group to DFHLIST, specifying DFHLIST in the system initialization table GRPLIST parameter does not cause CICS to install DFHWSAT automatically during an initial start.

Figure 8-2 shows the definition of the DFHWSATP PIPELINE resource.

```
OBJECT CHARACTERISTICS                                CICS RELEASE = 0650
CEDA View Pipeline( DFHWSATP )
  PiPeline      : DFHWSATP
  Group        : DFHWSAT
  Description   :
  SStatus      : Enabled           Enabled | Disabled
  Configfile   : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserv
  (Mixed Case) : icePROV.xml
               :
               :
  SHelf        : /var/cicsts/
  (Mixed Case) :
               :
               :
  Wsdir        :
  (Mixed Case) :
               :
               :
                                     SYSID=T3C1 APPLID=A6POT3C1
```

Figure 8-2 Definition of the DFHWSATP PIPELINE resource

Note that the definition contains the fully-qualified name of the pipeline configuration file. If you install the CICS-supplied registrationservicePROV.xml configuration file in a different directory when you install CICS (as we did), then you must make a copy of the entire DFHWSAT group, change the definition of the DFHWSATP pipeline, and install the modified group.

Tip: If you attempt to modify the definition of the DFHWSATP pipeline in group DFHWSAT, you get the message, Unable to perform operation: DFHWSAT is IBM protected.

If you add the DFHWSAT group to your startup list and then attempt to override the definition of DFHWSATP that is provided in the DFHWSAT group by adding a modified DFHWSATP definition that appears later in the startup list, then you get the message, DFHAM4892 W, indicating that the install of the second group completed with errors.

If you copy only the DFHWSATP, DFHWSATR, and DFHRSURI definitions to a new group and try to let program autoinstall automatically install the definitions of the programs DFHWSATH, DFHWSATR, DFHWSATX, and DFHPIRS, you might also have problems. These four programs require access to containers that use CICS-key storage, and therefore they must run with EXECKEY(CICS) unless storage protection is turned off. You would have a problem if the model for the program you use for program autoinstall does not specify EXECKEY(CICS).

Example 8-1 shows the contents of the registrationservicePROV.xml file. This configuration file defines a pipeline that contains only one message handler program, DFHWSATX.

Example 8-1 The registrationservicePROV.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```

Figure 8-3 shows the definition of the DFHWSATR PIPELINE resource.

```
OBJECT CHARACTERISTICS                                CICS RELEASE = 0650
CEDA View Pipeline( DFHWSATR )
Pipeline       : DFHWSATR
Group         : DFHWSAT
Description    :
Status        : Enabled           Enabled | Disabled
Configfile    : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserv
(Mixed Case)  : iceREQ.xml
              :
              :
              :
SHeLF         : /var/cicsts/
(Mixed Case)  :
              :
              :
              :
WsdIr         :
(Mixed Case)  :
              :
```

SYSID=T3C1 APPLID=A6POT3C1

Figure 8-3 Definition of the DFHWSATR PIPELINE resource

Example 8-2 shows the contents of the registrationserviceREQ.xml file. This configuration file defines a pipeline that does not contain any message handlers.

Example 8-2 The registrationserviceREQ.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  requester.xsd">
</requester_pipeline>
```

Recommendation: Do not modify the registrationserviceREQ.xml pipeline configuration file or the registrationservicePROV.xml pipeline configuration file. If you modify one of these files, you might inadvertently alter the flow of the registration or protocol service messages and thereby affect the integrity of your data.

Figure 8-4 shows the definition of the DFHRSURI URIMAP resource.

```
OBJECT CHARACTERISTICS                                CICS RELEASE = 0650
CEDA View Urimap( DFHRSURI )
  Urimap      : DFHRSURI
  Group      : DFHWSAT
  Description :
  Status     : Enabled           Enabled | Disabled
  USAge     : Pipeline         Server | Client | Pipeline
UNIVERSAL RESOURCE IDENTIFIER
  Scheme     : HTTP           HTTP | HTTPS
  HOST      : *
  (Lower Case) :
  PAth     : /cicswsat/RegistrationService
  (Mixed Case) :
  :
  :
  :
ASSOCIATED CICS RESOURCES
  TCpipservice :
  Analyzer     : No           No | Yes
  COnverter    :
  TRansaction  : CPIH
  PRogram     :
  PIpeline  : DFHWSATP
  Webservice  : (Mixed Case)
SECURITY ATTRIBUTES
  USErid      :
  CIphers     :
```

Figure 8-4 Definition of the DFHRSURI URIMAP resource

When the CICS-supplied CWXN transaction finds that the URI in an HTTP request matches the PATH attribute of the DFHRSURI URIMAP definition, it uses the PIPELINE attribute of that definition to get the name of the PIPELINE definition that it uses to process the incoming request. As Figure 8-4 shows, this is the DFHWSATP pipeline. As we have already seen, the DFHWSATP PIPELINE definition specifies that CICS should use registrationservicePROV.xml as the pipeline configuration file. As we have also already seen, this configuration file defines a pipeline that contains only one message handler program, DFHWSATX. Thus when the URI in an HTTP request contains /cicswsat/RegistrationService, CICS invokes the DFHWSATX message handler.

This raises the question: What causes CICS to receive an HTTP request whose URI field contains /cicswsat/RegistrationService?

The answer is that the pipeline configuration file for the service requester application (running in CICS AOR1) must specify:

- ▶ One of the CICS-provided SOAP message handlers (cics_soap_1.1_handler or cics_soap_1.2_handler)
- ▶ The mandatory invocation of the DFHWSATH header processing program to add a CoordinationContext header to the SOAP request
- ▶ A <registration_service_endpoint> element within a <service_parameter_list>

This is shown in Example 8-3.

Example 8-3 Service requester pipeline configuration file which supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
            http://schemas.xmlsoap.org/ws/2004/10/wscoord
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
  <service_parameter_list>
    <registration_service_endpoint>
      http://requester.example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</requester_pipeline>
```

The <registration_service_endpoint> element contains the address of the Registration service endpoint that runs in the requesting CICS region (AOR1). The path component of this address matches the PATH attribute defined in the DFHRSURI URIMAP resource definition of AOR1. Participant Web services should send Register requests and Prepared and Committed (or Aborted) notifications to this address.

In the service requester pipeline in Example 8-3:

- ▶ Since the `<mandatory>` element contains `True`, the pipeline flows a `CoordinationContext` with the message.
- ▶ If you change the `<mandatory>` element to `False` or remove `DFHWSATH` from the pipeline, the pipeline does not flow a `CoordinationContext` with the message.

The pipeline configuration file (Example 8-4) for the service *provider* application must specify:

- ▶ One of the CICS-provided SOAP message handlers (`cics_soap_1.1_handler` or `cics_soap_1.2_handler`)
- ▶ Invocation of the `DFHWSATH` header processing program whenever the SOAP message contains a `CoordinationContext` header
- ▶ A `<registration_service_endpoint>` element within a `<service_parameter_list>`

Example 8-4 Service provider pipeline configuration file that supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
            http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
      http://provider.example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```

This time the `<registration_service_endpoint>` element contains the address of the Registration service endpoint that runs in the provider CICS region. The Coordinator should send RegisterResponse messages and Prepare and Commit (or Abort) notifications to this address.

In the service provider pipeline in Example 8-4:

- ▶ The pipeline accepts flows with a `CoordinationContext`, and such flows are be treated as part of a WS-AT transaction.
- ▶ Because the `<mandatory>` element contains `False`, the pipeline also accepts messages *without* a `CoordinationContext` but they are *not* part of any WS-AT transaction.
- ▶ If you change the `<mandatory>` element to `True`, the pipeline *requires* that a `CoordinationContext` flow with the message. A fault is raised if a requester attempts to use the service without a `CoordinationContext`.
- ▶ If you remove `DFHWSATH` from the pipeline, the pipeline raises a `mustUnderstand` fault when a `CoordinationContext` arrives with `mustUnderstand` set to `True`.

DFHPIDIR

In addition to the above group definitions, a pipeline directory file must be defined and installed. The purpose of this file is to manage the mapping between transaction contexts and CICS tasks. This file can be a local VSAM file, shared VSAM file using RLS, or a shared file residing in the coupling facility. The file must be defined with a name of `DFHPIDIR`. Sample resource definitions for the different types of file are provided in groups `DFHPIVS`, `DFHPIVR`, and `DFHPICF`. The same file must be used by all CICS systems that provide a WS-AT capable Web Service provider.

Example 8-5 provides a suitable VSAM file definition for `DFHPIDIR`.

Example 8-5 Sample VSAM definition for DFHPIDIR

```
DEFINE CLUSTER(NAME(@dsindex@.CICS@regname@.DFHPIDIR)-
  INDEXED-
  LOG(UNDO)-                               1
  CYL(2 1)-
  VOLUME(@dsvol@)-                          2
  RECORDSIZE( 1017 1017 )-                  3
  KEYS( 16 0 )-
  FREESPACE ( 10 10 )-
  SHAREOPTIONS( 2 3 ))-
DATA (NAME(@dsindex@.CICS@regname@.DFHPIDIR.DATA) -
  CONTROLINTERVALSIZE(1024)) -
INDEX (NAME(@dsindex@.CICS@regname@.DFHPIDIR.INDEX))
```

Important: DFHPIDIR is only required if APAR PK56777 (CICS TS V3.2) or PK60532 (CICS TS V3.1) is on your system. The file is not used prior to those APARs and the resource definitions do not exist.

We conclude this section with a few remarks on the function provided by two programs in the CICS-supplied DFHWSAT group.

DFHPIRS

DFHPIRS is the central component of CICS support for WS-AT. It provides the bulk of the function that is required for registration and protocol processing. It can be called to service the following actions: Register, RegisterResponse, Prepare, Prepared, Aborted, ReadOnly, Commit, Rollback, Committed, and Replay.

If DFHPIRS encounters an unrecoverable error, it abends with an abend code of APIO.

DFHWSATH

When included in the configuration file for a Web service *requester* pipeline, DFHWSATH controls the processing that causes a `CoordinationContext` to be created and added to the SOAP message before it is sent. To do this, DFHWSATH calls DFHPIAT, which finds the local netname, the unit-of-work ID (UOWID), and the value of the `DTIMOUT` attribute on the `TRANSACTION` definition for the transaction ID under which the requesting application is running. DFHPIAT also finds the endpoint address of the Registration service for the requesting region. From all of this information, DFHPIAT creates a `CoordinationContext`.

When included in the configuration file for a Web service *provider* pipeline, this program is invoked if the CICS-supplied SOAP message handler detects a `CoordinationContext` header in a message. When called, DFHWSATH controls the processing that extracts data from the `CoordinationContext` header.

If DFHWSATH encounters an unrecoverable error, it abends with an APIP abend, which causes the pipeline manager to terminate the processing of the current request.

8.1.2 More elaborate CICS to CICS configuration

In the simple CICS to CICS configuration shown in 8.1.1, “CICS to CICS configuration” on page 236, all the requester registration endpoints reside in the same region as the service requester application, and all the provider registration endpoints reside in the same region as the service provider application. This does not have to be the case, and a number of different configurations are possible, including the one shown in Figure 8-5.

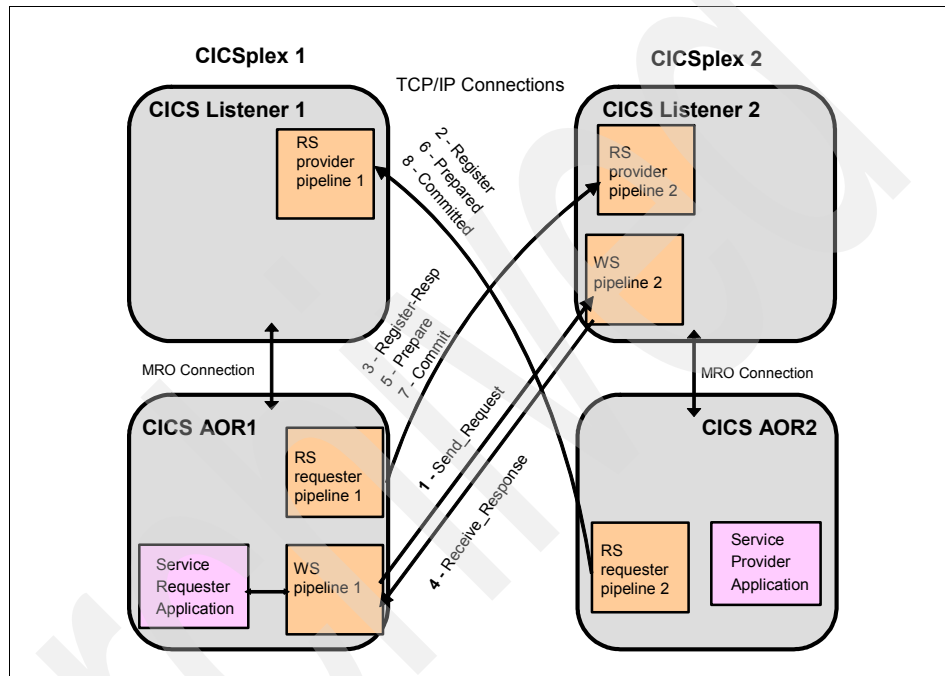


Figure 8-5 More elaborate configuration

The service requester pipeline (WS pipeline 1) must run in the same region as the service requester application that it supports. However, the service provider pipeline (WS pipeline 2) does not have to run in the same region as the service provider application that it manages. Instead, these applications are eligible for dynamic routing as described in 3.4, “Configuring for high availability” on page 113. Similarly, Registration services provider pipelines, such as RS provider pipeline 1 and RS provider pipeline 2 in Figure 8-5, do not have to be located in the same region as the applications that they service.

In the case of RS provider pipeline 1, the requests arriving there are for registration with the service requester application running in AOR1, and protocol response messages that refer to this application's role in coordinating the atomic transaction. The RS provider pipeline 2 receives registration response messages and protocol notification messages intended for the service provider application running in AOR2.

Because eventually these messages must be processed in the same region as the service provider application, the WS-AT implementation in CICS TS makes use of *RequestStreams* to pass these messages from one region to another. RequestStreams are dependent on MRO connections when they exchange information between CICS regions.

Note: If a Registration Service provider pipeline is configured in a different region than the application that it services, then an MRO connection is required between the CICS regions.

The use of RequestStreams means that individual Registration services provider pipelines can be used by different applications, and those applications do not have to reside in the same region. A set of cloned AORs can share a common Registration services provider pipeline that is located in a specific WS-AT CICS region. Work can then be distributed across the AORs and each of the WS-AT messages, arriving at the Registration services provider pipeline, contains sufficient information to allow it to be forwarded to the correct AOR.

Tip: A set of cloned AORs can share a common RS provider pipeline located in a specific WS-AT region, thus reducing the number of pipeline configurations required. In this configuration, it is recommended that you do not run applications in the specific WS-AT region and that you create clones for improved failover and availability.

Figure 8-5 on page 247 shows that the Registration services requester pipelines are in the same region as the applications that they service. The RS requester pipeline 1 services the service requester application in AOR1, and the RS requester pipeline 2 services the service provider application in AOR2. This configuration cannot be changed because CICS requires that each WS-AT message must be dispatched from the same region as the application that it relates to. This means that each AOR in a cloned set of AORs used to support a common Web services workload, must have its own Registration services requester pipeline configured.

Note: A Registration services requester pipeline must be configured in each AOR that hosts applications which participate in atomic transactions.

8.2 Enabling atomic transactions in WebSphere

WebSphere Application Server Version 6 implements the WS-AT specification. A J2EE application programmer demarcates a global transaction in a program by using the Java Transaction API (JTA) `UserTransaction` interface as shown in Example 8-6.

Example 8-6 Demarcating a global transaction using the JTA

```
UserTransaction userTransaction = null;
try {
    InitialContext context = new InitialContext();
    userTransaction = (UserTransaction)
        context.lookup("java:comp/UserTransaction");
    userTransaction.begin();

    // insert record into database
    .
    .
    .
    // commit
    userTransaction.commit();
} catch (java.rmi.RemoteException re) {
    try {
        userTransaction.rollback();
    }
    .....
}
```

If a Web service request is made by an application component running under a global transaction, WebSphere Application Server implicitly propagates a `CoordinationContext` to the target Web service if the appropriate application deployment descriptors have been specified.

If WebSphere Application Server is the system hosting the target endpoint for a Web service request that contains a WS-AT `CoordinationContext`, WebSphere automatically establishes a subordinate JTA transaction in the target run-time environment that becomes the transactional context under which the target Web service application executes.

Application developers do not have to explicitly register WS-AT participants. The WebSphere Application Server run time takes responsibility for the registration of WS-AT participants. At transaction completion time, all WS-AT participants are atomically coordinated by the WebSphere Application Server transaction service.

There are no specific development tasks required for Web service applications to take advantage of WS-AT; however, there are some application deployment descriptors that have to be set appropriately:

- ▶ In a Web module that invokes a Web service, specify `Send Web Services Atomic Transactions` on requests to propagate the transaction to the target Web service.

See “Change the deployment descriptor” on page 266 for information about how we enabled our Web application to send WS-AT SOAP headers in requests to a CICS service provider application.

- ▶ In a Web module that implements a Web service, specify `Execute using Web Services Atomic Transaction` on incoming requests to run under a received client transaction context.
- ▶ In an EJB module that invokes a Web service, specify `Use Web Services Atomic Transaction` to propagate the EJB transaction to the target Web service.
- ▶ In an EJB module, bean methods must be specified with transaction type `Required`, which is the default, to participate in a global atomic transaction.

Transaction scenarios

In this chapter we show different scenarios that demonstrate how you can synchronize resource updates using the WS-Atomic Transaction support in CICS and WebSphere Application Server.

We start with an explanation of the scenarios that we set out to test, and how we prepared the system and the settings that we used for the configuration of our system.

We then cover in detail the changes that we made to the CICS pipeline configuration and WebSphere Application Server to enable Web services transactional integration.

