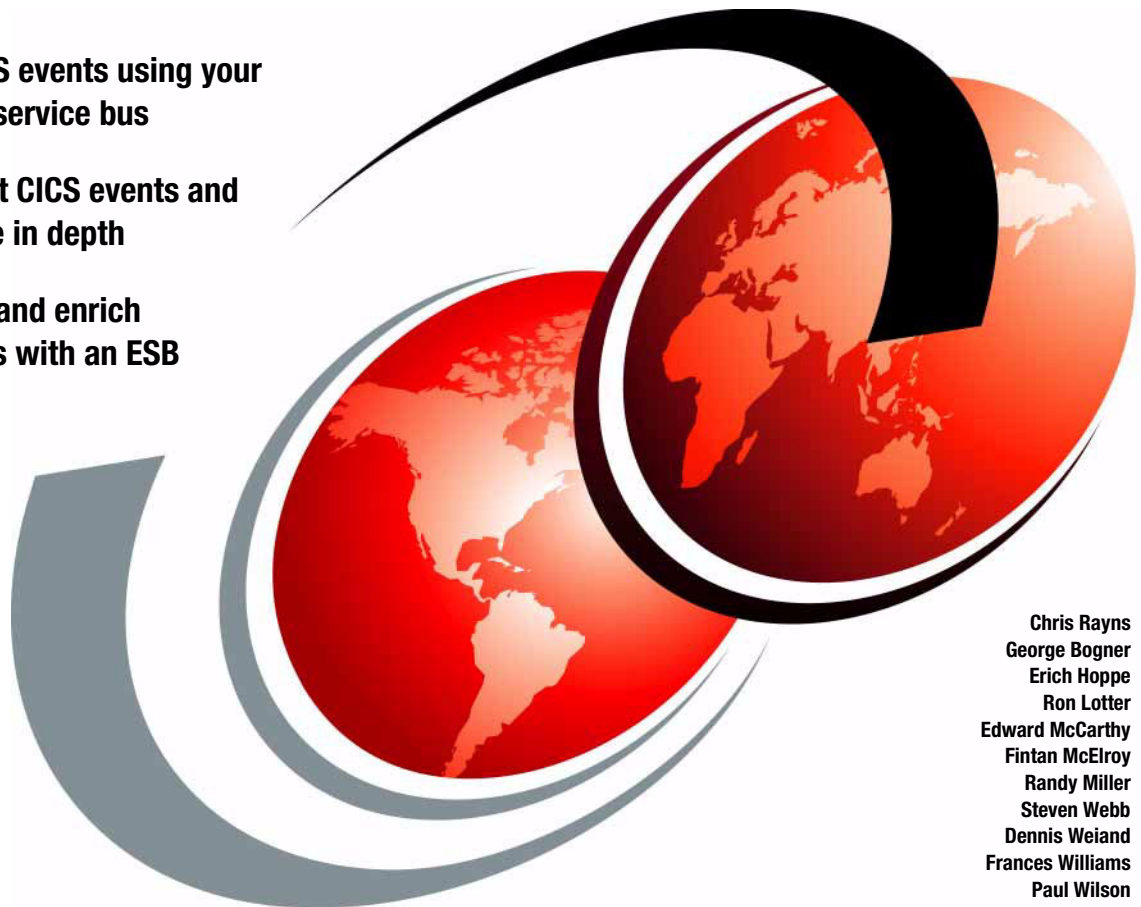


Leveraging CICS Events with an ESB

Exploit CICS events using your enterprise service bus

Learn about CICS events and governance in depth

Transform and enrich CICS events with an ESB



Chris Rayns
George Bogner
Erich Hoppe
Ron Lotter
Edward McCarthy
Fintan McElroy
Randy Miller
Steven Webb
Dennis Weiland
Frances Williams
Paul Wilson



International Technical Support Organization

Leveraging CICS Events with an ESB

August 2010

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

First Edition (August 2010)

This edition applies to Version 4, Release 1, of IBM CICS Transaction Server for z/OS (5655-S97).

© Copyright International Business Machines Corporation 2010. 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 who wrote this book	xii
Now you can become a published author, too!	xv
Comments welcome	xv
Stay connected to IBM Redbooks	xvi
Part 1. Introduction	1
Chapter 1. Introduction to event processing	3
1.1 Events	4
1.2 Event processing	4
1.2.1 Simple events	5
1.2.2 Complex events	5
1.3 Why you need events	5
1.4 Business application events and system events	6
1.5 IBM solutions for business event processing	7
1.5.1 CICS Transaction Server	7
1.5.2 CICS Explorer	8
1.5.3 WebSphere Business Events	8
1.5.4 WebSphere Business Monitor	9
1.5.5 WebSphere Enterprise Service Bus	10
1.5.6 WebSphere Message Broker	11
1.6 DataPower	12
1.7 WebSphere Process Server	12
1.8 IBM solution for system events	15
1.8.1 Tivoli OMEGAMON XE for CICS on z/OS	15
1.9 Solutions reviewed	15
Chapter 2. CICS event processing	17
2.1 Why emit events from CICS applications	18
2.2 How CICS event processing works	18
2.3 CICS Event Binding editor	19
2.4 Event specification	21
2.5 Capture specification	22
2.5.1 Capture point	23
2.5.2 Filter and predicates	25

2.5.3 Information sources	26
2.6 Event binding	29
2.7 Non-invasive events or SIGNAL EVENT	30
2.7.1 Automatic capture specification for SIGNAL EVENT	31
2.8 Event processing adapters	34
2.8.1 Temporary storage queue EP adapter	36
2.8.2 Transaction start EP adapter	36
2.8.3 WebSphere MQ (WMQ)	37
2.8.4 Custom (user-written) EP adapter	38
2.9 Exporting event schema or copybook	38
2.10 EP adapter advanced options	39
2.10.1 Dispatch priority	40
2.10.2 Transaction ID	40
2.10.3 User ID	40
2.10.4 System ID	40
2.10.5 Transactional events	41
2.11 Bundles	41
2.12 Deploy a bundle to zFS	42
Chapter 3. Integrating event processing and enterprise service bus	45
3.1 Conceptual architecture	46
3.2 Why use an enterprise service bus as the Event Bus	53
Part 2. Environment	55
Chapter 4. Overview of the application and business scenarios	57
4.1 Objectives	58
4.2 Overall architecture	58
4.3 The catalog manager example application	59
4.4 ESB structure	65
4.5 Using WebSphere Enterprise Service Bus as the ESB	66
4.6 Using DataPower as the ESB	67
4.7 Using WebSphere Message Broker as the ESB	68
4.8 Event scenarios used by our team	68
4.8.1 Successful events	70
4.8.2 Failure events	72
4.8.3 Scenario 1: Successful order event	74
4.8.4 Scenario 2: Multiple high value orders in three days event	78
4.8.5 Scenario 3: Failed order due to insufficient stock event	80
4.8.6 Scenario 4: Multiple insufficient stock failures event	83
4.9 Testing each scenario	87
Part 3. Scenarios	89