9.1 Introduction to our scenarios

Figure 9-1 shows three different atomic transaction scenarios that we considered testing.

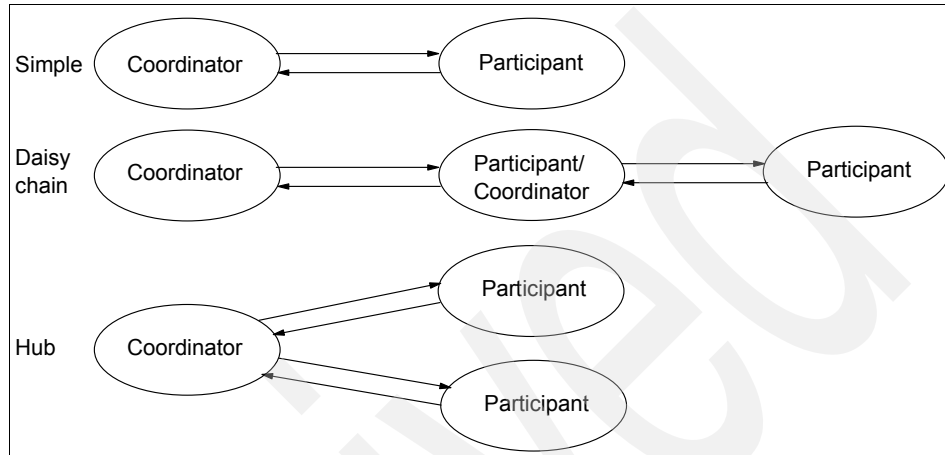


Figure 9-1 Three possible atomic transaction scenarios

The scenario at the top is the simple case where a coordinator controls a single participant. Both parties can make recoverable updates. Another possibility is that only the participant makes recoverable updates. A third possibility is that the participant does nothing recoverable; in this case it sends a `ReadOnly` notification when it is coordinated.

The second scenario is often referred to as a *daisy chain*. Here there is a primary (or root) coordinator that invokes a Web service, and the invoked Web service then invokes a second Web service. The entire atomic transaction is controlled by the primary coordinator. The middle system takes on the role of a coordinator *and* the role of a participant at different times in its life cycle. When the primary coordinator instructs it during transaction termination, this system acts as a participant. However, before it responds to the primary coordinator, it then takes on the role of coordinator of its own participant.

The third scenario is a *hub* configuration. A single coordinator invokes one Web service and then another. It then coordinates them together.

You can probably think of many other scenarios. In this chapter we more closely look at two of these scenarios:

- ▶ The simple scenario, in which WebSphere Application Server V6.0 is the coordinator and CICS TS V3.1 is the participant. See “The simple atomic transaction scenario” on page 255.
- ▶ The daisy chain scenario, in which CICS TS V3.1 is both a coordinator and a participant. See 9.3, “The daisy chain atomic transaction scenario” on page 298.

9.1.1 Software checklist

Table 9-1 shows the software we used in the scenarios described in this chapter.

Table 9-1 Software used in the atomic transaction scenarios

Windows	z/OS
Windows XP SP4	z/OS V1.9
IBM WebSphere Application Server - ND V6.0.0.2	CICS Transaction Server V3
Internet Explorer V6.0	
DB2 V8.1.7.445	
User-supplied programs: <ul style="list-style-type: none"> ▶ CatalogAtomic.ear A modified service requester application used for the WS-AT scenarios ▶ DispatchAtomic.ear A modified service provider application used for the WS-AT scenarios 	User-supplied programs: <ul style="list-style-type: none"> ▶ SNIFFER (message handler program written in COBOL) ▶ WSATHND (header processing program written in C)

Important: You should install the fix for APAR PK16509 if you are using a later version of WebSphere than IBM WebSphere Application Server - Network Deployment V6.0.0.2.

9.1.2 Definition checklist

Table 9-2 shows the definitions we used in the scenarios described in this chapter.

Table 9-2 Definitions used in the atomic transaction scenarios

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	cam21-pc11.mop.ibm.com
IP address	9.100.193.167	9.100.199.238
Job name	CIWST3C1	
APPLID	A6POT3C1	
SIT parameter	SEC=NO (see APAR PK10849)	
TCPIP SERVICE definition	T3C1	
PORT attribute on T3C1 TCPIP SERVICE definition	15301	
FILE definition for sample catalog VSAM file	EXMPCAT	
RECOVERY attribute on EXMPCAT FILE definition	BACKOUTONLY	
Web service provider PIPELINE definition	PIPE1	
CONFIGFILE attribute on PIPE1 PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_soap11 provider.xml	
Web service requester PIPELINE definition	PIPE2	
CONFIGFILE attribute on PIPE2 PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_soap11 requester.xml	
RDO group containing copy of DFHWSAT group	CTS310C	
Registration Service provider PIPELINE	DFHWSATP	
CONFIGFILE attribute on DFHWSATP PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_registr ationservicePROV.xml	
Registration Service requester PIPELINE	DFHWSATR	

Value	CICS TS	WebSphere Application Server
CONFIGFILE attribute on DFHWSATR PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_registrationserviceREQ.xml	

Important: At the time of running this scenario, APAR PK10849 was open. It reports that if you attempt to use WS-AT in a CICS region that is running with the SIT parameter SEC=YES, then during the RegisterResponse step of coordination RACF® issues messages ICH408I and IRR012I and CICS issues message DFHPI0002. Therefore, we ran our region with SEC=NO.

9.2 The simple atomic transaction scenario

The sample Catalog application provided with CICS TS V3.1 provides three Web services:

- ▶ inquireSingle
- ▶ inquireCatalog
- ▶ placeOrder

Only the placeOrder Web service *updates* the VSAM file that contains the information about the company's products. Therefore, we naturally decided that the service requester running in WebSphere Application Server should invoke the placeOrder Web service in our WS-AT scenario.

We modified the service requester application (see "Installing the service requester" on page 99) to create a global transaction and to update a DB2 table. We call the new service requester application *AtomicClient*.

Note: Rather than using JDBC™ to update a DB2 table, we could have chosen to use another J2EE connector such as the JCA or JMS. Updates to resources accessed by these connectors can be synchronized with Web service requests in the same way.

To be more specific, we create the table ITSO.ORDER in DB2. Before calling the placeOrder Web service, the AtomicClient inserts a row in this table so that we have a log of all of the orders placed through our application. We can now have a global transaction that updates two resources: a DB2 table in the Windows environment and a VSAM file in the z/OS environment.

Figure 9-2 shows the sequence of events in AtomicClient as it begins a global transaction, updates a DB2 database, calls the placeOrder Web service, and then either commits or rolls back the updates.

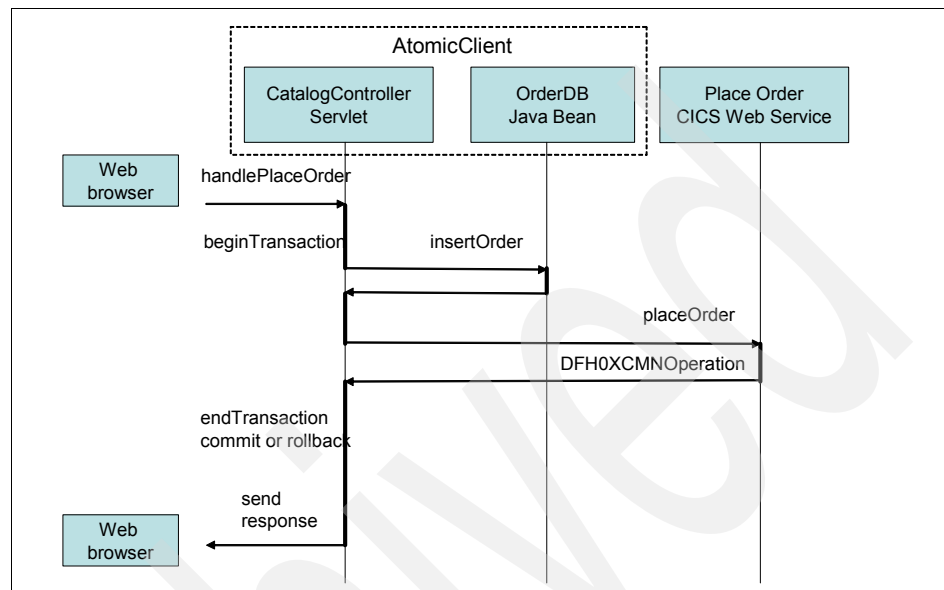


Figure 9-2 Simple atomic transaction sequence of events

Figure 9-3 shows a more global view of the sequence of events:

1. The user uses his Web browser to invoke the AtomicClient that runs in WebSphere Application Server.
2. The AtomicClient updates the ITSO.ORDER table in DB2.
3. WebSphere Application Server sends the insertOrder SOAP message containing the order to CICS.
4. CICS uses Web service provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
5. DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
6. CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
7. DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.

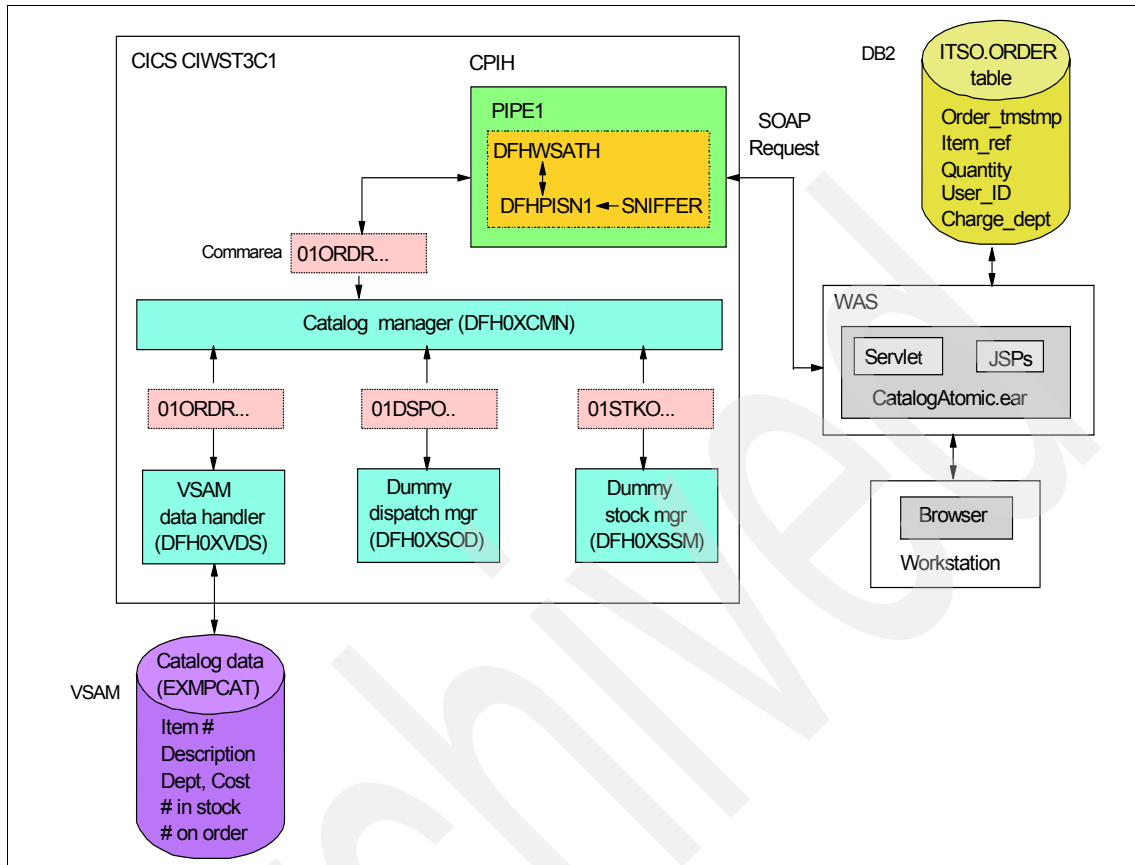


Figure 9-3 WebSphere as service requester and CICS as service provider

In the following sections we describe how we set up CICS for this scenario, how we created the AtomicClient and the ITO.ORDER table, and the results of testing this scenario.

9.2.1 Setting up CICS for the simple scenario

To set up CICS for this scenario we performed the following steps:

1. We set the value of the SEC system initialization table parameter to NO; see 9.1.2, “Definition checklist” on page 253.
2. We changed the value of the RECOVERY attribute of the EXMPCAT FILE definition to BACKOUTONLY so that the file is recoverable. (The EXMPCAT FILE definition defines the VSAM file that contains the catalog data for the sample application).

3. We edited PIPE1's configuration file:

/CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

So that it contains the XML shown in Example 9-1. This XML contains:

- The same XML as shown in Example 8-4 on page 244, except that we changed the address in the registration_service_endpoint element to be specific to the CICS region CIWST3C1
- XML to add the SNIFFER message handler

Example 9-1 PIPE1: /CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <service_handler_list>
      <handler>
        <program>SNIFFER</program>
        <handler_parameter_list/>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
            http://schemas.xmlsoap.org/ws/2004/10/wscoop
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
      http://MVS63.mop.ibm.com:15301/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```

Tip: Example 9-1 on page 258 shows the `registration_service_endpoint` element split across three lines for readability. However, during our testing we found that we had to place its start tag, contents, and end tag on the *same* line.

4. We copied the group DFHWSAT to the group CTS310C and added the group CTS310C to our startup list:

```
CEDA COPY GROUP(DFHWSAT) TO(CTS310C)
CEDA ADD GROUP(CTS310C) LIST(LISTT3C1)
```

5. We edited DFHWSATP's configuration file:

```
/CIWS/T3C1/config/ITSO_7206_wsat_registrationservicePROV.xml
```

So that it contained the XML shown in Example 9-2. This XML contains:

- The same XML as shown in Example 8-1 on page 240
- XML to add the WSATHND header processing program (see “How we monitored the exchange of WS-AT messages” on page 260)

Example 9-2 DFHWSATP: /CIWS/.../ITSO_7206_wsat_registrationservicePROV.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace>*</namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```

6. We edited DFHWSATR's configuration file:

/CIWS/T3C1/config/ITSO_7206_wsat_registrationserviceREQ.xml

So that it contained the XML shown in Example 8-2. This XML contains:

- The same XML shown in Example 8-2 on page 241
- XML to add the WSATHND header processing program

Example 9-3 DFHWSATR: /CIWS/.../ITSO_7206_wsat_registrationserviceREQ.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace>*</namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
  </service>
</requester_pipeline>
```

How we monitored the exchange of WS-AT messages

The user-written WSATHND header processing program writes messages to the CESO transient data queue, which normally is an extrapartition queue with DDname CEEOUT. We used WSATHND to monitor the exchange of registration and protocol service messages between WebSphere Application Server and CICS.

The message output of WSATHND depends on the values the programmer has assigned to the two variables MESSAGES_ON and FULL_MESSAGES_ON:

1. If MESSAGES_ON is set to 1, the program writes the following message:

WSAT: REACHED HANDLER - function

Where function is the contents of the DFHFUNCTON container (RECEIVE-REQUEST, SEND-RESPONSE, SEND-REQUEST, RECEIVE-RESPONSE, PROCESS-REQUEST, NO-RESPONSE, or HANDLER-ERROR).

When the function is NO-RESPONSE, then the handler is being invoked after processing a request, when there is no response to be processed.

2. If the function is not NO-RESPONSE, then:
 - If FULL_MESSAGES_ON is set to 1, WSATHND writes the following message:

WSAT: contents of the DFHREQUEST container

- If MESSAGES_ON is set to 1, it writes the following message:

WSAT: ACTION: action

where action has one of the following values: Register, RegisterResponse, Prepare, Prepared, Commit, Committed, ReadOnly, Abort, Aborted, Rollback.

Thus, WSATHND provides a way to see the registration and protocol service messages that are being sent between the service requester and the service provider as they happen. We found it easier to use this program than to take a TCP/IP trace or to search for these messages in a CICS auxiliary trace.

9.2.2 Creating the AtomicClient and ITSO.ORDER table

To create the AtomicClient and the ITSO.ORDER table, we did these steps:

1. Created the OrderBean JavaBean that represents the order.
2. Created the OrderDB JavaBean that inserts the order into the DB2 database.
3. Changed the handlePlaceOrder method of the CatalogController servlet so that it creates an OrderBean and then calls OrderDB to insert it into the DB2 database as part of a global transaction.
4. Changed the deployment descriptor of the Web module CatalogAtomicWeb. This is the module that invokes the Web Service.
5. Created the ITSO.ORDER table.
6. Created a data source.

We explain each step in detail in the following sections.

Create the OrderBean JavaBean

The OrderBean JavaBean represents the order; it has all the fields from the PlaceOrder JavaServer™ Page (JSP™) plus a timestamp that records the time when the Web browser user placed the order. Example 9-4 shows the code for the OrderBean JavaBean.

Example 9-4 The OrderBean JavaBean

```
package itso.mop.objects;

import java.sql.Timestamp;

public class OrderBean {

    private Timestamp orderTmstmp;
    private short itemRef;
    private short quantity;
    private String userId;
    private String chargeDept;

    // getters and setters
    ...
}
```

Create the OrderDB JavaBean

Example 9-5 shows the code for the OrderDB JavaBean, which inserts the order into the DB2 table.

Example 9-5 The OrderDB JavaBean

```
package itso.mop.db;

import itso.mop.objects.OrderBean;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class OrderDB {

    private static final String _JNDI_NAME = "jdbc/ORDERDS";

    public int insertOrder(OrderBean order){
        Connection con = getConnection();
        PreparedStatement pm = null;
        int rs = 0;
        try {
            pm = con.prepareStatement("INSERT INTO ITSO.ORDER (ORDER_TMSTMP,
ITEM_REF, QUANTITY, USER_ID, CHARGE_DEPT) VALUES(?,?,?,?,?)");
            pm.setTimestamp(1, order.getOrderTmstmp());
            pm.setInt(2, order.getItemRef());
            pm.setInt(3, order.getQuantity());
        }
    }
}
```