Chapter 5. WebSphere Enterprise Service Bus business scenario	91
5.1 Environment overview	92
5.2 Environment configuration	93
5.2.1 CICS configuration	93
5.2.2 WebSphere Business Events configuration	104
5.2.3 WebSphere Business Monitor configuration	106
5.2.4 WebSphere Process Server configuration	106
5.2.5 WebSphere Enterprise Service Bus configuration	106
5.3 Scenario 1	112
5.3.1 ESB transformation	113
5.3.2 Test results	121
5.4 Scenario 2	121
5.5 Scenario 3	121
5.5.1 ESB transformation	122
5.5.2 Test results	123
5.6 Scenario 4	124
5.7 Problems encountered, hints, and tips	124
5.7.1 Cross-cell Pub/Sub in WebSphere Application Server V7	124
5.7.2 WebSphere Business Events plug-in for WebSphere Integration Developer	125
5.7.3 Modifications to WebSphere Business Events event schema	127
5.7.4 When WebSphere Business Monitor and WebSphere Enterprise Service Bus are in the same cell	128
5.7.5 CBE details in the WebSphere Application Server run time	128
5.7.6 Deploying to the next test stage	129
5.8 Summary	130
 Chapter 6. WebSphere Message Broker business scenario	 131
6.1 Environment overview	132
6.2 Configuring the environment	132
6.2.1 Configuring CICS	133
6.2.2 WebSphere Message Broker configuration	143
6.2.3 WebSphere Business Events configuration	145
6.2.4 WebSphere Business Monitor configuration	145
6.2.5 WebSphere Process Server configuration	145
6.3 Scenario 1	146
6.3.1 WebSphere Message Broker transformation	146
6.4 Scenario 2	167
6.5 Scenario 3	168
6.5.1 Scenario 3 overview	168
6.6 Scenario 4 test	181
6.7 Problems encountered, hints, and tips	182
6.8 Summary	182

Chapter 7. DataPower business scenario	185
7.1 Environment overview	186
7.2 Environment configuration	187
7.2.1 CICS configuration	187
7.2.2 DataPower configuration	198
7.2.3 Scenario 1	204
7.2.4 Scenario 3	225
7.3 Hints and tips	232
7.3.1 Probe for debug	232
7.3.2 External tools to help create stylesheets	232
7.4 Summary	233
Chapter 8. Scenario flow	235
8.1 Scenario 1	236
8.2 Scenario 2	239
8.3 Scenario 3	242
8.4 Scenario 4	245
Chapter 9. WebSphere Business Events scenario	249
9.1 Development setup and WebSphere Business Events tooling	250
9.1.1 WebSphere Business Events	250
9.1.2 WebSphere Business Events development tooling	250
9.1.3 WebSphere Business Events scenario description	251
9.1.4 Building the WebSphere Business Events project	251
9.1.5 Configuring and testing WebSphere Business Events	264
9.1.6 Tips and hints for developing with WebSphere Business Events	269
Chapter 10. WebSphere Business Monitor	271
10.1 Configuring WebSphere Business Monitor	272
10.1.1 Defining a CEI bus destination in WebSphere Business Monitor	272
10.1.2 Establishing the MQ to CEI link	273
10.1.3 Defining the MQ channels and queues	273
10.2 Designing the monitor model	276
10.2.1 Creating the monitor project and model	277
10.2.2 Importing CBE schema	277
10.2.3 Defining the monitor details model	277
10.2.4 Defining the KPI model	280
10.3 Creating the Business Space dashboard	281
10.4 Viewing the CICS CBE monitor dashboard	283
10.4.1 Sending test events with the Integrated Test Client	283
10.4.2 Successful order instances	291
10.4.3 Insufficient stock instances	292
10.4.4 Total successful and failed orders	293
10.4.5 Event rate and average meantime between failure orders	293

10.5 Summary	294
Chapter 11. WebSphere Process Server	295
11.1 Process	296
11.1.1 Designing the process flow	296
11.2 Building the process	298
11.2.1 Products we used	298
11.2.2 CICS Web Services Description Language	298
11.2.3 Starting WebSphere Integration Developer V7	299
11.2.4 Creating a business integration project	299
11.2.5 Defining business objects	301
11.2.6 Creating a new business process	306
11.2.7 Variables	312
11.2.8 Adding a snippet	314
11.2.9 Time for a first test run	318
11.2.10 Adding a web service call to get item details	324
11.2.11 Adding the ForEach activity	329
11.2.12 Defining new Business Objects	331
11.2.13 Adding a global variable to hold the total cost	332
11.2.14 Adding the Assign activity	333
11.2.15 Adding an intermediary interface	336
11.2.16 Connecting the process to the Web Service	341
11.2.17 Updating the mediation flow	344
11.2.18 Adding the invoke activity	348
11.2.19 Adding the snippet to calculate the cost	351
11.2.20 Testing the process	352
11.2.21 Adding the choice activity	353
11.2.22 Adding a reply activity	357
11.2.23 Adding a human to-do task	358
11.2.24 Completed process	362
11.2.25 Testing the completed process	364
11.2.26 Testing using the Integrated Test Client	366
11.2.27 Exporting the process wsdl	371
11.2.28 Exporting the ear file	373
11.2.29 Application to test the process	373
11.2.30 WebSphere Business Events to process the mediation flow	373
11.2.31 Supplied files	374
11.2.32 Summary	375
Part 4. Best practices	377
Chapter 12. Best practices	379
12.1 Plan and organize	380
12.1.1 Naming conventions	380

12.1.2	Transport	381
12.1.3	Communication formats	381
12.2	Governance	381
12.2.1	CICS events and governance	384
12.2.2	Artifacts	385
12.2.3	Audit and change control	387
12.3	Security considerations for CICS events	389
12.3.1	Development security	391
12.3.2	Resource security	392
12.3.3	Deployment security	395
12.3.4	Runtime security	395
12.4	Troubleshooting	397
12.4.1	Problems we encountered	397
Appendix A. Additional material		401
	Locating the web material	401
	Using the web material	402
	System requirements for downloading the web material	402
	How to use the web material	402
Related publications		403
	IBM Redbooks publications	403
	Other publications	403
	Online resources	403
	How to get IBM Redbooks publications	404
	Help from IBM	404

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:

AIX®	FileNet®	Rational®
CICS Explorer™	IBM®	Redbooks®
CICSplex®	IMS™	Redbooks (logo)  ®
CICS®	Lotus Notes®	System z®
DataPower device®	Lotus®	Tivoli®
DataPower®	MVS™	WebSphere®
DB2®	Notes®	z/OS®
Domino®	OMEGAMON®	zSeries®

The following terms are trademarks of other companies:

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

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.

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

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

Preface

Many functions exist to help you use an enterprise service bus (ESB) as the event bus:

- ▶ Transformation: Function that transforms the incoming event by translating or splitting it
- ▶ Enrichment: Function that enriches the content of events with reference data from multiple possible sources
- ▶ Validation: Function to provide validation against required criteria
- ▶ Pattern detection: Function that recognizes actual and retrospective patterns; a combination from possibly multiple events, characterizing a significant business situation
- ▶ Filtering: Stateless function that filters events based on their content; that is, the information that is carried by the message generated when the event happened
- ▶ Aggregation: Function that can group events as necessary
- ▶ Routing: Function that routes events to the destination based on various possible routing patterns, such as pre-established itinerary, calendar-based, subscription or “intelligent” routing decisions

In this IBM® Redbooks® publication, we show examples of using an ESB to transform and enrich an event received from Customer Information Control System (CICS®) Transaction Server. We also show an example of enriching an event.