```

        pm.setString(4, order.getUserId());
        pm.setString(5, order.getChargeDept());

        rs = pm.executeUpdate();

        System.out.println("OrderDB.insertOrder() - inserted the order in the
database!!!!");
    } catch (SQLException e) {
        System.out.println("OrderDB.insertOrder() - Exception inserting the
order in the database!!");
        e.printStackTrace(System.err);
        rs = 0;
    } finally {
        try {
            if (pm != null)
                pm.close();
            if (con != null)
                con.close();
        } catch (Exception e) {
            e.printStackTrace(System.err);
        }
    }
    return rs;
}

private Connection getConnection() {
    Connection con = null;
    try {
        InitialContext ic = new InitialContext();
        DataSource ds = (DataSource) ic.lookup(_JNDI_NAME);
        con = ds.getConnection();
    } catch (NamingException e) {
        e.printStackTrace(System.err);
    } catch (SQLException e) {
        e.printStackTrace(System.err);
    }
    return con;
}
}

```

Change the handlePlaceOrder method

In the Catalog application, the CatalogController servlet handles all the requests from the Web browser. In particular, the `handlePlaceOrder` method of this servlet handles requests from the Web browser to place an order. We made the following changes to this method:

1. Created the OrderBean and populated all of its fields

2. Created the transaction (UserTransaction) and then began the transaction
3. Inserted the order into the DB2 database using the OrderDB JavaBean
4. Called the CICS Web service
5. Coded the method to throw a RemoteException when the "ROLLBACK" user ID enters the order (for testing transaction rollback)
6. Committed the transaction
7. Rolled back the transaction in the case of an Exception

Note: The rollback logic was added to the application for testing purposes only; normal applications would not be expected to behave in this way.

Example 9-6 shows the code for the `handlePlaceOrder` method.

Example 9-6 The `handlePlaceOrder` method of the `CatalogController` servlet

```
private void handlePlaceOrder(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
    OrderDetails orderDetails = new OrderDetails();
    try
    {
        // Retrieve the data from the submitted form
        orderDetails.setItemRefNumber(
short.parseShort(request.getParameter("itemRef")));
        orderDetails.setQuantityRequired(Short.parseShort(
            request.getParameter("quantity")));
        orderDetails.setUserId(request.getParameter("userName"));
        orderDetails.setChargeDepartment(request.getParameter("deptName"));
    }
    catch(Exception e)
    {
        handleError(request, response, "Input data format incorrect, please try
again");
        return;
    }

    UserTransaction userTransaction = null;
    try
    {
        //1. Create the order bean and populate all of its fields
        System.out.println("CatalogController.handlePlaceOrder() - creating the
order");
        OrderBean order = new OrderBean();
        order.setOrderTmstmp(new Timestamp(Calendar.getInstance()).
```



```

        getTime().getTime()));
order.setItemRef(orderDetails.getItemRefNumber());
order.setQuantity(orderDetails.getQuantityRequired());
order.setUserId(orderDetails.getUserId());
order.setChargeDept(orderDetails.getChargeDepartment());

// 2. Create and begin the Transaction
InitialContext context = new InitialContext();
userTransaction = (UserTransaction)
    context.lookup("java:comp/UserTransaction");
System.out.println("CatalogController.handlePlaceOrder() - beginning the
transaction");
userTransaction.begin();

// 3. Insert the order into the database
System.out.println("CatalogController.handlePlaceOrder() - inserting the
order in the database");
OrderDB orderDB = new OrderDB();
orderDB.insertOrder(order);

// 4. Make the Web service call to CICS
System.out.println("CatalogController.handlePlaceOrder() - calling the
CICS web service");
orderDetails.populateResponse(
getOrderProxy(request).DFHOXCMNOperation(orderDetails.getRequestProgramInterfac
e()) );
System.out.println("CatalogController.handlePlaceOrder() - response back
from the CICS web service");
// 5. Throw exception if user ID ROLLBACK
if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
{
    System.out.println("CatalogController.handlePlaceOrder() - simulating
the RemoteException");
    throw new RemoteException("Throwing the RemoteException");
}
// 6. Commit the transaction
System.out.println("CatalogController.handlePlaceOrder() - commit the
transaction");
userTransaction.commit();
System.out.println("CatalogController.handlePlaceOrder() - after
commit");
}
// 7. Rollback in case of exception
catch (Exception e)
{
    try {
        System.out.println("CatalogController.handlePlaceOrder() -
rollingback the transaction");
        userTransaction.rollback();
    }
}

```

```

        } catch (Exception e1) {
            System.out.println("CatalogController.handlePlaceOrder() - Exception
when rollingback the transaction");
            e1.printStackTrace();
        }

        if (e.getMessage().startsWith("java.net.ConnectException"))
        {
            String errorMessage = "Unable to connect to service endpoint: "+
                getOrderProxy(request).getEndpoint()+
                "<BR> Please ensure service is running.";
            handleError(request, response, errorMessage);
            return;
        }
        else
        {
            e.printStackTrace();
            handleError(request, response, "An Error ocured calling the service:
"+e.getMessage());
            return;
        }
    }
    // Call the response page setting the OrderDetails bean on the request
    RequestDispatcher dispatcher =
request.getRequestDispatcher(orderResponsePage);
    request.setAttribute("orderDetails",orderDetails);
    dispatcher.forward(request,response);
}

```

Note: Since the service requester application is a Web application (as opposed to an enterprise bean) we used a bean-managed transaction. For an enterprise bean, it is generally recommended to use container-managed transactions.

Change the deployment descriptor

The deployment descriptor of a J2EE application is used to specify whether the application component, if it makes any Web service requests, expects any transaction context to be propagated with the Web service requests (in accordance with the WebSphere WS-AT support).

To activate the WS-AT support:

1. We imported the client application archive CatalogAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (CatalogAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 9-4).

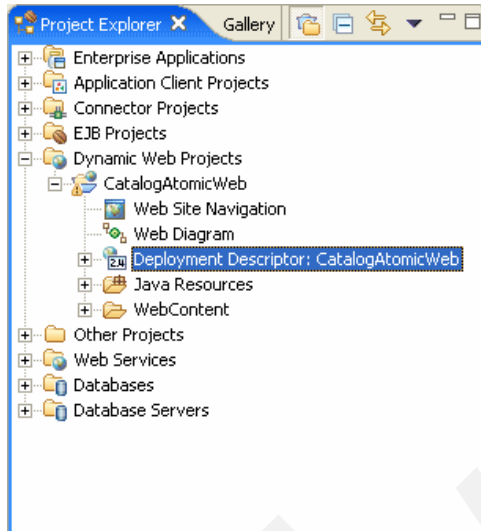


Figure 9-4 Project Explorer for CatalogAtomicWeb application

2. In the Deployment Descriptor, we clicked the **Servlets** tab and selected the **CatalogController** servlet. Then we scrolled down to Global Transaction and selected **Send Web Services Atomic Transactions on requests** (Figure 9-5).

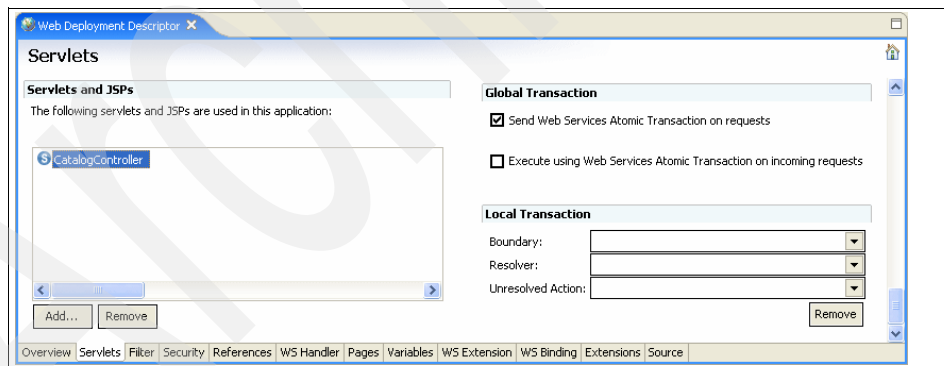


Figure 9-5 Activating atomic transaction in the Web deployment descriptor

3. We saved and closed the file.

Note: A similar deployment descriptor attribute **Use Web Services Atomic Transaction** can be used for enterprise beans.

Create the ITSO.ORDER table

We created the ITSO.ORDER table to record all the orders that the Web browser user sends to the Catalog application. Example 9-7 shows the script we used to create the database and the table.

Example 9-7 Creation of ITSOWS database and ITSO.ORDER table

```
-- IBM ITSO
CONNECT RESET;
-- Create database ITSOWS and schema ITSO
CREATE DATABASE ITSOWS;
CONNECT TO ITSOWS;
CREATE SCHEMA ITSO;

-- Table definitions for Order
CREATE TABLE ITSO.ORDER
  (ORDER_TMSTMP TIMESTAMP NOT NULL ,
   ITEM_REF INTEGER NOT NULL ,
   QUANTITY INTEGER NOT NULL ,
   USER_ID CHARACTER (20) NOT NULL ,
   CHARGE_DEPT CHARACTER (20) NOT NULL ,
   CONSTRAINT ORDERKEY PRIMARY KEY ( ORDER_TMSTMP ) );
```

The ITSO.ORDER table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table.

Figure 9-6 shows the ITSOWS database in the Control Center.

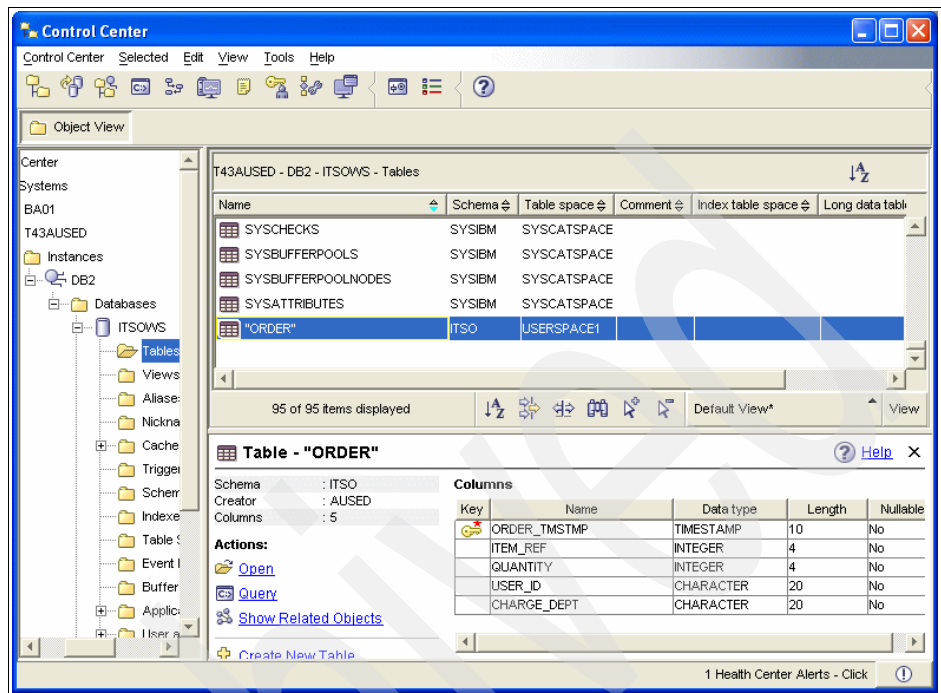


Figure 9-6 The ITSOWS database in the Control Center

Create a WebSphere data source

Next we configured the data source in WebSphere through the Administrative Console:

1. We opened the Admin console by pointing a Web browser at:
`http://cam21-pc11:9060/admin`
2. In the main window, we opened **Resources** and then clicked **JDBC Providers** (Figure 9-7).

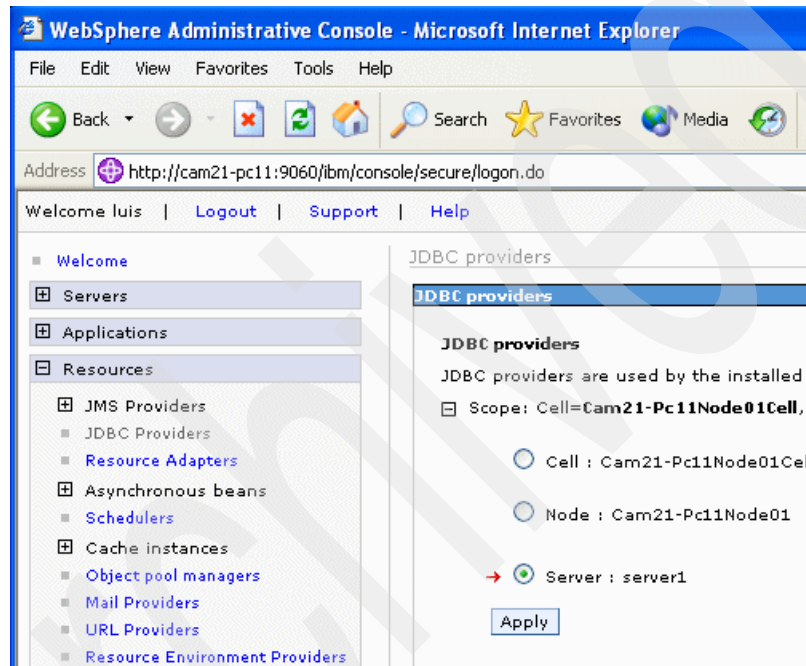


Figure 9-7 Admin console - JDBC providers

3. Because we are running a single server, we chose the Server scope.
4. To create a new JDBC provider, we clicked **New**.
A new JDBC provider window was presented (Figure 9-8).

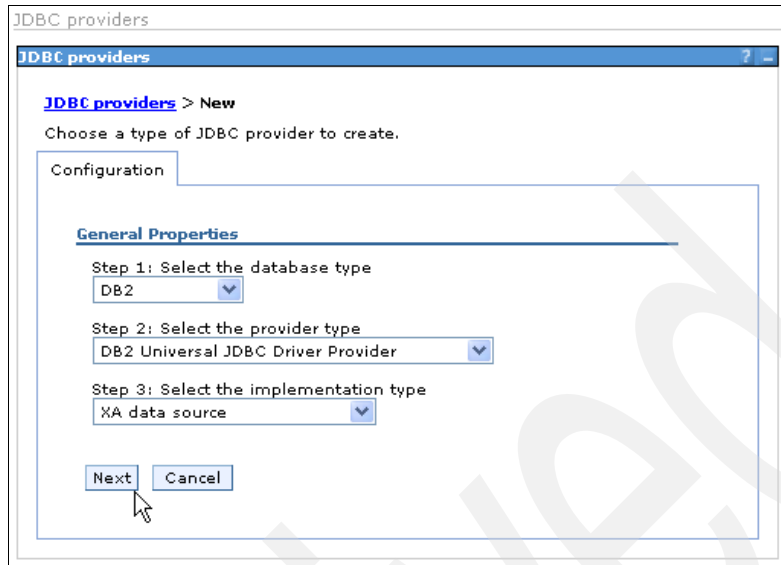


Figure 9-8 Admin console - New JDBC provider

5. In the new JDBC provider window (Figure 9-8), we selected:

- DB2 as database type
- DB2 Universal JDBC Driver Provider as provider type
- XA data source as the implementation type

We clicked **Next**, and were presented with the window shown in Figure 9-9.

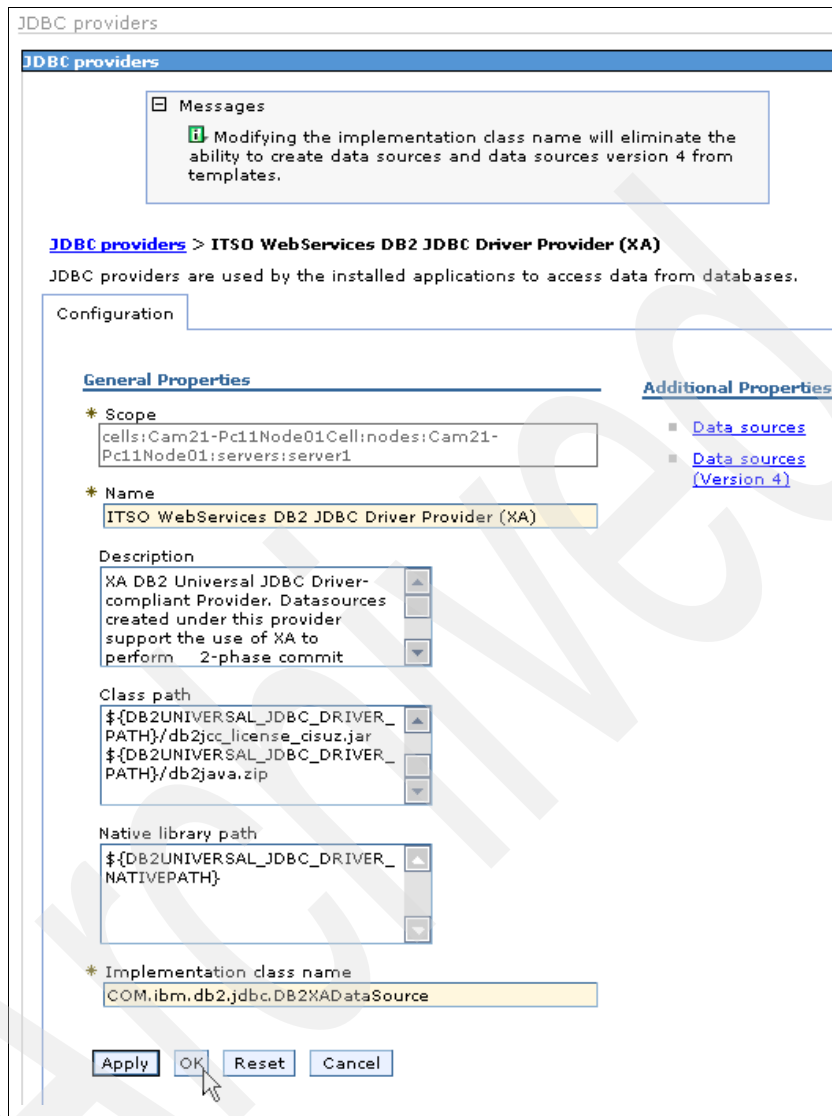


Figure 9-9 Admin console - New JDBC provider 2

- We provided a name for the JDBC provider, in this instance:
ITSO WebServices DB2 JDBC Driver Provider (XA)

7. We noted that WebSphere offers the Implementation class name `COM.ibm.db2.jdbc.DB2XADataSource`. This is the class of the DB2 driver that has two-phase commit capability. Because this class is not in the default class path provided by WebSphere, we added to the end of the class path the following statement:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2java.zip
```

8. We clicked **OK** and saved the configuration changes.

Now we have the new JDBC provider defined and we can see it in the list of JDBC providers.