This book contains four parts:

- ▶ Part 1, “Introduction” on page 1 introduces event processing. We explain what it is and why you need it. We also review the CICS TS implementation of event processing. We discuss enterprise service bus technology and how to integrate event processing with an ESB.
- ▶ Part 2, “Environment” on page 55 of the book focuses on our environment and the application we chose to use.

- ▶ Part 3, “Scenarios” on page 89 describes our scenarios with three separate ESBs along with information about the IBM WebSphere® Business Events, IBM WebSphere Business Monitor, and IBM WebSphere Process Server for z/OS® setup details:
 - Scenario 1: IBM WebSphere Enterprise Service Bus for z/OS business scenario
 - Scenario 2: IBM WebSphere Message Broker for z/OS business scenario
 - Scenario 3: DataPower® business scenario
 - Scenario flow
 - WebSphere Business Events
 - WebSphere Business Monitor
 - WebSphere Process Server
- ▶ Part 4, “Best practices” on page 377 of this book describes best practices:
 - Governance
 - Security
 - Best practices for performance

The team who 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 Project Leader at the ITSO Poughkeepsie Center in New York. Chris specializes in security and writes extensively about all areas of CICS. Before joining the ITSO, he worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

George Bogner is a Software IT Specialist working in IBM Sales and Distribution supporting the CICS Transaction Server product suite. George has worked at IBM for 25 years, specializing in the DB/DC area working with CICS, IMS™, and DB2®, supporting client accounts in IBM Global Services. He currently works out of Raleigh, North Carolina, supporting North American clients by providing CICS seminars, proofs of technology (POT), proofs of concept (POC), and consulting services for CICS-related topics.

Erich Hoppe is a CICS Level 2 Technical Support Software Engineer in Raleigh, NC. He has two years of experience in Level 2 technical support. He holds a Bachelor of Science degree in Computer Engineering from Pennsylvania State

University in State College, PA. His primary areas of focus are CICS event processing, CICS Explorer™, storage, and CICS Web Services.

Ron Lotter is a Senior Software Engineer in the Software Services for WebSphere organization in Raleigh, NC. He has worked for IBM for 27 years holding various management and technical positions and has 11 years of experience with the WebSphere product family. He holds a Masters degree in Electrical Engineering from Case Western Reserve University in Cleveland, Ohio. His areas of expertise include WebSphere Application Server on z/OS and WebSphere Process Server on z/OS, as well as Service Component Architecture (SCA) and Java™ 2 Platform, Enterprise Edition (J2EE) development.

Edward McCarthy joined IBM Global Technology Services in 2000 and designed and built WebSphere Application Server environments on Windows®, AIX®, Linux®, zLinux and z/OS for various clients. For the past two years, he has worked in Software Group providing technical support to sales staff in the Asia/Pacific region focusing on WebSphere on System z®. Prior to joining IBM, he worked as a CICS and MQ system programmer for an Australian Government department. He has written several ITSO Redbooks publications and presented at various technical conferences.

Fintan McElroy is a Certified Consulting IT Specialist working for IBM Software Services for WebSphere (ISSW), whose mission it is to assist IBM clients in their adoption of the WebSphere middleware product suite. Fintan has over 15 years IT experience in a variety of roles, including infrastructure architecture and design, application architecture, business analysis, and application development. He has been a leading proponent of business process management (BPM) with service-oriented architecture (SOA) since its inception and before that had been a practitioner of object-oriented architecture and design methods. He currently provides consultancy to IBM clients on enabling agility in their enterprise through the adoption of IBM WebSphere Business Process Management solutions.

Randy Miller is an Advisory Software Engineer in the Application and Integration Middleware division of IBM. He has held various hardware-related and software-related positions in IBM over his 31-year career. In 2001, Randy joined the Level 2 Support Team for WebSphere Message Broker. His current responsibilities include working directly with clients to help to resolve issues related to WebSphere Message Broker.

Steven Webb is a CICS Knowledge Engineer (KE) in the United States. He has 14 years of experience with CICS and was a CICS Level 2 Technical Support Software Engineer for 12 years where his primary areas of focus were CICS Web services, CICS Web support, and Simple Object Access Protocol (SOAP). As a KE, he leads the creation, maintenance, management, and delivery of support-related content for the CICS Transaction Server and CICS Tools

products. He holds a Bachelor of Science degree in Computer Science from Michigan Technological University.

Dennis Weiland is a Technical Sales Specialist at the IBM Dallas Systems Center. Currently, Dennis works primarily with Web services, Web 2.0, Events, and Java as they relate to CICS, plus the CICS Transaction Gateway. He holds a Masters degree in Computer Science from Tarleton State University in central Texas.

Frances Williams is a Senior Consultant with IBM Systems & Technology Group Infrastructure Solutions in the United States. She has over 20 years of experience in the IT field as an application IT architect. Her focus is on z/OS platform technologies, which include the WebSphere suite of products, CICS, and many development languages.

Paul Wilson is an Advisory Software Engineer at IBM in the United States with over 25 years of design and development experience. His current assignment is with the WebSphere Business Events development team at the Mass Lab in Littleton, Massachusetts. His broad experience includes WebSphere, DataPower, Lotus® Notes®, and Domino®, object-oriented databases, and numerous operating systems internals. Paul holds degrees in Education and Computer Science from the University of Massachusetts and the Wentworth Institute of Technology.

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

Richard M. Conway
International Technical Support Organization, Raleigh Center

Jefferson Lowrey, WebSphere Message Broker Level 2 Customer Support
IBM US

Peter MacFarlane, Software Developer
IBM Hursley

Catherine Moxey, IBM STSM CICS TS
IBM Hursley

Peter Crocker, Development Lead, Architect, WebSphere Business Events
IBM Hursley

Albert Chung, Software Developer
IBM Raleigh

Satyan Bodla, Advanced Technical Sales Specialist
IBM Dallas

Luis Sanchez, Software Developer
IBM Raleigh

Steve Bolton, Software Developer
IBM Hursley

The team of authors of *Implementing Event Processing with CICS*,
SG24-7792-00

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and client satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 publications in one of the following ways:

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

ibm.com/redbooks

- ▶ Send your comments in an email 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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent IBM Redbooks publications with RSS feeds:
<http://www.redbooks.ibm.com/rss.html>



Part 1

Introduction

In this part of the book, we introduce event processing, how Customer Information Control System (CICS) implements event processing, and how you can integrate event processing in your enterprise service bus (ESB).



Introduction to event processing

In this chapter, we define event processing and explain the distinctions between simple and complex events.

We show why event processing is useful to your business, describe business events, and conclude with a summary of IBM products for event processing.

1.1 Events

An *event* is something that happens that is significant to a system, for example:

- ▶ Open a bank account.
- ▶ Sense a temperature change.
- ▶ Click a mouse button.
- ▶ Browse an inventory without making a purchase.
- ▶ Observe an unusual history of purchases on a credit card.

For this book, *event* is the term used to describe an electronic message indicating a change in the state of an enterprise. An event has a name and usually data, and it is sometimes referred to as the *event payload*.

Events are generated and processed asynchronously in near real time. The processing of an event is decoupled from the computer operations that caused it to be emitted.

1.2 Event processing

Event processing is the capture, enrichment, formatting, and emission of events, the subsequent routing and any further processing of emitted events (sometimes in combination with other events), and the consumption of the processed events.

Events can be produced throughout a business enterprise. At the edges of the enterprise, events can be detected by sensors. In the enterprise network, events can be produced when business processes start and then either complete or fail. The activity of the enterprise and its business can be monitored and changed as a result of events. Event processing consists of three main steps:

- ▶ Event sources emit events into the event processing system. Examples of event sources are simple radio frequency identification (RFID) sensors and actuators, business flows, and Customer Information Control System (CICS) applications. The event processing system can perform a variety of actions on events:
 - Simple enriching of the event (for example, adding a time stamp to the event data).
 - Adding information about the source of the event.
 - Processing multiple simple events, from multiple event sources, against event patterns to produce a new derived event. Processing of this kind is often referred to as *complex event processing*.
- ▶ The event resulting from processing is made available for consumption.

- ▶ The event consumer reacts to the event. The event consumer might be simple and just update a database or a visual dashboard with the data carried with the event, or it might carry out new business processing based on the event.

We categorize events as simple or complex.

1.2.1 Simple events

The first three examples in the list in 1.1, “Events” on page 4 are *simple events*. For many years, organizations have used simple event processing to detect and respond to discrete events (for example, when bank customers open a new account, sending them a welcome letter that explains other facilities that are provided by the bank).

1.2.2 Complex events

The last two examples in the list in 1.1, “Events” on page 4 describe what we call *complex events* or *derived events* that are obtained by looking at the patterns of simple events over time. To detect customers browsing an inventory without making a purchase, we can emit events when the customers perform the following actions:

- ▶ Browse an item in the inventory
- ▶ Buy an item

Detecting the patterns from these simple events can produce the complex event as shown in example four in 1.1, “Events” on page 4.

In large organizations, tens of millions of events occur every day, but not all events are of equal importance. Greater insight can be obtained when a pattern of related or seemingly unrelated events from one or more sources is detected and responses to that pattern are coordinated. Considerable complexity, time, and cost can be involved in writing custom code for such a solution. This complexity, time, and cost can be replaced by general software technology that is designed to detect event patterns and coordinate responses, which is known as *complex event processing software*.

1.3 Why you need events

Events allow businesses to be more responsive and flexible and to address governance and compliance concerns.

The mainstream adoption of service-oriented architecture (SOA) has opened new opportunities for highly responsive business solutions. SOA brings greater flexibility to business processes and helps bring business and IT in line with each other. Enterprises are challenged by seeking to maximize SOA solution advantages (such as speed to market), in addition to complying with business controls, industry standards, and government legislation.

Business governance and compliance are increasingly important in many industries. These terms cover a crucial range of issues including financial transparency, information privacy, and process control. The Sarbanes-Oxley Act, the Health Insurance Portability and Accountability Act (HIPAA), or Basel II and their associated information requirements are a few of the standards required for business compliance and governance.

Governance describes a formalization of the decision-making processes within an organization. It can cover many aspects of business and depends on accurately maintaining and auditing which decisions can be freely made and which decisions need specific approvals. It also determines who can make the decisions.

Compliance is about ensuring adherence to mandated standards and governance policy. Compliance includes the definition of information about which governance decisions are based. Also, it includes maintaining accurate operational control to ensure that business application execution meets the required enterprise, industry, and government standards.

1.4 Business application events and system events

CICS event processing is designed for business application events. All events in an enterprise can be seen as having a business consequence and so can be described as business events. Whether the event and event processing are specified by a line-of-business manager or an IT programmer, the event relates to the business application or system that is running. It therefore has business relevance. For example, an event that implies the imminent failure of a system running an order-processing application can be considered relevant to the line-of-business manager. In general, the procedures that emit and consume business events are distinct from those procedures that process IT infrastructure-related events.

System-level events generally have a technical focus and relate to monitoring operating system, application execution, and middleware running on the system. The event data is usually equally technical, specifying the identifiers of the resources under observation.

Business application events are usually related to higher-level business processes. They specify conditions in terms of what the application does for the business. For example, contrast the business event “a new order has been placed” with the system events that are used to assess the compute time for processing the order: inventory file opened, order information storage released, and so forth.

The consumers of business events are often required to be independent of the implementation specifics of the systems that emit the events. For example, it is possible for one event consumer to process events from several disparate ordering systems and provide a single consolidated view of the business application’s state.

In general, we consider system events and business events as distinct event types with separate software solutions and audiences.

1.5 IBM solutions for business event processing

The IBM software portfolio enables a range of options for integrated processing of business events.

1.5.1 CICS Transaction Server

IBM CICS Transaction Server (TS) business applications are the major source of business information in most large enterprises. The CICS run time detects instances of events that are enabled, and it captures the events and payload without the need to make application code changes. CICS event processing is a core component of the CICS run time and provides all the qualities of service that you expect of CICS. When CICS captures events, it carries out specified filtering, enriches the event with information about the application context in which it occurred, formats the event and routes it to the appropriate event consumer.

You can emit events in formats that are suitable for consumption by WebSphere Business Events, WebSphere Business Monitor, and other consumers.

CICS event processing support is extensible with options for customization.

See *CICS Transaction Server for z/OS, Version 4 Release 1* at the following web page:

<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>

1.5.2 CICS Explorer

IBM CICS Explorer is the new face of CICS and an integration point for CICS tooling with rich CICS views, data, and methods. It provides a common, intuitive, and Eclipse-based environment for architects, developers, administrators, system programmers, and operators.

CICS Explorer has the following features:

- ▶ Provides task-oriented views giving integrated access to a broad range of data and control capabilities
- ▶ Has powerful, context-sensitive resource editors and manages all CICS resources, including those resources for event processing
- ▶ Is an integration point for IBM CICS Transaction Server for z/OS, IBM CICS Tools Family, IBM CICS Transaction Gateway for z/OS, IBM Problem Determination Tools Family, and IBM Rational® Tools

For CICS event processing, the CICS Explorer contains the CICS Event Binding Editor, which you use to perform the following tasks:

- ▶ Defines events to be emitted and their payload data
- ▶ Specifies to the CICS run time how to detect when the events occur
- ▶ Indicates how events are formatted and routed
- ▶ Deploys the event definitions to the System z file system (zFS) for installation into CICS

For more information, see the CICS Explorer at the following web page:

<http://www-01.ibm.com/software/htp/cics/explorer/>

1.5.3 WebSphere Business Events

IBM WebSphere Business Events is an IBM software product designed to support business event processing by meeting the high-volume demands and processing required across industries and application domains. IBM WebSphere Business Events provides graphical, codeless user interfaces that simplify implementation and empower business users to develop and maintain event processing logic.