Define the DB2UNIVERSAL_JDBC_DRIVER_PATH variable

We defined the `DB2UNIVERSAL_JDBC_DRIVER_PATH` WebSphere variable as follows:

1. In the Admin console main window, we clicked **Environment** and then clicked **WebSphere Variables** as shown in Figure 9-10.



Figure 9-10 Admin console - WebSphere variables window

2. We set the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH variable to the default JDBC driver location C:\Program Files\IBM\SQLLIB\java (Figure 9-11).

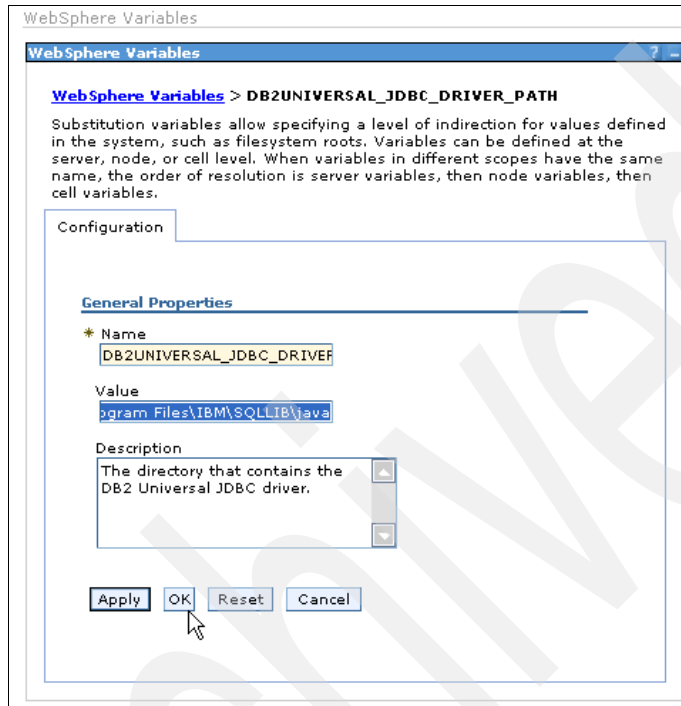


Figure 9-11 Admin console - Defining the DB2 driver path variable

Create the J2C Authentication data

Before creating the data source, we had to create the J2EE Connector Architecture (J2C) Authentication data in JAAS Configuration:

1. In the Admin console main window, we selected **Security** and then **Global Security**.
2. In the Global security page, we clicked **JAAS Configuration under Authentication**, and selected **J2C Authentication data**.
3. In the J2EE Connector Architecture (J2C) authentication data entries page, we clicked **New**.

The window shown in Figure 9-12 was displayed.



Figure 9-12 Admin console - Define J2C authentication alias

4. In the General Properties, we entered the following values:
 - DB2user as Alias
 - Administrator as User ID
 - User password as Password
5. We clicked **OK** and saved the configuration changes.

Configure the data source for the JDBC provider

Next we configured the data source for the JDBC provider:

1. In the Admin console main window, we selected **Resources** and then **JDBC Providers**.
2. In the JDBC providers window, we clicked the new provider that we defined in “Create a WebSphere data source” on page 270:
3. ITSO WebServices DB2 JDBC Driver Provider (XA)
4. In Additional Properties, we clicked **Data sources**, then clicked **New**.
We were presented with the window shown in Figure 9-13.

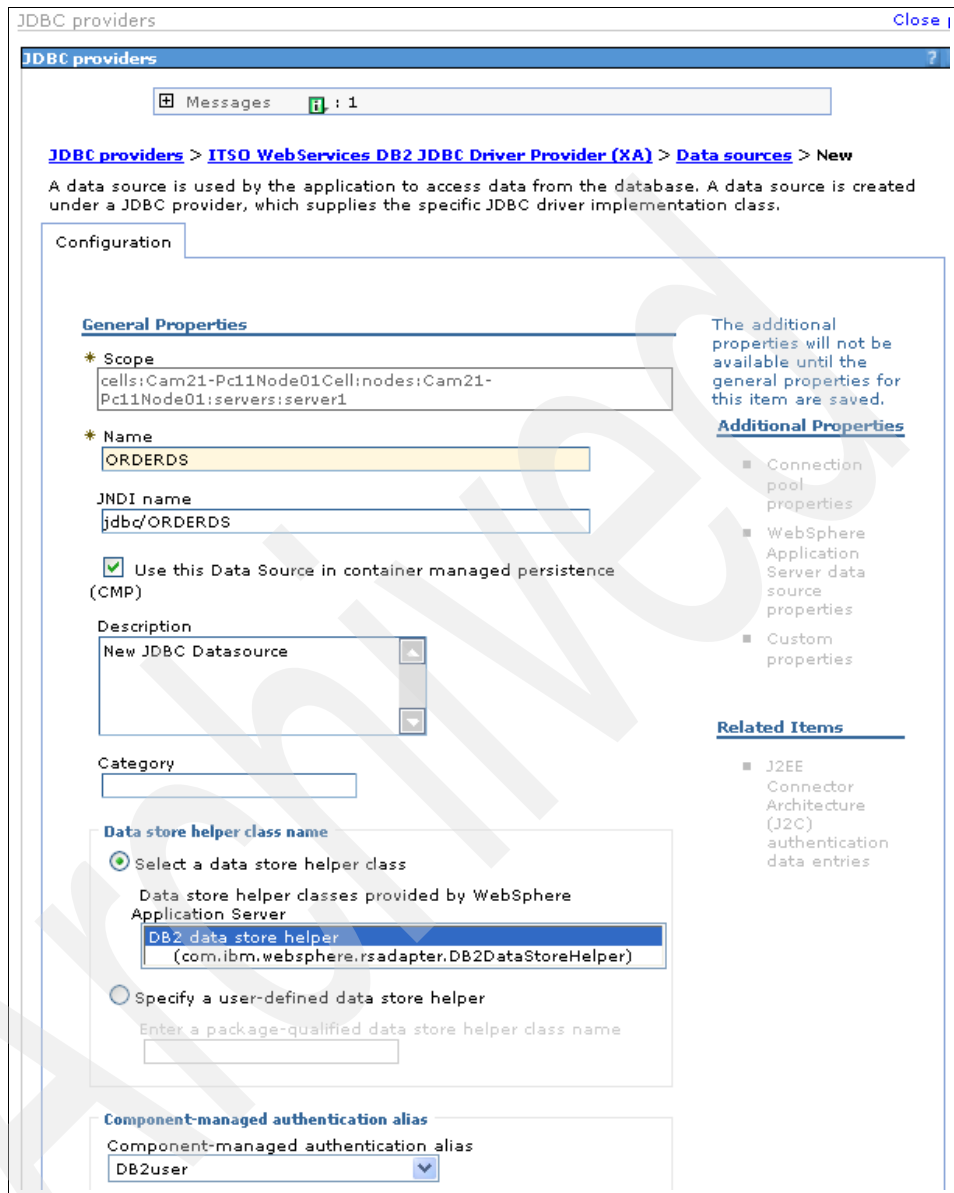


Figure 9-13 Admin console - New Data source (1 of 2)

5. In the General Properties, we entered the following values:
 - ORDERDS as the Data source name.
 - jdbc/ORDERDS as the JNDI name.
 - DB2user as the Component-managed authentication alias.
6. We then scrolled down to the bottom of the window and entered ITSOWS as the Database name (Figure 9-14).

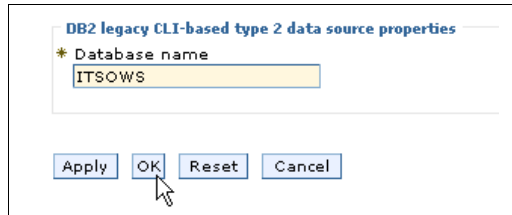


Figure 9-14 Admin console - New data source (2 of 2)

7. We clicked **OK** and then saved the changes.

9.2.3 Testing the simple scenario

We performed two tests of the simple scenario:

- ▶ Normal transaction termination (described next)
- ▶ Abnormal transaction termination (see “Simple scenario: Abnormal transaction termination” on page 295)

Simple scenario: Normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario, we followed these steps:

1. We opened the welcome window of the Catalog application using the URL:

`http://cam21-pc11:9080/CatalogAtomicWeb/Welcome.jsp`

We were presented with the window shown in Figure 9-15.

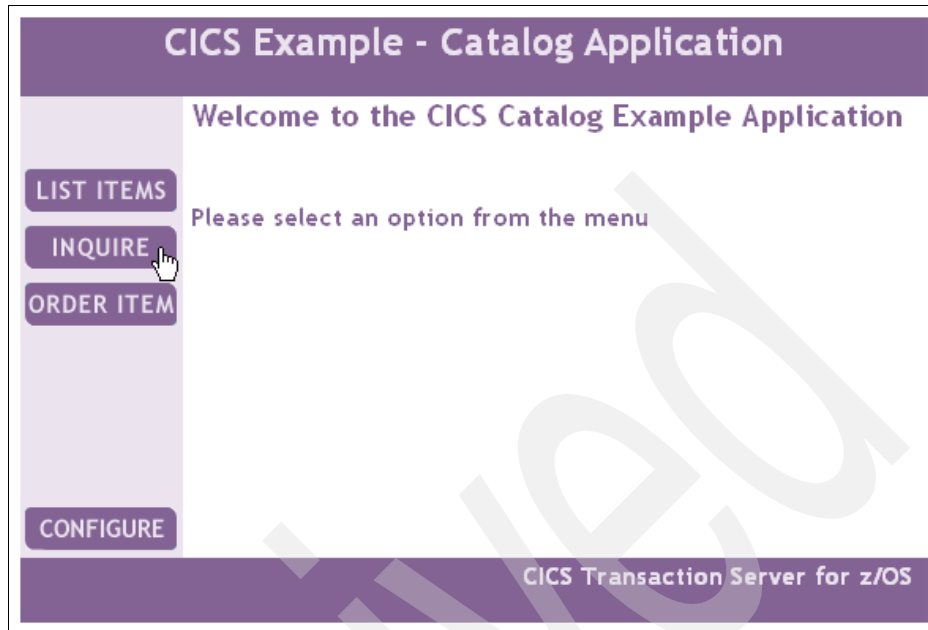


Figure 9-15 Catalog Application - Welcome window

2. We clicked **INQUIRE**.
3. In the Inquire Single window, we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 9-16.

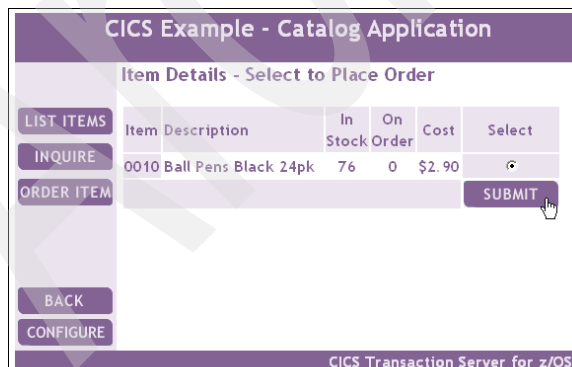


Figure 9-16 Catalog Application - Inquire single

4. We noted that the number of items in stock was 76. This value is taken from the CICS VSAM file.

5. We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 9-17.

CICS Example - Catalog Application	
Enter Order Details	
LIST ITEMS	Item Reference Number 0010
INQUIRE	Quantity 001
	User Name Luis
	Department Name D001
	SUBMIT
BACK	
CONFIGURE	

Figure 9-17 Catalog Application - Enter order details

6. In the Enter Order Details window, we provided a User Name and a Department Name and clicked **SUBMIT**.
7. After the CICS Web service processed the order, we got a response telling us that the order was successfully placed (Figure 9-18).

CICS Example - Catalog Application	
Order Placed	
LIST ITEMS	ORDER SUCESSFULLY PLACED
INQUIRE	
ORDER ITEM	
BACK	
CONFIGURE	
CICS Transaction Server for z/OS	

Figure 9-18 Catalog Application - Order placed response

- In the WebSphere server log, we see trace entries, generated by the Catalog Controller servlet, which show that the transaction was successfully committed (Example 9-8).

Example 9-8 WebSphere server log with successful Place Order

```
CatalogController.doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

- Next we checked the same item number through the Inquire Single service and verified that the stock level decreased by one item (Figure 9-19).

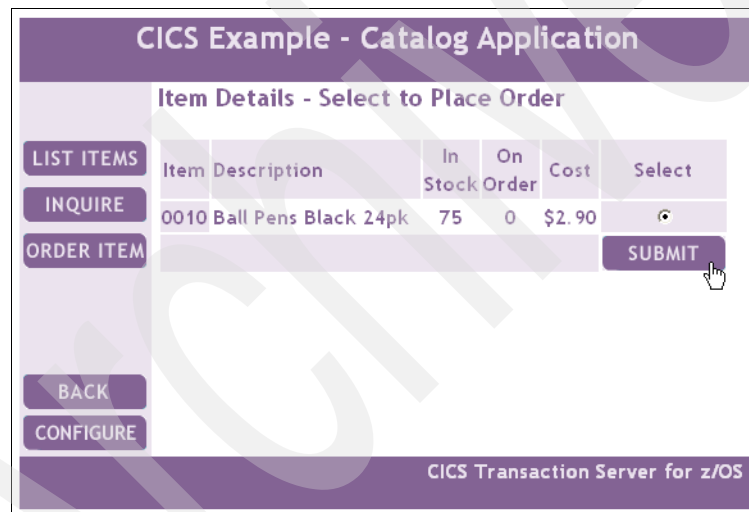


Figure 9-19 Catalog Application - Inquire single

- We opened a DB2 Control Center and issued the SQL command:

```
SELECT * FROM ITSO.ORDER
```

This lists all of the records in our ITSO.ORDER table.

Figure 9-20 shows that our new record is in the table.

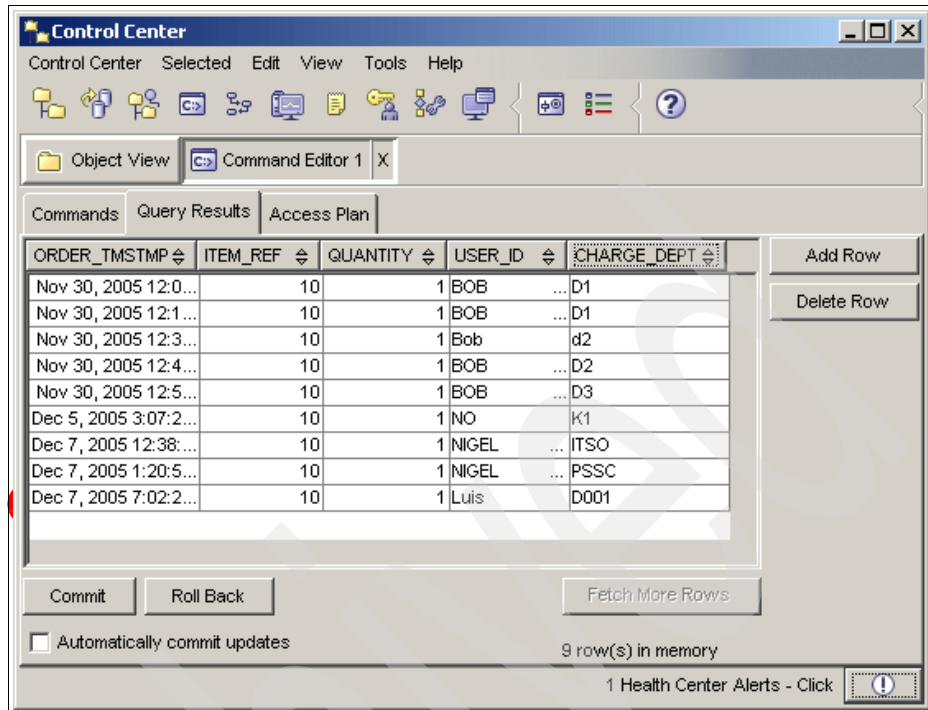


Figure 9-20 The new record in the ITSO .ORDER table

Figure 9-21 shows the registration and protocol service messages that were exchanged between CICS and WebSphere Application Server during our test. Note that WebSphere Application Server uses separate endpoint addresses for its Registration service, Protocol service, and fault messages.

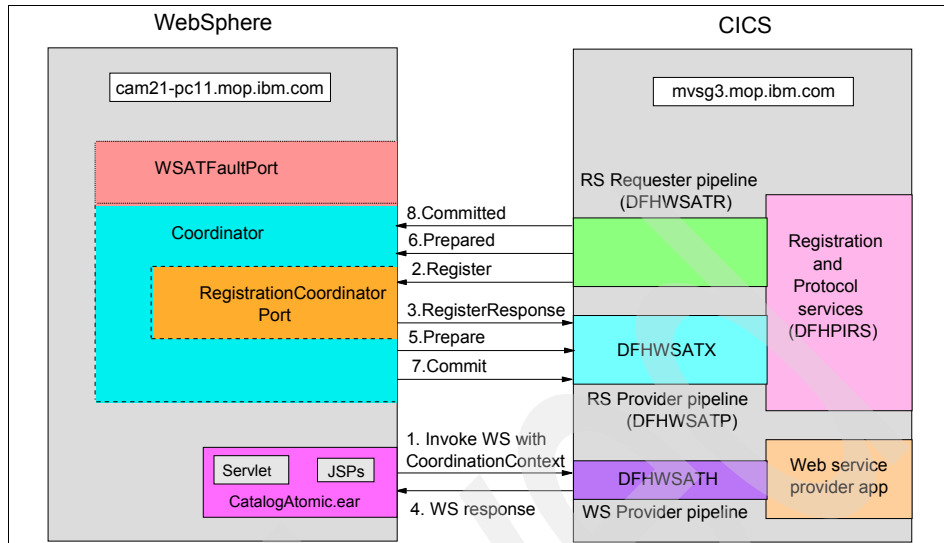


Figure 9-21 Messages exchanged during test of normal atomic transaction termination

We conclude our discussion of this test by showing the complete messages summarized by Figure 9-21. To make each message easier to understand:

- ▶ We format it.
- ▶ We replace the URL in each xmlns attribute with "..." for each namespace in Example 9-9. For instance:
We replaced xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" with xmlns:soap="..." .

Example 9-9 Namespaces used in messages between WebSphere and CICS

```

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="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"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscor="http://schemas.xmlsoap.org/ws/2004/10/wscor"
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension"
xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline"

```

Invoke Web service with coordination context (message 1)

Example 9-10 shows the SOAP 1.1 message that WebSphere sends to CICS to invoke the placeOrder Web service. The message consists of a SOAP envelope that contains a SOAP header and a SOAP body.

Example 9-10 WebSphere sends to CICS a Web service request with a CoordinationContext header

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
        xmlns:wscor="..." xmlns:wsa="...">
  <soapenv:Header>
    <wscor:CoordinationContext soapenv:mustUnderstand="1">
      <wscor:Expires>Never</wscor:Expires>
      <wscor:Identifier>
        com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
        ecc889b8eab92e29d91254fd4fb0ff47b9
      </wscor:Identifier>
      <wscor:CoordinationType>
        http://schemas.xmlsoap.org/ws/2004/10/wsat
      </wscor:CoordinationType>
      <wscor:RegistrationService xmlns:wscor="...">
        <wsa:Address xmlns:wsa="...">
          http://9.100.199.238:9080/_IBMSYSAPP/wscor/services/RegistrationCoordinatorPort
        </wsa:Address>
        <wsa:ReferenceProperties xmlns:wsa="...">
          <websphere-wsat:txID xmlns:websphere-wsat="...">
            com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
            ecc889b8eab92e29d91254fd4fb0ff47b9
          </websphere-wsat:txID>
          <websphere-wsat:instanceID xmlns:websphere-wsat="...">
            com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
            ecc889b8eab92e29d91254fd4fb0ff47b9
          </websphere-wsat:instanceID>
        </wsa:ReferenceProperties>
      </wscor:RegistrationService>
    </wscor:CoordinationContext>
  </soapenv:Header>
  <soapenv:Body>
    <p635:DFHOXCMN xmlns:p635="http://www.DFHOXCMN.DFHOXCP5.Request.com">
      <p635:ca_request_id>010RDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>Luis</p635:ca_userid>
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<p635:ca_charge_dept>D001</p635:ca_charge_dept>
<p635:ca_item_ref_number>10</p635:ca_item_ref_number>
<p635:ca_quantity_req>1</p635:ca_quantity_req>
<p635:filler1 xsi:nil="true"/>
</p635:ca_order_request>
</p635:DFHOXCMN>
</soapenv:Body>
</soapenv:Envelope>
```

▶ SOAP header:

The SOAP header contains only one header, a `CoordinationContext` header. The `CoordinationContext` header has a `mustUnderstand` attribute whose value is `1`. This means that CICS must process the `CoordinationContext` header. If CICS cannot process the `CoordinationContext` header (for example, because the provider pipeline configuration file does not specify the `DFHWSATH` message handler) then CICS must stop all further processing of the message and generate a SOAP fault whose fault code is `MustUnderstand`.

The `CoordinationContext` element contains four elements:

– Expires

CICS treats the content of the `Expires` element as a character string and determines whether the string represents a number.

- If it does, it is treated as a millisecond value, which is then converted to an integer representing the number of seconds for which the Web service transaction waits for a response to a **Register** request or for the Coordinator to send various 2PC messages. If this integer is 0, or greater than 4080, then CICS uses 4080 seconds (the maximum value which CICS allows for the `DTIMOUT` attribute of the `TRANSACTION` resource definition).
- If it does not, then CICS does not set a suspend time and the Web service transaction waits forever.

Note: We noted that the `Expires` element was not an unsigned integer but the text `Never`. The default CICS behavior is not to terminate the transaction but to wait indefinitely for WebSphere to commit or roll back the atomic transaction.

– Identifier

WebSphere creates this identifier as a unique global identifier for each WS-AT transaction as required by the WS-Coordination specification.

– CoordinationType

The CoordinationType element specifies the WS-AtomicTransaction coordination type.

– RegistrationService

The RegistrationService element contains an endpointReference for WebSphere's Registration service. The endpointReference has two elements:

- Address

The Address element tells CICS where to send its **Register** request. CICS copies the contents of the Address element to the To message information header when it builds its **Register** request.

- ReferenceProperties

WebSphere provides two reference properties: txID and instanceID. The content of the txID element is the same as the content of the Identifier element. The content of the instanceID element is initially the same as that of the txID element, but that changes later.

CICS adds these reference properties to the SOAP Header when it builds its **Register** request. This allows WebSphere to map the **Register** request to this service request. The detail which WebSphere puts into the txID and instanceID properties is implementation specific and is only ever parsed and understood by WebSphere.

► SOAP body

The content of the <p635:ca_request_id> element tells CICS which Web service to invoke; 01ORDR indicates the placeOrder Web service. The contents of the <p635:ca_userid>, <p635:ca_charge_dept>, <p635:ca_item_ref_number>, and <p635:ca_quantity_req> elements provide the placeOrder Web service with the details of the order (see Figure 9-17 on page 279).

Register (message 2)

Example 9-11 shows the SOAP message which contains the **Register** request that CICS sends to WebSphere. The message consists of a SOAP envelope which contains a SOAP header and a SOAP body.

Example 9-11 CICS sends a Register request to WebSphere

```
<soap:Envelope xmlns:wscor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
  <soap:Header>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/10/wscor/Register
```

```

</wsa:Action>
<wsa:MessageID>PIAT-MSG-A6POT3C1-003343146052627C</wsa:MessageID>
<wsa:ReplyTo>
  <wsa:Address>
    http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
  </wsa:Address>
  <wsa:ReferenceProperties>
    <cicswsat:UOWID>BE092025168B596B</cicswsat:UOWID>
    <cicswsat:PublicId>
      310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
      F1C3C9E6E2F3C44040BE092025167B0000092025167B00004040404040404040
    </cicswsat:PublicId>
  </wsa:ReferenceProperties>
</wsa:ReplyTo>
<wsa:To>
  http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
  </wsa:To>
  <websphere-wsat:txID xmlns:websphere-wsat="...">
    com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
    ecc889b8eab92e29d91254fd4fb0ff47b9
  </websphere-wsat:txID>
  <websphere-wsat:instanceID xmlns:websphere-wsat="...">
    com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
    ecc889b8eab92e29d91254fd4fb0ff47b9
  </websphere-wsat:instanceID>
</soap:Header>
<soap:Body>
  <wscoor:Register>
    <wscoor:ProtocolIdentifier>
      http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
    </wscoor:ProtocolIdentifier>
    <wscoor:ParticipantProtocolService>
      <wsa:Address>
        http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </wsa:Address>
      <wsa:ReferenceProperties>
        <cicswsat:UOWID>BE092025168B596B</cicswsat:UOWID>
        <cicswsat:PublicId>
          310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
          F1C3C9E6E2F3C44040BE092025167B0000092025167B00004040404040404040
        </cicswsat:PublicId>
      </wsa:ReferenceProperties>
    </wscoor:ParticipantProtocolService>
  </wscoor:Register>

```

`</soap:Body>`
`</soap:Envelope>`

► SOAP header:

The SOAP Header contains several message information headers and the two reference properties (`txID` and `instanceID`) which WebSphere sent in its Web service request. These reference properties allow WebSphere to match this **Register** request to its initial request for the `placeOrder` service. The message information headers are as follows:

– To

The `To` header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region which is monitoring port 9080 on a system whose IP address is 9.100.199.238. CICS copied this from the `Address` element of the `endpointReference` for the `RegistrationService` in WebSphere's service request.

– Action

The `Action` header indicates that CICS wishes to register with the Registration Coordinator Port.

– MessageID

The ID of the message uniquely identifies the message in space and time:

- `PIAT` is part of the name of module `DFHPIAT`; `DFHPIAT` generates `CoordinationContext` elements for CICS and interprets `CoordinationContext` elements received from other systems.
- `A6POT3C1` is the VTAM APPLID of the CICS region which is sending the message.
- `003343146052627C` is the abstime value returned by an EXEC CICS INQUIRE TIME issued in our CICS region.

– ReplyTo

The `Address` element of the `ReplyTo` header shows that the reply to this message should be sent to the Registration service running in a CICS region which is monitoring port 15301 on a z/OS system whose IP address is `MVSG3.mop.ibm.com`. This Registration service has two reference properties:

- `UOWID`
- `PublicID`

When WebSphere sends the **RegisterResponse** to CICS, it adds each of these reference properties to the response as a SOAP header. CICS uses the `PublicID` to find the region and the UOW in that region where the Web service provider is waiting for the response; then CICS routes it there for processing. The detail which CICS puts into the `UOWID` and `PublicID` properties is there for CICS to understand; other products do not use it.

► SOAP body:

The SOAP Body contains the **Register** request, which contains two elements:

- `ProtocolIdentifier`

CICS uses this element to register for the durable two-phase commit protocol.

- `ParticipantProtocolService`

Note that the address of CICS's Protocol service is the same as the address of its Registration service.

RegisterResponse (message 3)

Example 9-12 shows the **RegisterResponse** that WebSphere sends to CICS.

Example 9-12 WebSphere sends a RegisterResponse to CICS

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
    xmlns:wsa="...">
  <soapenv:Header>
    <wsa:MessageID>uuid:10A19D35-0108-4000-E000-094409D4835B</wsa:MessageID>
    <wsa:To>http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService</wsa:To>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse
    </wsa:Action>
    <wsa:FaultTo xmlns:wsa="...">
      <wsa:Address>
        http://9.100.199.238:9080/_IBMSYSAPP/wsatfault/services/WSATFaultPort
      </wsa:Address>
      <wsa:ReferenceProperties xmlns:wsa="...">
        <websphere-wsat:txID xmlns:websphere-wsat="...">
          com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
          ecc889b8eab92e29d91254fd4fb0ff47b9
        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
          com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
          ecc889b8eab92e29d91254fd4fb0ff47b9
        </websphere-wsat:instanceID>
      </wsa:ReferenceProperties>
    </wsa:FaultTo>
  </soapenv:Header>
  <soapenv:Body>
    <RegisterRequest xmlns="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
      </RegisterRequest>
  </soapenv:Body>
</soapenv:Envelope>
```



```

</wsa:FaultTo>
<wsa:From xmlns:wsa="...">
  <wsa:Address>
    http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
  </wsa:Address>
  <wsa:ReferenceProperties xmlns:wsa="...">
    <websphere-wsat:txID xmlns:websphere-wsat="...">
      com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
      ecc889b8eab92e29d91254fd4fb0ff47b9
    </websphere-wsat:txID>
    <websphere-wsat:instanceID xmlns:websphere-wsat="...">
      com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
      ecc889b8eab92e29d91254fd4fb0ff47b9
    </websphere-wsat:instanceID>
  </wsa:ReferenceProperties>
</wsa:From>
<wsa:RelatesTo>PIAT-MSG-A6P0T3C1-003343146052627C</wsa:RelatesTo>
<cicswsat:UOWID xmlns:cicswsat="...">BE092025168B596B</cicswsat:UOWID>
<cicswsat:PublicId xmlns:cicswsat="...">
  310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
  F1C3C9E6E2F3C44040BE092025167B0000092025167B00004040404040404040
</cicswsat:PublicId>
</soapenv:Header>
<soapenv:Body>
  <RegisterResponse xmlns="http://schemas.xmlsoap.org/ws/2004/10/wscoor">
    <wscoor:CoordinatorProtocolService xmlns:wscoor="...">
      <wsa:Address xmlns:wsa="...">
        http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
      </wsa:Address>
      <wsa:Reference Properties xmlns:wsa="...">
        <websphere-wsat:txID xmlns:websphere-wsat="...">
          com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
          ecc889b8eab92e29d91254fd4fb0ff47b9
        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
          http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
        </websphere-wsat:instanceID>
      </wsa:ReferenceProperties>
    </wscoor:CoordinatorProtocolService>
  </RegisterResponse>
</soapenv:Body>
</soapenv:Envelope>

```

► SOAP header:

The SOAP Header contains several message information headers and the two reference properties (UOWID and PublicID), which CICS sent in its **Register** request. If we were running a CICSplex, CICS would use these reference properties to route the response to the region and the UOW in that region where the Web service provider is waiting for it. The message information headers are as follows:

– MessageID

– To

WebSphere is sending this message to CICS's RegistrationService. It obtained this address from the ReplyTo header in the **Register** request.

– Action

This message is a **RegisterResponse**.

– FaultTo

WebSphere wants CICS to send any SOAP faults that it has to generate to WebSphere's WSATFaultPort.

– From

This message is coming from WebSphere's RegistrationCoordinatorPort.

– RelatesTo

This message relates to the **Register** request which CICS sent.

► SOAP body:

The SOAP Body contains the RegisterResponse element, which in turn contains only a CoordinatorProtocolService element. The address of WebSphere's Protocol Service is http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator. Note that its two reference properties, txID and instanceID, no longer have the same contents; instanceID now contains the address of CICS's Registration service.

Web service response (message 4)

Since CICS has registered its interest in the atomic transaction and WebSphere has responded to that, CICS can now run the placeOrder service. When the placeOrder service has completed its work, CICS sends a response to WebSphere as shown in Example 9-13. The essence of the response is the message ORDER SUCCESSFULLY PLACED.

Example 9-13 CICS sends the Web service response to WebSphere

```
<SOAP-ENV:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
  xmlns:xsi="..." xmlns:wscor="..." xmlns:wsa="..."
  xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    <DFHOXCMNResponse xmlns="http://www.DFHOXCMN.DFHOXCP5.Response.com">
      <ca_request_id>01ORDR</ca_request_id>
      <ca_return_code>0</ca_return_code>
      <ca_response_message>ORDER SUCESSFULLY PLACED</ca_response_message>
      <ca_order_request>
        <ca_userid>Luis </ca_userid>
        <ca_charge_dept>D001 </ca_charge_dept>
        <ca_item_ref_number>10</ca_item_ref_number>
        <ca_quantity_req>1</ca_quantity_req>
        <filler1>
          ...
        </filler1>
      </ca_order_request>
    </DFHOXCMNResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Prepare (message 5)

When the `handlePlaceOrder` method of the `CatalogController` servlet issues the `userTransaction.commit` command, WebSphere sends a **Prepare** command to CICS as shown in Example 9-14 in the Action message information header and in the `<p320:Prepare.../>` element of the SOAP Body.

Example 9-14 WebSphere sends a Prepare notification to CICS

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
  xmlns:xsi="..." xmlns:wsa="...">
  <soapenv:Header>
    <wsa:MessageID>uuid:10A19F38-0108-4000-E000-094409D4835B</wsa:MessageID>
    <wsa:To>http://MVS63.mop.ibm.com:15301/cicswsat/RegistrationService</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare</wsa:Action>
    <wsa:FaultTo xmlns:wsa="...">
      <wsa:Address>
        http://9.100.199.238:9080/_IBMSYSAPP/wsatfault/services/WSATFaultPort
      </wsa:Address>
      <wsa:ReferenceProperties xmlns:wsa="...">
        <websphere-wsat:txID xmlns:websphere-wsat="...">
          com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
          ecc889b8eab92e29d91254fd4fb0ff47b9
        </websphere-wsat:txID>
      </wsa:ReferenceProperties>
    </wsa:FaultTo>
  </soapenv:Header>
```

```

        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
            http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
        </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
</wsa:FaultTo>
<wsa:ReplyTo xmlns:wsa="...">
    <wsa:Address>
        http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
    </wsa:Address>
    <wsa:ReferenceProperties xmlns:wsa="...">
        <websphere-wsat:txID xmlns:websphere-wsat="...">
            com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
            ecc889b8eab92e29d91254fd4fb0ff47b9
        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
            http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
        </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
</wsa:ReplyTo>
<cicswsat:UOWID xmlns:cicswsat="...">BE092025168B596B</cicswsat:UOWID>
<cicswsat:PublicId xmlns:cicswsat="...">
    310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
    F1C3C9E6E2F3C44040BE092025167B0000092025167B00004040404040404040
</cicswsat:PublicId>
</soapenv:Header>
<soapenv:Body>
    <p320:Prepare xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
    </soapenv:Body>
</soapenv:Envelope>

```

Prepared (message 6)

CICS responds with **Prepared** as shown in Example 9-15 in the Action message information header and in the <wsat:Prepared.../> element of the SOAP Body. Since **Prepared** is a terminating message, it does not contain a ReplyTo message information header.

Example 9-15 CICS sends Prepared notification to WebSphere

```

<soap:Envelope xmlns:wscor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
    <soap:Header>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared</wsa:Action>
        <wsa:MessageID>PIAT-MSG-A6P0T3C1-003343146053278C</wsa:MessageID>
    </soap:Header>

```