IBM WebSphere Business Events consists of the following basic constructs:

- ▶ Connectivity to business events
- ▶ Event processing engine for evaluating and detecting event patterns
- ▶ Initiation of business responses (actions)

Business events can exist anywhere within the extended computing infrastructure, both inside and outside the firewall. Events can be communicated directly between systems or pushed into the communications backbone for use by any system. A message-based publish-or-subscribe or request-or-reply transport, such as IBM WebSphere MQ, is an ideal transport infrastructure for event processing.

Based on user definitions, the WebSphere Business Events processing engine detects and sifts through the mass of events occurring across the information infrastructure, identifying only those events and patterns of interest. Upon detecting a defined event or pattern (*actionable situation*), the engine initiates one or more business responses (*actions*).

Responses range from sending electronic alerts to initiating the execution of follow-on processes. These actions are communicated directly to systems (or over the communications backbone), indicating that an actionable event or pattern has been detected.

For more information, see WebSphere Business Events at the following web page:

<http://publib.boulder.ibm.com/infocenter/wbevents/v6r2m1/index.jsp>

1.5.4 WebSphere Business Monitor

IBM WebSphere Business Monitor is a comprehensive business activity monitoring (BAM) product that provides business users and managers with a real-time and end-to-end view of business processes, events, and operations. WebSphere Business Monitor aggregates and correlates events into metrics that give objective measurements about the status of business processes.

WebSphere Business Monitor shows business users real-time information about the performance of critical business processes. It offers user-friendly and customizable dashboards that enable complete insight into the business flowing through the system. These dashboards can calculate and display key performance indicators (KPIs) and metrics derived from the following sources:

- ▶ Business processes
- ▶ Business activity data
- ▶ Business events

Business users can view these KPIs, metrics, events, and alerts through various means, including lightweight web interfaces, Smartphones, corporate portals, and on desktops. These options give business users immediate actionable information and insight into their business operations to mitigate risk and take advantage of opportunities.

The following list details many of the real-time events from which WebSphere Business Monitor enables collection:

- ▶ CICS
- ▶ IBM Business Process Management (BPM) Suite and Connectivity portfolio:
 - WebSphere Process Server
 - IBM FileNet® P8 Business Process Manager
 - WebSphere MQ Workflow
 - WebSphere Business Events
 - WebSphere Message Broker
 - WebSphere Enterprise Service Bus
 - WebSphere DataPower Integration Appliance XI50
 - WebSphere Partner Gateway
- ▶ Events and data through the use of WebSphere adapters and the IBM Connectivity portfolio from the following sources:
 - Oracle
 - SAP
 - Siebel
 - Other enterprise resource planning (ERP) and customer relationship management (CRM) applications
- ▶ Other third-party applications through the IBM Connectivity portfolio

A bidirectional flow of events between WebSphere Business Monitor and IBM WebSphere Business Events is enabled. Clients can use a single dashboard to view the performance of business processes through KPIs and view any alerts generated by IBM WebSphere Business Events to get real-time insight into what is happening within the organization. Similarly, the bidirectional event flow enables clients to feed any alerts about processes or business that is generated by WebSphere Business Monitor to IBM WebSphere Business Events for detecting patterns within those alerts, which might otherwise go undetected, and to initiate follow-on processing.

See WebSphere Business Monitor at the following web page:

<http://www-01.ibm.com/software/integration/wbimonitor/>

1.5.5 WebSphere Enterprise Service Bus

IBM WebSphere Enterprise Service Bus provides Web services connectivity and Java Message Service (JMS) messaging, improving flexibility through the adoption of service-oriented interfaces.

WebSphere Enterprise Service Bus provides a smart approach to SOA, delivering a standards-based connectivity and integration solution that allows you

to create and deploy interactions quickly and easily between applications and services, with a reduced number and complexity of interfaces.

WebSphere Enterprise Service Bus provides the following features:

- ▶ Offers easy-to-use tools that require minimal programming skills
- ▶ Is simple to install, configure, build, and manage
- ▶ Supports hundreds of independent software vendor solutions through WebSphere adapters
- ▶ Reconfigures dynamically to meet changing business processing loads
- ▶ Provides easy interactions with any JMS and HTTP applications

See WebSphere Enterprise Service Bus at the following web page:

<http://www-01.ibm.com/software/integration/wsesb/>

1.5.6 WebSphere Message Broker

IBM WebSphere Message Broker is built for universal connectivity and transformation in heterogeneous IT environments. It distributes information and data generated by business events in real time to people, applications, and devices throughout your extended enterprise and beyond.

WebSphere Message Broker offers the following features:

- ▶ Provides a smart approach to SOA, extending the reach of your business beyond your firewall by supporting a broad range of multiple transport protocols and data formats
- ▶ Integrates multiple applications, networks, and device types using a platform-independent-based enterprise service bus (ESB) that lets you conduct business reliably and securely
- ▶ Increases business agility and flexibility, extending easily to a federated ESB model, while reducing development costs by separating integration logic from applications
- ▶ Improves the flow of information around the business, moving away from hard-coded point-to-point links to more flexible distribution mechanisms, such as publish/subscribe and multi-cast
- ▶ Uses a simple programming model for connectivity and mediation, including a robust set of pre-built mediation functions and ways to customize mediations
- ▶ Exploits the industry-leading WebSphere MQ messaging infrastructure

- ▶ Supports transformation options with graphical mapping, Java, ESQL, Extensible Stylesheet Language (XSL), and WebSphere Transformation Extender
- ▶ Delivers extensive administration and systems management facilities for developed solutions

See WebSphere Message Broker at the following web page:

<http://www-01.ibm.com/software/integration/wbmessagebroker/>

1.6 DataPower

The IBM WebSphere DataPower Integration Appliance XI50 is a complete hardware platform for delivering highly manageable, more secure and scalable integration solutions, which offers the following functions:

- ▶ Simplifies infrastructure with the Application Optimization option to provide intelligent back-end application workload balancing (available only for 9235 machine type)
- ▶ Bridges to Web 2.0 technologies with JavaScript Object Notation (JSON) filtering and validation, supports Representational State Transfer (REST) verbs, and converts and bridges REST and Web services
- ▶ Offers rapidly configurable Web Application Firewall security to protect against cross-site scripting, SQL injection, and a wide variety of XML threats
- ▶ Provides fast and flexible application integration with declarative any-to-any transformations between disparate message formats
- ▶ Reduces integration costs with wirespeed mediation, protocol bridging, transport mediation and content-based message routing
- ▶ Enables extreme reliability by securing services at the network layer with advanced XML/SOAP/WS-Web services processing and policy enforcement
- ▶ Lowers operational costs with native connectivity to existing access control, monitoring, and database and management systems and processes
- ▶ Offers standards-based, centralized governance and security, with support for a broad array of standards, such as WS-Security and WS-SecurityPolicy

1.7 WebSphere Process Server

IBM WebSphere Process Server is a high-performance business process automation engine to help form processes that meet your business goals.

Built on open standards, it deploys and executes processes that orchestrate services (people, information, systems, and trading partners) within your SOA or non-SOA infrastructure.

WebSphere Process Server helps increase efficiency and productivity by automating complicated processes that span people, partners, and systems. It helps cut costs by enabling flexible business processes with reusable assets, reducing the need to hard-code changes across multiple applications. IT extends the value of core applications by centralizing business processes and sharing them across the enterprise to maximize resources and increase return on investment (ROI). WebSphere Process Server provides strong support for human workflow and enables rapid process changes, providing the business agility required to compete in emerging markets by using resources efficiently. It accelerates time to value by enabling rapid change and reconfiguring existing IT assets without redeployment.

WebSphere Process Server offers the following features:

- ▶ Increases business flexibility with powerful human workflow capabilities:
 - Reacts dynamically to changing business requirements with the ability to install new versions of a process and migrate running processes to a new version
 - Supports additional human workflow scenarios, including parallel approval with voting and result aggregation
 - Offers new human tasks and workflow widgets that deliver more scenarios, such as human task, workflow, escalation management, ad hoc multi-column filtering, and adaptive paging
 - Provides richer capabilities to manage in-flight processes, such as modifying ownership of a process instance and enhanced activity repair capabilities
- ▶ Empowers users and accelerates productivity across all process roles:
 - Faster server start-up time and deployment of BPM solutions from WebSphere Business Modeler and WebSphere Integration Developer
 - Improved user experience for Interactive Process Design scenario with faster deployment
 - Enhanced operational visibility with new and improved role-based widgets, enabling better service monitoring and health determination
 - Improved problem determination with consistent fault handling across Service Component Architecture (SCA) bindings and cross-component trace enhancements

- New widgets for solution administrators enable better module administration
- ▶ Accelerate time to value for implementing and deploying BPM solutions:
 - Simplifies system installation, including easier cluster configuration
 - Provides consistent, flexible, and independent BPM topology and database configuration and management
 - Simplifies handling of runtime environment outages with support for unexpected service downtime with “store and forward” capability to queue events until service is restored
 - Eases the process of loading or unloading static relationship data through a data import and export capability
 - Uses web-based forms rendered from Lotus Forms Server, in addition to the existing Lotus Forms Client rendering capabilities
- ▶ Enhancements for easier migration from WebSphere Business Integration heritage server solutions:
 - Enhances the maintainability of the generated Business Process Execution Language (BPEL) from migrated WebSphere InterChange Server repositories
 - Improves performance of WebSphere InterChange Server migration with an improved user experience for migrating large repositories
- ▶ Enhances support for open standards:
 - Java enhancements, including Java Enterprise Edition (EE) 5, Enterprise JavaBeans (EJB) 3.0, Java Persistence API (JPA), Java Development Kit (JDK) 6 support, and enhanced Java integration
 - Web services enhancements, including WS-Addressing, Attachments, Kerberos token profile, and WS-Policy support
 - OpenSCA support, enhanced Open Service Gateway initiative (OSGi) support, and enhanced XML fidelity
- ▶ Platform alignment and currency:
 - Exploits and extends WebSphere Application Server V7.0, providing enhanced standards support, simplified system installation and administration, and enhanced WebSphere MQ V7 integration
 - Enables the use of SQL Server 2008 as the underlying database for storing WebSphere Process Server program data (excluding Business Process Choreographer (BPC) Explorer reporting capabilities)
 - Improves the integration of the WebSphere Customization Tool with the WebSphere Process Server for z/OS and WebSphere ESB for z/OS installation experience, enhanced tool support to assist with the DB2

database creation process, and a Common Installer Framework that provides an integrated “look and feel” for all z/OS BPM products

- Complies with the security settings as defined by the Federal Desktop Core Configuration (FDCC) for the U.S. Federal Government

1.8 IBM solution for system events

Next, we discuss the IBM solution for systems events, Tivoli® OMEGAMON® XE for CICS on z/OS.

1.8.1 Tivoli OMEGAMON XE for CICS on z/OS

Although this book is about business application event processing, we mention an available solution for processing system events for CICS.

IBM Tivoli OMEGAMON XE for CICS on z/OS enables the monitoring and management of CICS transactions and resources. It quickly detects and isolates problems when they occur on your complex CICS systems to minimize or eliminate any effect on your customers and your business.

See *IBM Tivoli OMEGAMON XE for CICS Transaction Gateway on z/OS: User's Guide*, SC23-5963.

1.9 Solutions reviewed

With our review of several of the products involved in event processing, we consider the question of which products to use in which situations.

If your business processing runs in CICS, CICS will be the source of your events and forms the subject of this publication. There will be situations in which the actions to take as a result of the events also involve processing in CICS. In other situations, you will want to use other products.

If you want to monitor the processing that happens in CICS, to look at key performance indicators, to provide a dashboard to allow business users to understand the behavior of the business, and to receive alerts, you can use WebSphere Business Monitor.

If you want to derive additional information from combinations of events, potentially including events from other sources in addition to CICS, or to consider events over time, you can use WebSphere Business Events.

You might also want to monitor processing based on certain derived events, in which case, WebSphere Business Events can look for the patterns of interest and can send the resulting events to WebSphere Business Monitor.



CICS event processing

In this chapter, we explain why Customer Information Control System (CICS) event processing (EP) is useful. We discuss the implementation of event processing in CICS, which uses event specifications, capture specifications, and EP adapter information contained in an event binding.

In this chapter, we introduce the Event Binding editor by including snapshots of the CICS catalog sample as we explain the concepts in CICS event processing. To explore the catalog sample event, download the CICS Explorer and use the wizard to generate the sample binding automatically.

In addition, we show how to deploy event bindings to System z file system (zFS) using a CICS bundle.

2.1 Why emit events from CICS applications

Given the massive amount of business processing that occurs in CICS systems across the world (over 30 billion transactions a day), CICS is a significant source of business events.

Emitting events from CICS applications can provide enhanced business flexibility and help you to meet governance and compliance regulations, as described in 1.3, “Why you need events” on page 5.

2.2 How CICS event processing works

Using event specifications defined to CICS, events can be captured from existing business application programs without altering the original code. Figure 2-1 shows an overview of the process.