```

<wsa:To>http://9.100.199.238:9080/_IBMSYSAPP/wsatservices/Coordinator</wsa:To>
<websphere-wsat:txID xmlns:websphere-wsat="...">
  com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
  ecc889b8eab92e29d91254fd4fb0ff47b9
</websphere-wsat:txID>
<websphere-wsat:instanceID xmlns:websphere-wsat="...">
  http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
</websphere-wsat:instanceID>
</soap:Header>
<soap:Body>
  <wsat:Prepared xmlns:wsat="..."/>
</soap:Body>
</soap:Envelope>

```

Commit (message 7)

Since CICS has voted **yes** in response to the **Prepare** command, and since no other system has registered an interest in this transaction, WebSphere sends a **Commit** command to CICS. See the Action header and the `<p320:Commit.../>` element of the SOAP Body in Example 9-16.

Example 9-16 WebSphere sends Commit notification to CICS

```

<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
  xmlns:wsa="...">
  <soapenv:Header>
    <wsa:MessageID>uuid:10A19F67-0108-4000-E000-094409D4835B</wsa:MessageID>
    <wsa:To>http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService</wsa:To>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit</wsa:Action>
  </soapenv:Header>
  <wsa:FaultTo xmlns:wsa="...">
    <wsa:Address>
      http://9.100.199.238:9080/_IBMSYSAPP/wsatfault/services/WSATFaultPort
    </wsa:Address>
    <wsa:ReferenceProperties xmlns:wsa="...">
      <websphere-wsat:txID xmlns:websphere-wsat="...">
        com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
        ecc889b8eab92e29d91254fd4fb0ff47b9
      </websphere-wsat:txID>
      <websphere-wsat:instanceID xmlns:websphere-wsat="...">
        http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
  </wsa:FaultTo>
  <wsa:ReplyTo xmlns:wsa="...">

```

```

<wsa:Address>
  http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
</wsa:Address>
<wsa:ReferenceProperties xmlns:wsa="...">
  <websphere-wsat:txID xmlns:websphere-wsat="...">
    com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
    ecc889b8eab92e29d91254fd4fb0ff47b9
  </websphere-wsat:txID>
  <websphere-wsat:instanceID xmlns:websphere-wsat="...">
    http://MVS3.mop.ibm.com:15301/cicswsat/RegistrationService
  </websphere-wsat:instanceID>
</wsa:ReferenceProperties>
</wsa:ReplyTo>
<cicswsat:UOWID xmlns:cicswsat="...">BE092025168B596B</cicswsat:UOWID>
<cicswsat:PublicId xmlns:cicswsat="...">
  310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
  F1C3C9E6E2F3C44040BE092025167B0000092025167B00004040404040404040
</cicswsat:PublicId>
</soapenv:Header>
<soapenv:Body>
  <p320:Commit xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
</soapenv:Body>
</soapenv:Envelope>

```

Committed (message 8)

After committing the update to the EXMPCAT VSAM file, CICS sends a **Committed** notification to WebSphere. See the Action message information header and the <wsat:Committed.../> element of the SOAP Body in Example 9-17. Since **Committed** is a terminating message, it does not contain a ReplyTo message information header.

Example 9-17 CICS sends Committed notification to WebSphere

```

<soap:Envelope xmlns:wscoor="..." xmlns:wsa="..." xmlns:cicswsat="..."
  xmlns:soap="...">
  <soap:Header>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed</wsa:Action>
    <wsa:MessageID>PIAT-MSG-A6POT3C1-003343146053324C</wsa:MessageID>
    <wsa:To>http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator</wsa:To>
    <websphere-wsat:txID xmlns:websphere-wsat="...">
      com.ibm.ws.wstx:0000010810a19b02000000010000000683e8b0
      ecc889b8eab92e29d91254fd4fb0ff47b9
    </websphere-wsat:txID>

```

```
<websphere-wsat:instanceID xmlns:websphere-wsat="...">
  http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
</websphere-wsat:instanceID>
</soap:Header>
<soap:Body>
  <wsat:Committed xmlns:wsat="..." />
</soap:Body>
</soap:Envelope>
```

The set of eight Web service, registration, and protocol service messages we have described here are the flows that guarantee database consistency in our scenario, a scenario in which the databases are accessed by middleware (WebSphere Application Server on a Windows platform and CICS on z/OS); and communication between the middleware products is based on open Web services standards. The same basic flows could be used between any middleware products which support the Web services, Web Services-Coordination (WS-C) and Web Services-Atomic Transaction (WS-AT) specifications.

Simple scenario: Abnormal transaction termination

We now return to the Place Order window to place a new order. This time we used ROLLBACK as the User Name. This tells the servlet to throw an exception after inserting the order in the DB2 table and placing the order with CICS.

Example 9-18 CatalogController servlet - Throwing the RemoteException

```
if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
{
  System.out.println("CatalogController.handlePlaceOrder() - simulating the
RemoteException");
  throw new RemoteException("Throwing the RemoteException");
}
```

On the Order Entry page, we entered the User Name ROLLBACK and clicked **SUBMIT** to place a new order (Figure 9-22). Note that, at this time, we still had 75 items in stock (see Figure 9-19 on page 280).

CICS Example - Catalog Application

Enter Order Details

LIST ITEMS	Item Reference Number	0010
INQUIRE	Quantity	001
	User Name	ROLLBACK
	Department Name	D001

SUBMIT

BACK

CONFIGURE

CICS Transaction Server for z/OS

Figure 9-22 Catalog Application - Place order for ROLLBACK

Figure 9-23 shows the expected error response.

CICS Example - Catalog Application

An Error Has Occurred

LIST ITEMS

INQUIRE

ORDER ITEM

Error:

An Error occured calling the service: Throwing the RemoteException

BACK

BACK

CONFIGURE

CICS Transaction Server for z/OS

Figure 9-23 Catalog Application - Error window

When the Exception is thrown, the transaction is rolled back; the order is deleted from the table and the CICS transaction is rolled back too.

We again checked in the WebSphere server log to see what happened (Example 9-19).

Example 9-19 WebSphere server log with ROLLBACK Place Order

```

CatalogController.doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction

```

We can see in the log that the order was inserted in the database and a good response was returned from calling the CICS Web service. Then we simulated the RemoteException and the rollback of the whole transaction took place.

When we inquired on the stock level, we noted that there were still 75 items in stock.

Figure 9-24 shows the registration and protocol service messages that CICS and WebSphere Application Server exchanged during our test.

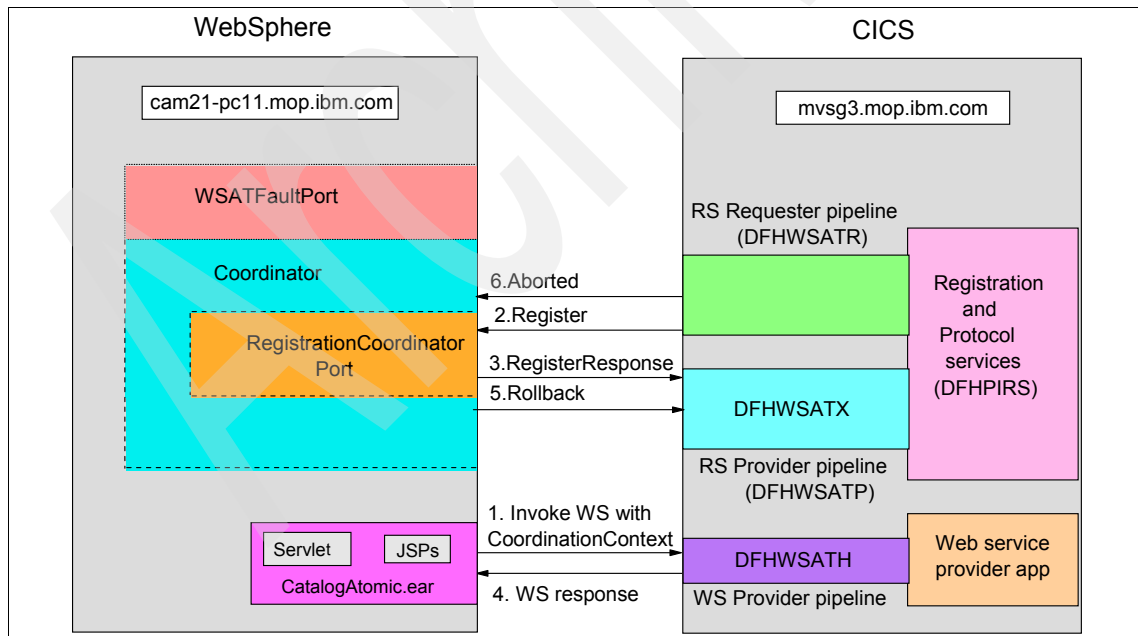


Figure 9-24 Messages exchanged when both sides back out

9.3 The daisy chain atomic transaction scenario

We now extend the simple atomic transaction scenario to a *daisy chain* scenario by making the following modifications:

- ▶ Changing the configuration of the CICS sample Catalog application such that it makes an outbound Web service call
- ▶ Changing the ExampleAppDispatchOrder.ear file provided with CICS TS V3.1

ExampleAppDispatchOrder.ear is an enterprise archive provided with the Catalog application that can be deployed in WebSphere Application Server and used as an order dispatch endpoint. See “Catalog manager example application” on page 53 for more information about the sample application.

We modified the ExampleAppDispatchOrder.ear file to insert records into a DB2 table called ITSO.DISPATCH. The ITSO.DISPATCH table logs all of the orders dispatched through the dispatchOrder Web service. We called the modified program DispatchOrderAtomic.ear.

By making these changes, we have a global transaction that updates three resources:

- ▶ A DB2 table in the Windows environment
- ▶ A VSAM file in the z/OS environment
- ▶ Another DB2 table in a Windows environment

We have a daisy chain from WebSphere to CICS to WebSphere. CICS acts as both a coordinator and a participant. Figure 9-25 shows the sequence of events for the whole transaction.

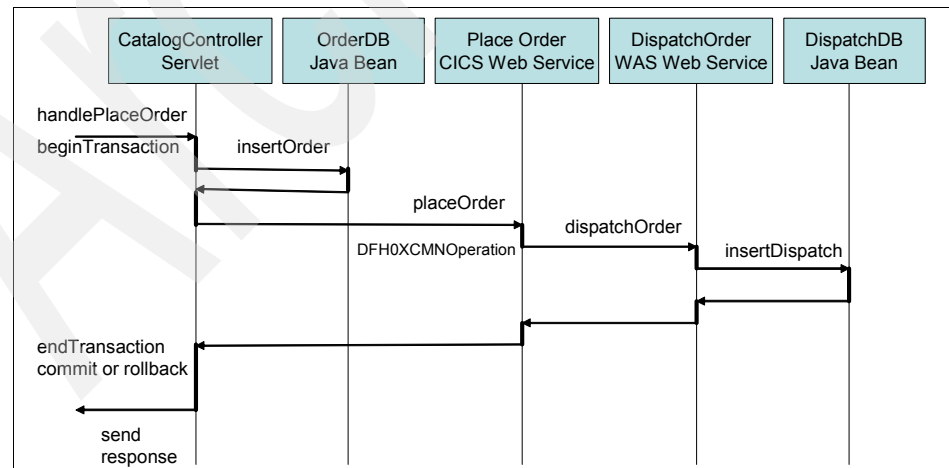


Figure 9-25 Daisy chain scenario sequence of events

Figure 9-26 shows a more global view of the sequence of events:

1. The user uses his Web browser to invoke the AtomicClient CatalogController servlet, which runs in WebSphere Application Server.
2. The AtomicClient updates the ITSO.ORDER table in DB2.
3. WebSphere Application Server sends the placeOrder SOAP message containing the order to CICS.
4. CICS uses the Web services provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
5. DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
6. CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
7. DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.
8. DFH0XCMN passes the data in the COMMAREA to DFH0XWOD.
9. DFH0XWOD invokes the dispatchOrder Web service, which points to the PIPELINE PIPE2.
10. PIPE2 contains DFHPISN1, which:
 - a. Converts the COMMAREA to a SOAP message body
 - b. Calls DFHWSATH to create a CoordinationContext SOAP header
 - c. Sends the dispatchOrder SOAP message to WebSphere
11. DispatchOrderAtomic updates the ITSO.DISPATCH table in DB2.

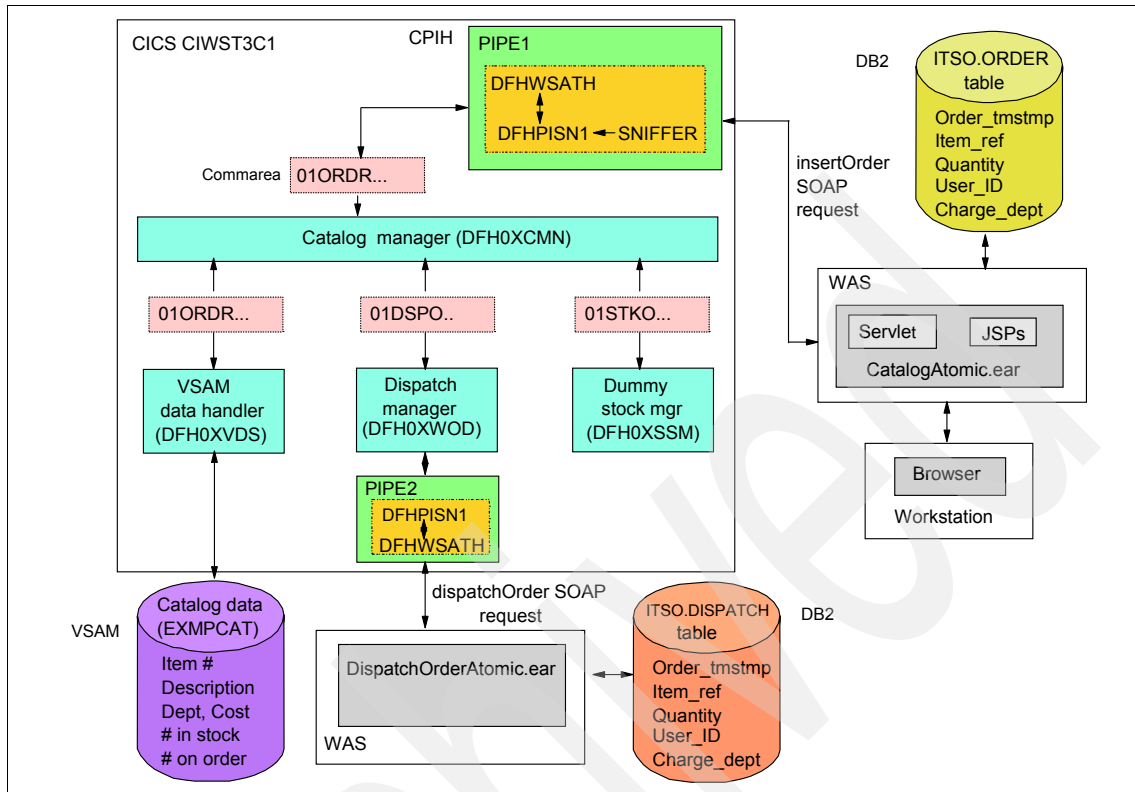


Figure 9-26 Daisy chain: CICS as a service provider and as a service requester

In the following sections we describe how we set up CICS for the daisy chain scenario, and how we created the DispatchOrderAtomic application and the ITSO.DISPATCH table. We then show the results of testing this scenario.

9.3.1 Setting up CICS for the daisy chain scenario

To set up CICS for this scenario we performed the following steps in addition to those shown in 9.2.1, “Setting up CICS for the simple scenario” on page 257:

1. We edited PIPE2’s configuration file
/CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

It now contains the XML shown in Example 9-20.

Example 9-20 PIPE2:/CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/software/http/cics/pipeline
requester.xsd ">
<service>
  <service_handler_list>
    <cics_soap_1.1_handler>
      <headerprogram>
        <program_name>DFHWSATH</program_name>
        <namespace>
          http://schemas.xmlsoap.org/ws/2004/10/wscoord
        </namespace>
        <localname>CoordinationContext</localname>
        <mandatory>true</mandatory>
      </headerprogram>
    </cics_soap_1.1_handler>
  </service_handler_list>
</service>
<default_transport_handler_list>
  <handler>
    <program>SNIFFER</program>
    <handler_parameter_list/>
  </handler>
</default_transport_handler_list>
<service_parameter_list>
  <registration_service_endpoint>
    http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
  </registration_service_endpoint>
</service_parameter_list>
</requester_pipeline>

```

2. We modified the configuration file for the sample Catalog application using the ECFG transaction at a 3270 terminal connected to our CICS region (see Figure 3-17 on page 110). In the 3270 screen titled Configure CICS Example Catalog Application:
 - a. We set the value of the Outbound WebService? field on this screen to **Yes**; this causes the Catalog Manager program (DFH0XCMN) to invoke the Dispatch Manager program (DFH0XWOD) rather than the Dummy Dispatch Manager program (DFH0XSOD) that we used in the simple scenario. While DFH0XSOD simply sets the return code in the COMMAREA to 0 and returns to its caller, DFH0XWOD issues an **EXEC CICS INVOKE WEBSERVICE('dispatchOrder')** **URI(outboundWebServiceURI)** command to make an outbound Web service call to an order dispatcher.
 - b. We set the value of the Outbound WebService URI field to the location of the Web service that implements the order dispatcher function. We ran the dispatchOrder service on WebSphere Application Server for Windows.

9.3.2 Creating DispatchOrderAtomic and the ITSO.DISPATCH table

To create the DispatchOrderAtomic application and the ITSO.DISPATCH table, we did the following steps:

1. We created the DispatchDB JavaBean that inserts the dispatch order into the DB2 table.
2. We changed the dispatchOrder method in the DispatchOrderSoapBindingImpl class in order to call the DispatchDB JavaBean. DispatchOrderSoapBindingImpl is the bean that serves the Web service and dispatchOrder is the method that is called when the Web service is requested.
3. We changed the deployment descriptor of the DispatchOrderAtomic application so that it is executed as part of a Web Services Atomic Transaction.
4. We created the ITSO.DISPATCH table.

We explain each step in detail in the following sections.

Create the DispatchDB JavaBean

In the same way that we previously created the OrderDB JavaBean, we created the DispatchDB JavaBean. This bean has a method that inserts the dispatch order into the ITSO.DISPATCH table. The input parameter is an OrderBean that has all the data required for the insert. We do not show the DispatchDB JavaBean code here because it is very similar to the OrderDB code shown in Example 9-4 on page 262.