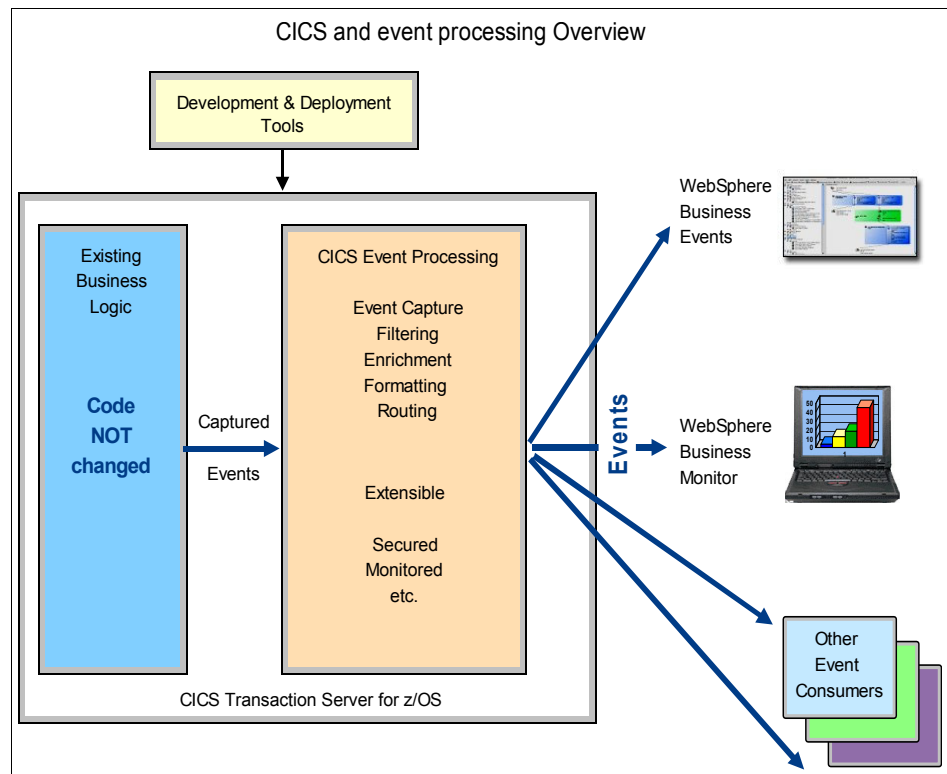


Figure 2-1 CICS and event processing overview

During execution of the business application, many potential capture points occur. These capture points include relevant CICS Application Programming Interface (API) calls and when a program starts. Each time a program executes a capture point, CICS checks each enabled capture specification that matches the capture point.

Each matching capture specification contains optional filters to be compared against the application context, several command options on the API call, and data passed on the API call.

If the filters match, CICS collects the payload information from information sources described in the capture specification, enriches it with context data, then queues the event for dispatch so that the application can continue to execute quickly.

A separate process in CICS routes the event and any payload data through the event processing adapter described in the event binding. Events with a high dispatch priority are routed first. Events marked as transactional are only emitted after the transaction reaches a sync point and commits. The EP adapter then emits the event to a consumer, such as WebSphere Business Events or WebSphere Business Monitor.

2.3 CICS Event Binding editor

Use the CICS Event Binding editor in the CICS Explorer resource perspective to create an event binding that contains one or more event specifications (see 2.4, “Event specification” on page 21).

Deploy a bundle, which contains the event specification to zFS, and install it in CICS (see 2.11, “Bundles” on page 41).

We show panel snapshots from the editor for the catalog sample when discussing the concepts, because the editor can automatically generate the catalog sample for you. You can download the CICS Explorer onto your workstation through the CICS Explorer website:

<http://www-01.ibm.com/software/htp/cics/explorer/>

To generate the catalog sample in the editor, perform the following steps:

1. Switch to Explorer Resource perspective. If you do not see the options for Resource and CICS SM shown, click the plus (+) on the left or use **Window Open Perspective Other** to show it. See Figure 2-2.



Figure 2-2 Explorer Resource perspective

2. Create a CICS bundle project by clicking the create a bundle project icon. See Figure 2-3.

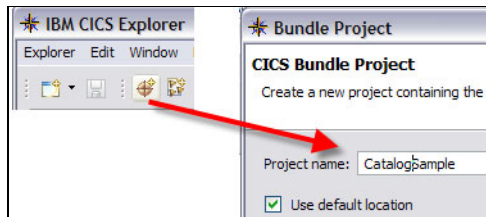


Figure 2-3 Creating a CICS bundle project

3. Generate the catalog sample event binding file. See Figure 2-4.

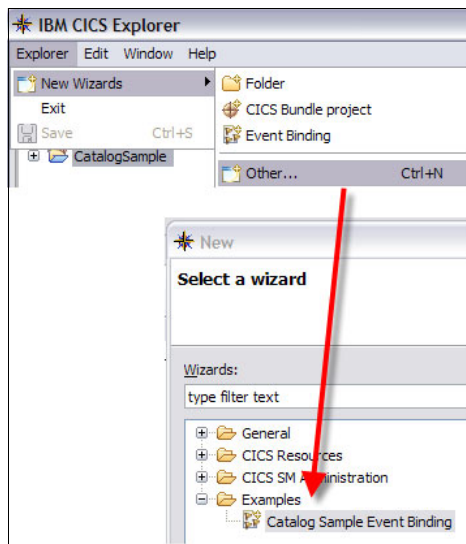


Figure 2-4 Catalog sample binding file

2.4 Event specification

An *event specification* defines a business event to CICS. An event specification can be created using the CICS Event Binding editor by business analysts and developers or by an application analyst in response to a business requirement. An event specification describes an event and its processing in natural language.

See Figure 2-5, which shows the components of an event specification.

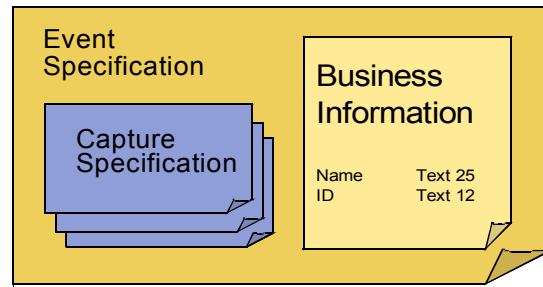


Figure 2-5 Event specification components

An event specification also defines the business information to be contained in the event (if a payload is required). Example 2-1 is an example of an event specification.

Example 2-1 Sample event specification

```
When The stock level is low and there is no re-order in place, capture:  
  Progam name (text)  
  Item Ref (numeric)  
  Item Description (text)  
  In Stock (numeric)  
  On Order(numeric)
```

The event specifies the order in which the payload is produced.

The event description can also indicate the intended use for this event, such as sending the data to the business orders dashboard.

The event specification is associated with one or more capture specifications. See 2.5, “Capture specification” on page 22 for more information about capture specifications.

Note: The event specification represents what the event is. The capture specification represents when and how to capture it.

Figure 2-6 shows the event specification for the catalog sample.