Change the dispatchOrder method

When the dispatchOrder Web service is called, the dispatchOrder method in the DispatchOrderSoapBindingImpl Java class is called. It is responsible to perform the service and send a response. We changed this method to create an OrderBean and to perform the insert by calling the insertDispatch method in the DispatchDB JavaBean.

The dispatchOrder method is shown in Example 9-21.

Example 9-21 The dispatchOrder method

```
public com.Response.dispatchOrder.exampleApp.www.DispatchOrderResponse
    dispatchOrder(com.Request.dispatchOrder.exampleApp.www.DispatchOrderRequest
requestPart) throws java.rmi.RemoteException {
    System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder():"+
        " ItemRef="+requestPart.getItemReferenceNumber()+
        " Quantity="+requestPart.getQuantityRequired()+
        " CustomerName="+requestPart.getCustomerId()+
```

```

        " Dept="+requestPart.getChargeDepartment());

        // Creating the order
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
creating the order");
        OrderBean order = new OrderBean();
        order.setOrderTmstmp(new
Timestamp(Calendar.getInstance().getTime().getTime()));
        order.setItemRef(requestPart.getItemReferenceNumber());
        order.setQuantity(requestPart.getQuantityRequired());
        order.setUserId(requestPart.getCustomerId());
        order.setChargeDept(requestPart.getChargeDepartment());

        // inserting the order in the database
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
inserting the order in the database");
        DispatchDB dispatchDB = new DispatchDB();
        dispatchDB.insertDispatch(order);
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() - after
the insert!!!");

        com.Response.dispatchOrder.exampleApp.www.DispatchOrderResponse
response = new
com.Response.dispatchOrder.exampleApp.www.DispatchOrderResponse();
        response.setConfirmation("Order in Dispatch");
        return response;
    }

```

Change the deployment descriptor

In order to specify that the dispatchOrder service should participate in the global transaction, we had to modify the Web Application deployment descriptor.

To activate WS-AT support:

1. We imported the application archive DispatchOrderAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (DispatchOrderAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 9-27).

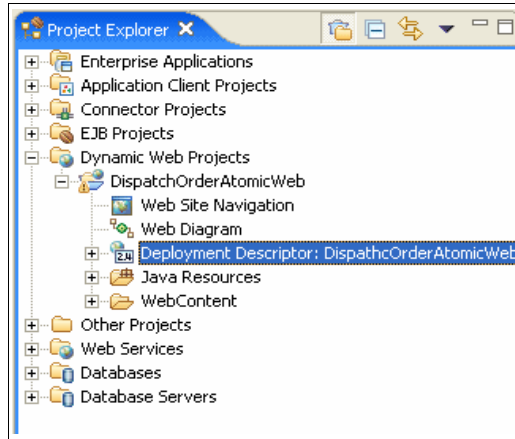


Figure 9-27 Opening the DispatchOrderAtomicWeb Deployment Descriptor

2. In the Deployment Descriptor:
 - a. We clicked the **Servlets** tab.
 - b. We selected the servlet, **com_dispatchOrder_exampleApp_www_DispatchOrderSoapBindingImpl**.
 - c. Then we scrolled down to Global Transaction and selected **Execute using Web Services Atomic Transaction on incoming request** (Figure 9-28).

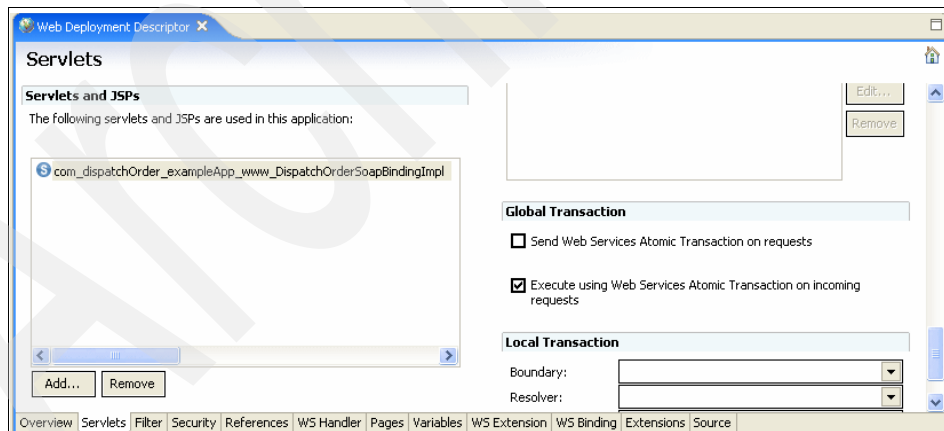


Figure 9-28 The DispatchOrderAtomicWeb Deployment Descriptor

3. We saved and closed the file. Now, when an incoming request contains a coordination context, the dispatchOrder Web service joins the global transaction.

Create the ITSO.DISPATCH table

We created the ITSO.DISPATCH table to record all the dispatch orders that come to the dispatchOrder Web service. Example 9-22 shows the script we used to create the table.

Example 9-22 Creation of ITSO.DISPATCH table

```
-- IBM ITSO

CONNECT RESET;

CONNECT TO ITSOWS;

-- Table definitions for Dispatch
CREATE TABLE ITSO.DISPATCH
  (DISPATCH_TMSTMP TIMESTAMP NOT NULL ,
   ITEM_REF INTEGER NOT NULL ,
   QUANTITY INTEGER NOT NULL ,
   USER_ID CHARACTER (20) NOT NULL ,
   CHARGE_DEPT CHARACTER (20) NOT NULL ,
   CONSTRAINT DISPATCHKEY PRIMARY KEY (DISPATCH_TMSTMP) ) ;
```

The ITSO.DISPATCH table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table.

9.3.3 Testing the daisy chain scenario

We performed two tests of the daisy chain scenario:

- ▶ Normal transaction termination
- ▶ Abnormal transaction termination (see “Daisy chain scenario: abnormal transaction termination” on page 313)

Daisy chain scenario: normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario, we did the following steps:

1. We opened the welcome page of the Catalog application.
`http://cam21-pc11:9080/CatalogAtomicWeb/Welcome.jsp`
2. We clicked **INQUIRE** to perform an inquireSingle operation.
3. In the **Inquire Single** window, we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 9-29.

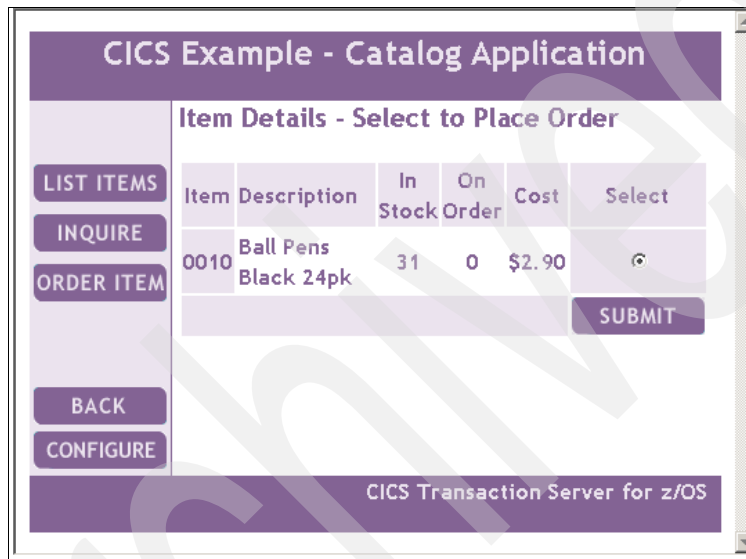


Figure 9-29 Catalog Application - Inquire single before

4. We noted that the number of items in stock was 31. This value is taken from the CICS VSAM file.
5. We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 9-30.

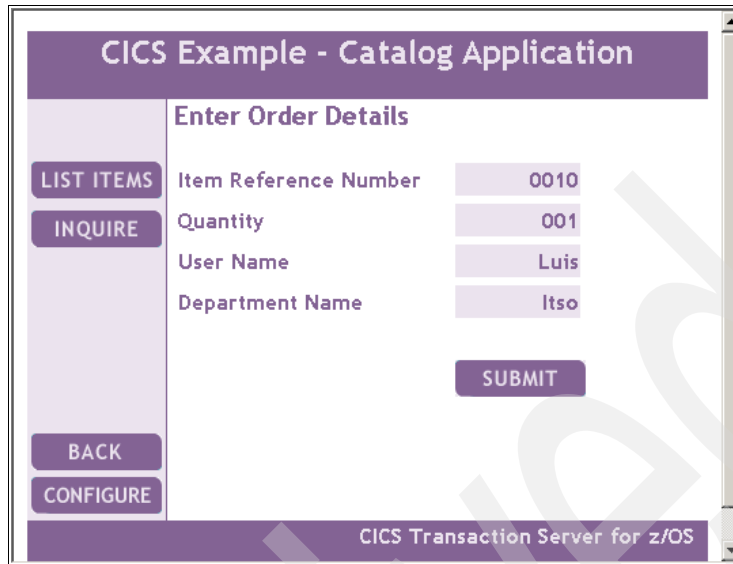


Figure 9-30 Catalog Application - Enter order details

6. In the Enter Order Details window, we provided a User Name and a Department Name and clicked **SUBMIT**.
7. After the CICS Web service processed the order and called the dispatchOrder WebSphere Web service, we got the response telling us that the order was successfully placed (Figure 9-31).

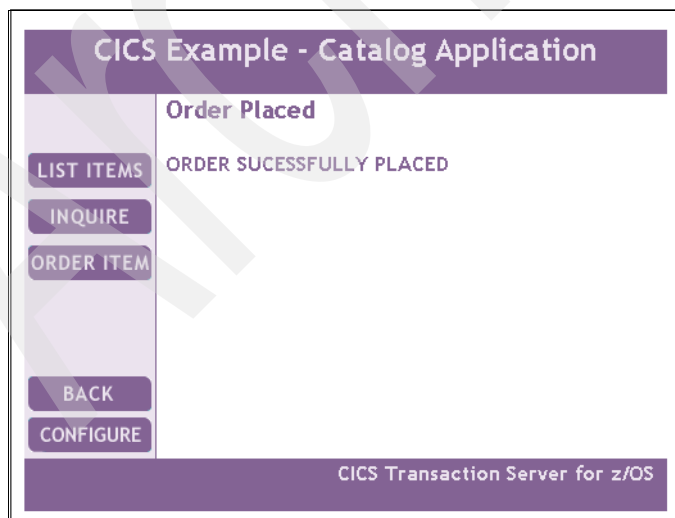


Figure 9-31 Catalog Application - Order successfully placed

8. In the WebSphere server log we see trace entries generated by the CatalogController servlet, which show that the transaction was successfully committed (Example 9-23).

Example 9-23 WebSphere server log with successful Place Order for CatalogController

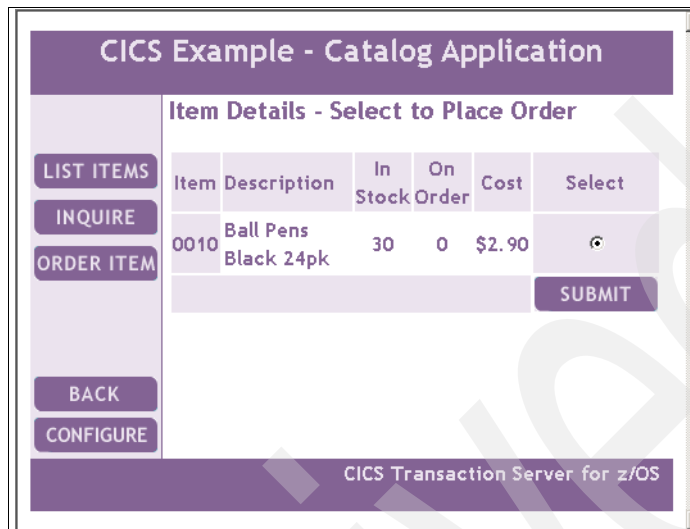
```
CatalogController.doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

9. We also can see the log for the WebSphere Application Server on which the DispatchOrderAtomic application is installed (Example 9-24).

Example 9-24 WebSphere server log with successful Place Order for DispatchOrder

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=Luis Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

10. Next we checked the same item number through the inquireService service and verified that the stock level decreased by one item (Figure 9-32).



The screenshot shows a CICS terminal window titled "CICS Example - Catalog Application". The main heading is "Item Details - Select to Place Order". On the left side, there is a vertical menu with buttons: "LIST ITEMS", "INQUIRE", "ORDER ITEM", "BACK", and "CONFIGURE". The main area contains a table with the following data:

Item	Description	In Stock	On Order	Cost	Select
0010	Ball Pens Black 24pk	30	0	\$2.90	<input type="radio"/>

Below the table is a "SUBMIT" button. At the bottom of the window, it says "CICS Transaction Server for z/OS".

Figure 9-32 Catalog Application - Inquire single after

11. We opened the DB2 Control Center and used the following SQL command on the Windows machine which hosted the ITSO.ORDER table:

```
SELECT * FROM ITSO.ORDER
```

Figure 9-33 shows the new record in the table.

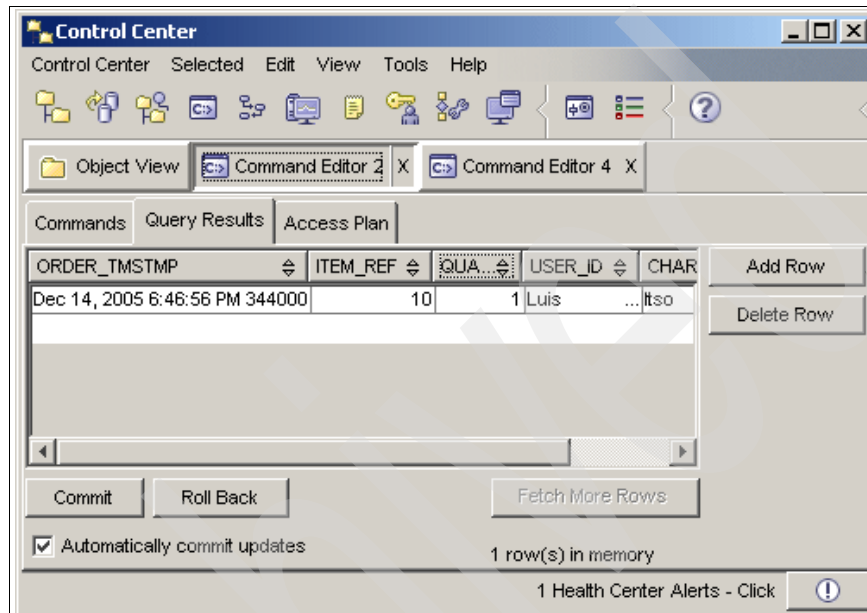


Figure 9-33 The new record in the ITSO.ORDER table

- ▶ We also used the following SQL command on the Windows machine which hosted the ITSO.DISPATCH table:

```
SELECT * FROM ITSO.DISPATCH
```

Figure 9-34 shows the new record in the table.

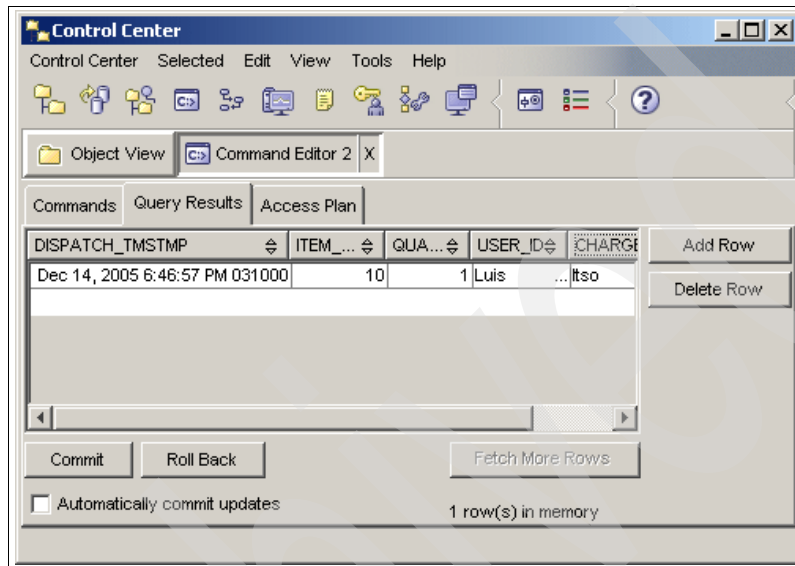


Figure 9-34 The new record in the ITSO.DISPATCH table

Figure 9-35 shows the registration and protocol service messages that are exchanged between CICS and the two WebSphere Application Servers during our test.

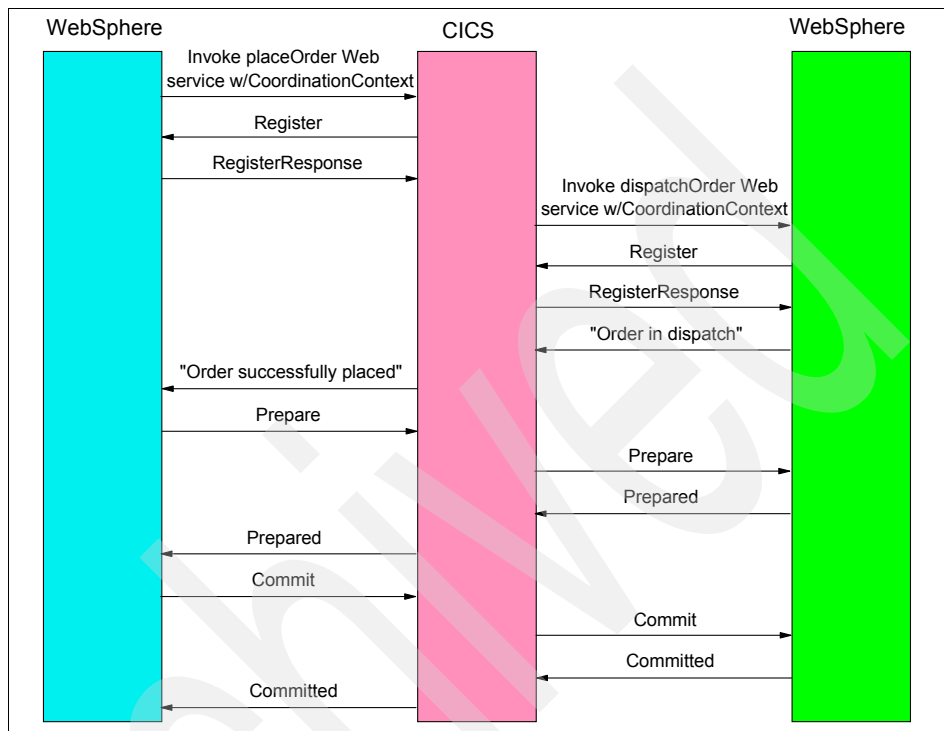


Figure 9-35 Successful daisy chain - Message flow

Example 9-25 shows the dispatchOrder request that CICS sends to WebSphere. Recall that when CICS is a *participant* in a WS-AT transaction, it uses two reference properties: UOWID and PublicID. We see in Example 9-25 that when CICS is the *coordinator*, it uses three reference properties: UOWID, Token, and Netname.

Example 9-25 CICS sends dispatchOrder request to WebSphere

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..." xmlns:wscor="..." xmlns:wsa="..."
xmlns:cicwsat="..." xmlns:soap="...">
  <SOAP-ENV:Header>
    <wscor:CoordinationContext>
      <wscor:Identifier>
        PIAT-CCON-A6POT3C1-003343578075119C
      </wscor:Identifier>
      <wscor:CoordinationType>
```



```

    http://schemas.xmlsoap.org/ws/2004/10/wsat
  </wscoor:CoordinationType>
  <wscoor:RegistrationService>
    <wsa:Address>
      http://MMSG3.mop.ibm.com:15301/cicswsat/RegistrationService
    </wsa:Address>
    <wsa:ReferenceProperties>
      <cicswsat:Netname>A6POT3C1</cicswsat:Netname>
      <cicswsat:Token>F0F0F0F0</cicswsat:Token>
      <cicswsat:UOWID>BEOF698D97751D67</cicswsat:UOWID>
    </wsa:ReferenceProperties>
  </wscoor:RegistrationService>
</wscoor:CoordinationContext>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <dispatchOrderRequest xmlns="http://www.exampleApp.dispatchOrder.Request.com">
    <itemReferenceNumber>10</itemReferenceNumber>
    <quantityRequired>1</quantityRequired>
    <customerId>Luis </customerId>
    <chargeDepartment>Itso </chargeDepartment>
  </dispatchOrderRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Daisy chain scenario: abnormal transaction termination

Finally, we tested the daisy chain scenario, but this time with a RemoteException thrown at the end of the execution. We returned to the Place Order window to place a new order. This time we used ROLLBACK as the User Name (see Figure 9-22 on page 296). In the daisy chain scenario, this exception takes place after the insertions in the two DB2 tables and the update of the CICS VSAM file.

When the exception is thrown, the transaction is rolled back and the updates are backed out from:

- ▶ The ITSO.ORDER table
- ▶ The CICS VSAM file
- ▶ The ITSO.DISPATCH table

We can see in the WebSphere logs that the AtomicClient application successfully inserted the order record in the database prior to rolling back the transaction (Example 9-26).

Example 9-26 Catalog Application log

```
CatalogController.doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction
```

We also can see that the DispatchOrderAtomic application successfully inserted the order record (Example 9-27).

Example 9-27 DispatchOrder Application log

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=ROLLBACK Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

Following the RemoteException, the atomic transaction is rolled back. When we inquired on the stock level, we noted that there were still 30 items in stock.

9.4 Transaction scenario summary

These test scenarios demonstrate how you can synchronize WebSphere and CICS updates using WS-AT. They show how the classical 2PC distributed transaction can be implemented using Web services, and that the distributed global transaction can be committed or rolled back based on a set of Web service flows that are managed entirely by the WebSphere and CICS middleware.

Important: Before implementing a solution based on WS-AT, you should be aware of the general issues that can arise from any implementation of a distributed transaction (for example, locked records preventing access to important data) and you should also compare the solution with alternatives such as the J2EE Connector Architecture.

Additional material

This appendix refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247657>

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 IBM Redbooks form number, SG247657.

Here we provide the file, `saz099.catalog.jar`, for the scenario presented in Chapter 5, “MTOM/XOP optimization” on page 143.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 318. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864
- ▶ *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- ▶ *CICS Transaction Server V3R1 Channels and Containers Revealed*, SG24-7227

Other publications

These publications are also relevant as further information sources:

- ▶ *CICS Web Services Guide*, SC34-6838

Online resources

These Web sites are also relevant as further information sources:

- ▶ Information about XML:
<http://www.w3.org/XML/>
- ▶ The specification for WSDL 1.1:
<http://www.w3.org/TR/wsd1>
- ▶ The specification for WSDL 2.0:
<http://www.w3.org/TR/wsd120>
- ▶ UDDI Version 3.0 and UDDI Version 2.0 information:
<http://www.uddi.org/>

- ▶ *ICS TS support for WebSphere Service Registry and Repository SupportPac:*
<http://www.software.ibm.com/ts/cics/txppacs>

How to get IBM Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

3270 interface 54

A

aborted 229

APARs

PK10849 255

PK16509 253

APP-HANDLER 44

application mapper 37

application sample 88, 145

APP-NAMESPACES 44

ATM

application 201

atomic transaction 197

AtomicClient 255–256, 261, 277

Automatic Teller Machine (ATM) 198

availability 139

configuring for 113

B

bean-managed transaction 266

binding 165

static 168

C

C 38, 41

C++ 38, 41

canonical data model 160

catalog application 135, 255, 277

Catalog manager

example application 53

catalog manager application

configuration transaction, ECFG 109, 135, 301

testing 102, 112, 137

Catalog.ear 99

CatalogAtomic.ear 266

CEDA 131, 259

CEDA transaction 58

CEMT 132

CICS

as a service provider 46

as a service requester 50

client 76

EXEC CICS INVOKE WEBSERVICE 75

Execution Diagnostic Facility (EDF) 137

service provider

using HTTP 88–89

using WMQ 128

service requester

timeout considerations 108

using HTTP 106–107

using WMQ 133

trace 117

URIMAP 57

Web Services Assistant 70

Web services assistant 70

Web services support

high availability 113, 139

problem determination 116

CICS SupportPac

downloading 171

installation 171–173

CICS TS V3

Web service resource definitions 57

CICS Web service APIs 45

CICSplex 114

CICS-to-CICS 124

CIWSMSGH 92, 105, 130

COBOL 38, 41

code page 86, 89

code page conversion 89

COMMAREA 73

commit 199, 229

committed 229

Communications Manager 114

completion protocol 225

CONFIGFILE 59

connection 91

connector 255

CONTAINER 73

container-managed transaction 266

Context containers

63

Control containers

63

- CoordinationContext 216
 - elements 284
 - Address 285
 - CoordinationType 285
 - Expires 284
 - Identifier 284
 - ReferenceProperties 285
 - RegistrationService 285
 - example 283
 - sample created by CICS 218
 - sample created by WebSphere 219
- coordinator 222, 252
- CPIH 105
- CPIL 128

D

- daisy chain 252
- DB2 table 255
- DFHOXCMN 55
- DFHCSDUP batch utility 58
- DFHERROR 44
- DFHFUNTION 44
- DFHHANDLERPLIST 44
- DFHHEADER 44
- DFHLS2WS 39, 70, 73–74, 173
- DFHNORESPONSE 44
- DFHPIRS 246
- DFHPISN1, message handler module 236, 256
- DFHREQUEST 44
- DFHRESPONSE 44
- DFHRESPONSE container 94
- DFHRSURI, URIMAP resource 242
- DFH-SERVICEPLIST 44
- DFHSR2WS 179
 - JCL 181
 - running 191
- DFHWS2LS 39, 70, 75–76
- DFHWS2SR 173
 - JCL 174
 - running 186
- DFHWS-APPHANDLER 44
- DFHWSATH, header processing program 236, 243–244, 246, 256
- DFHWSATP, provider pipeline 237, 239
- DFHWSATR, requester pipeline 237
 - PIPELINE definition 241
- DFHWSATX, message handler 237, 240
- DFHWS-BODY 44

- DFHWS-CID-DOMAIN 44
- DFHWS-DATA 44, 52
- DFHWS-IDTOKEN 44
- DFHWS-MEP 44
- DFHWS-MTOM-IN 44
- DFHWS-MTOM-OUT 44
- DFHWS-OPERATION 44
- DFHWS-PIPELINE 44
- DFHWS-RESTOKEN 44
- DFHWS-SERVICEURI 44
- DFHWS-SOAPACTION 44
- DFHWS-SOAPLEVEL 44
- DFHWS-STSACTION 44
- DFHWS-STSFault 44
- DFHWS-STSURi 44
- DFHWS-TOKENTYPE 44
- DFHWS-TRANID 44
- DFHWS-URI 44
- DFHWS-USERID 44
- DFHWS-WEBSERVICE 44
- DFHWS-XMLNS 44
- DFHWS-XOP-IN 44
- DFHWS-XOP-OUT 44
- dispatchOrder.ear 111
- DispatchOrderAtomic.ear 298
- distributed routing 114–115
- DPL subset 94
- dynamic program routing 114, 116
- dynamic routing 116

E

- EAR file 111
- ECFG 109, 135, 301
- ECFG transaction 56
- endpoint
 - reference 207
- enterprise bean 266
- error
 - handling 17
- ExampleAppClient.ear 99
- ExampleAppDispatchOrder 110
- ExampleAppDispatchOrder.ear 111, 298
- Execution Diagnostic Facility (EDF) 137
- Extensible Markup Language (XML) 8, 12

F

- FAULT 44

G

governance 166

H

heuristic decision 232
HFS file system 86
high availability 113, 139
 with HTTP 113
 with WMQ 139
HTTP 85
hub configuration 252

I

in-doubt window 231
INPUT 44
INVOKE WEBSERVICE 53

J

J2EE
 connector 255
J2EE Connector Architecture (JCA) 314
Java Transaction API (JTA) 249
JCA 255
JCL 74, 76
 DFHLS2WS 74
 DFHWS2LS 76
JDBC 255
JDBC provider 275
JMS 255
JVM 101

M

MAPPING-LEVEL 72
message adapter 37
message handler 93, 130
 writing 93
Message handlers 62
message information headers 210
MTOM) 12
MTOM/XOP 42
mustUnderstand 16
MVS 114

N

NAMESPACES 44
Namespaces 14

non-terminal message 231
notification message 230

O

one-way message 230

P

Parallel Sysplex 114
participant 233, 252
PGMINT 76
PGMINT=CONTAINER 75
PGMNAME 76
PIPELINE 35, 58
 defining 96, 107, 131, 150
pipeline 92, 130
 configuration file 62
 customizing 92
 message handler 62
pipeline alias transaction 105
PIPELINE-ERROR 44
Pipelines 36
PL/I 38, 41
Port definition 30
prepared 229
presumed abort 199
problem
 determination 116

R

ReadOnly 229
recoverable 199
Redbooks Web site 318
 Contact us xiii
reference
 parameter 209
 property 209
Register
 request 220
 response 222
remote procedure call (RPC) 18
replay 229
REQMEM 76
REQUEST-BODY 44
RequestStream 248
RESPMEM 76
RESPONSE-BODY 44
RESPWAIT 53, 59

rollback 199, 228
RPC 18
 style 32

S

sample application 88, 145, 255
scalability 113
security
 CICS 193
 z/OS 193
Service broker 5
service definition 157
service deployment lifecycle 157
 active 158
 deprecate 158
 plan 158
 sunset 158
 test 158
service message model 160
service migration 159
service monitoring 161
service ownership 161
Service provider 5, 49
service provider 88
service registry 160
 in SOA governance 160
service requester
 installing 99
 timeout considerations 136
Service requestor 5
service security 162
service testing 162
service versioning 159
service-oriented architecture 3–4
SIT parameters 90
 TCPIP 90
SNIFFER 118, 256
SOA governance 163, 165–167
 defined 154
 service definition 157
 service deployment lifecycle 157
 active 158
 deprecate 158
 plan 158
 sunset 158
 test 158
 service message model 160
 service migration 159

service monitoring 161
service ownership 161
service registry 160
service security 162
service testing 162
service versioning 159
SOAP 9
 binding 31
 body 17, 285
 inbound data conversion 49
 outbound data conversion 49
 communication styles 18
 document 18
 RPC 18
 encodings 18
 literal 19
 SOAP encoding 18
 envelope 13, 15
 envelope builder 37
 envelope parser 37
 fault 17
 headers 15, 284
 intermediary 16
 introduction 13
 message 69
 message handler 37
 messaging mode 19
 MustUnderstand 16
 namespaces 14
 request 37
 SOAP for CICS feature 36
 SOAP Message Transmission Optimization
 Mechanism (MTOM) 42
 SOAP-ACTION 44
 SOAPFAULT
 commands
 ADD 64
 CREATE 64
 DELETE 64
 SOAPFAULT commands 64
 SOCKETCLOSE 91
 standards
 JCA 314
 JTA 249
 WS-Addressing 197
 WS-Atomic Transaction 197
 WS-Coordination 197
 static binding 168
 Sysplex Distributor 114, 141

T

- TARGET-TRANID 44
- TARGET-URI 44
- TARGET-USERID 44
- TCP/IP 233
 - load balancing 114
- TCPIP, SIT parameter 90
- TCPIPSERVICE 47, 90
 - attributes
 - PORTNUMBER 91
 - PROTOCOL 91
- terminal message 230
- testing 102, 112, 137, 277, 305
- tracing
 - CICS 117
- transactional scope 201
- transactions
 - atomic 205, 217
 - enabling CICS support 235
 - business activity 217
 - classic 198–199
 - two phase commit 199
- transport 123
- two phase commit (2PC) 199
 - presumed abort 200
 - walkthrough 201
 - with WS-AT 228

U

- UDDI
 - Universal Description, Discovery, and Integration 11
- Unicode 89
- URI 76, 135
- URIMAP 35, 57, 133
 - defining 98
- User containers 63
- USER-CONTAINERS 44

W

- Waldo 198
- Web service
 - Interoperability 11
 - properties 7
- Web services 3
- Web Services Description Language 167
 - defined 164
 - publishing from z/OS 173

- retrieving from z/OS 179
- WEBSERVICE 35
 - dynamically installing 97
- WebSphere
 - MQ 123
 - service requester
 - deploying 100, 110–111
 - using HTTP 99, 109
 - WebSphere Application Server
 - administrative console 100
 - connection pooling 101
 - WebSphere Developer for zSeries (WebSphere Developer) 77
 - WebSphere MQ 52, 75
 - see WMQ
 - WebSphere Service Registry and Repository
 - See also service registry and repository
 - SOA governance
 - See also SOA governance
- WMQ 124
 - configuration 126
 - defining queues 127
 - trigger process 128
 - shared queues 140–141
- WS-Addressing 197, 206
 - CICS Registration Service endpoint 213
 - endpoint reference 207
 - specification 206
- WSATHND 260
- WS-Atomic Transaction (WS-AT) 197, 203, 224
 - Completion protocol 225
 - daisy chain test scenario 298
 - testing 305
 - enabling CICS support 236, 238
 - enabling WebSphere support 249, 266
 - resynchronization processing 231
 - simple test scenario 257
 - testing 277
 - specification 216
 - Two-Phase Commit protocol 227
 - Durable 2PC 227
 - Volatile 2PC 227
 - typical scenario 255
 - walkthrough 203, 229
- WSBind file 68
- WS-Business Activity specification 216
 - unsupported by CICS 217
- WS-Coordination 197
 - Activation service 214

- Coordination service 214
- Protocol services 214
- Registration service 214
- specification 213
- walkthrough 222

WSDL 19

- binding 20, 29
- bindings 31
- definition 25
- document 20
 - anatomy 21
- message 20, 26
- namespaces 24
- operation 20, 27
- port 20, 30
- port type 20, 27
- service 20
- service definition 30
- SOAP binding 31
- type 20
- types 25
- Web Services Description Language 10

WSDL 2.0 41

X

- XML parsing 45
- XML Schema Definition
 - defined 164
- XML-binary Optimized Packaging (XOP) 42

Z

- z/OS 114
 - Communications Manager 114
 - Parallel Sysplex 114
- z/OS UNIX System Services HFS file 57



Implementing CICS Web Services

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Implementing CICS Web Services



Configuring and securing Web services in CICS Transaction Server

Enabling MTOM support in CICS Transaction Server

SOA governance and WSRR

Today more and more companies are embracing the principles of on demand business by integrating business processes end-to-end across the company and with key partners, enabling them to respond flexibly and rapidly to new circumstances. The move to an on demand business environment requires technical transformation, moving the focus from discrete applications to connected, interdependent information technology components.

Open standards such as Web services enable these components to be hosted in the environments most appropriate to their requirements, while still being able to interact easily, independent of hardware, run-time environment, and programming language.

The Web services support in CICS Transaction Server Version 3 enables your CICS programs to be Web service providers and requesters. CICS supports a number of specifications including SOAP Version 1.1 and Version 1.2, and Web services distributed transactions (WS-Atomic Transaction).

This IBM Redbooks publication describes how to configure CICS Web services support for HTTP-based and WebSphere MQ-based solutions, and demonstrates how Web services can be used to integrate J2EE applications running in WebSphere Application Server with COBOL programs running in CICS.

The book begins with an overview of Web services standards and the Web services support provided by CICS TS V3. Complete details for configuring CICS Web services using both HTTP and WebSphere MQ are provided next. We concentrate on the implementation specifics such as security, transactions, and availability.

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

SG24-7657-00

ISBN 0738431753