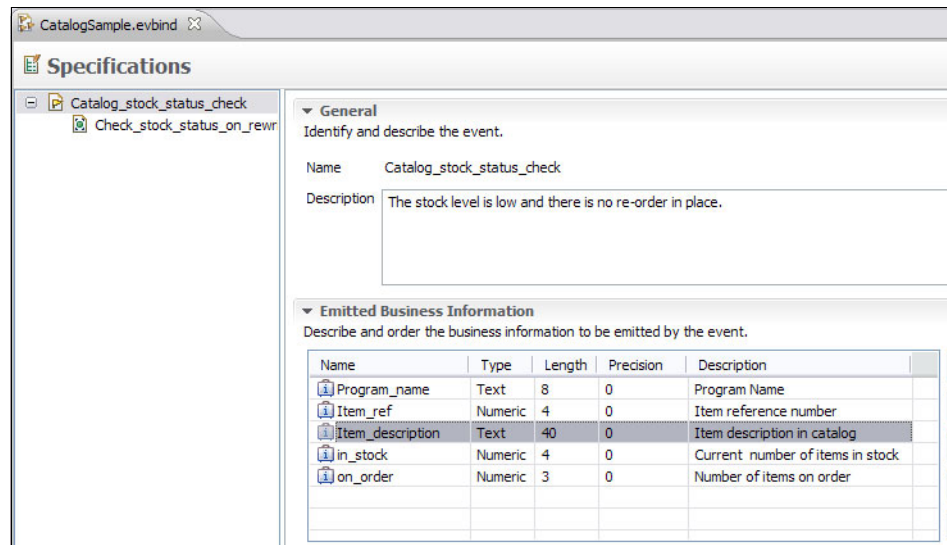


Figure 2-6 Catalog sample event specification

You need at least one capture specification to detail where to capture the event (normally only one capture specification is required). However, there might be more than one place in your application where the event can occur and it is represented by separate capture points. For example, in a stock application, orders can be placed through an online form or through a queue. These orders require separate capture points and the format of the incoming data might differ. In this case, you can define more capture specifications to describe these additional places.

2.5 Capture specification

An application analyst with knowledge of your business applications takes a defined business event and defines one or more capture specifications to satisfy the event (Figure 2-7 on page 23).

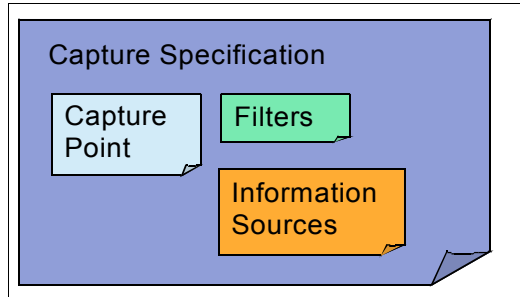


Figure 2-7 Components of a capture specification

A *capture specification* consists of the capture point, such as an EXEC CICS command, relating to the event, and several filter predicates that give more information about the exact location or locations where the event occurs. The location of the event in the application logic depends on how it is specified. If a CICS application contains two instances of the same EXEC CICS API command, and the filter specification does not distinguish between the two commands, an event is emitted when both instances of the command are executed.

The capture specification must contain an information source for each item of business information in the event specification.

You add a capture specification by clicking **Add a Capture Specification** on the window for an event specification or by right-clicking the event specification in the tree at the left and selecting **Add a Capture Specification** on the pop-up menu.

2.5.1 Capture point

The *capture point* is the place in a CICS application where a particular event can be captured (for example, an EXEC CICS READ FILE command or a program starting).

The *program initiation* capture point is the only one that is not an EXEC CICS API call.

Table 2-1 on page 24 lists the EXEC CICS capture points.

Table 2-1 Capture points

CONVERSE	DELETE FILE	DELETEQ TD
DELETEQ TS	INVOKE SERVICE	LINK PROGRAM
PUT CONTAINER	READ	READNEXT
READPREV	READQ TD	READQ TS
RECEIVE	RECEIVE MAP	RETRIEVE
RETURN	REWRITE	SEND
SEND MAP	SEND TEXT	SIGNAL EVENT
START	WEB READ	WEB READNEXT
WRITE FILE	WRITEQ TD	WRITEQ TS
XCTL		

With most capture points, if the filter is TRUE, the event is generated after the capture point has executed. For CONVERSE, INVOKE SERVICE, and LINK PROGRAM capture points, choose whether to generate the event before or after the call. For the program initiation, RETURN, and XCTL capture points, the event is always generated before the call.

Figure 2-8 shows the capture point for the catalog sample.

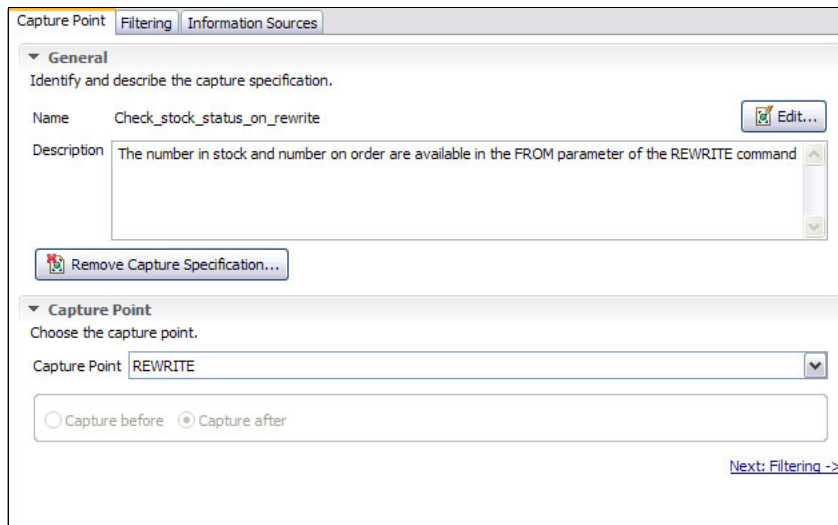


Figure 2-8 Catalog sample capture point

2.5.2 Filter and predicates

The *filter* is a set of predicates connected by AND, which is used to determine whether an event is captured. If all predicates evaluate to TRUE, the event is captured. Predicates that evaluate to FALSE filter out events.

A *predicate* is an expression used as part of a filter, consisting of a data item, an operator, and a value. A predicate is used with data values on the API call or context data to restrict the occasions when an event is emitted to the occurrences of interest.

In the following example of a predicate, Current Program Starts With EXAM, the data item is Current Program, the operator is Starts With, and the value is EXAM. Any program name starting with EXAM will be TRUE, such as in the following example:

```
EXAMPLE          EXAM01
```

Depending on the capture point, you might be able to specify predicates for the application context, options on the API command, and application data.

You use the following predicates to filter the application context:

```
Transaction ID      Current Program      User ID
Response Code       EIBAID              EIBCPOSN
```

You can use predicates to filter by application command options. For example, on a SEND MAP command, use these available command options:

```
MAP*              MAPSET              ALARM
```

At program initiation, use the command options:

```
PROGRAM*          CHANNEL
```

In these examples, MAP and PROGRAM are marked with an asterisk in the editor to indicate that they are *primary predicates*. Specify filter operators and values for primary predicates to maintain CICS performance. Primary predicates are defined for all commands with the following exceptions:

```
CONVERSE, RECEIVE, RETRIEVE, RETURN, SEND TEXT, WEB READ, and WEB
READNEXT
```

Primary predicate: All commands have a primary predicate. For these commands, the primary predicate is not a command option, but it is typically the current program in the application context.

When using capture points for API calls that pass data between the application and CICS, you can specify predicates for the application data. You can import a language structure for the application data to help specify the type of data, the offset into the application data and the length and precision, if applicable. For example, on an XCTL capture point, the application data areas are COMMAREA and CHANNEL.

The command will either use COMMAREA or CHANNEL, so you cannot define application data for both data areas in one capture specification.

Figure 2-9 shows filtering for the catalog sample. The filtering on this panel checks that the transaction equals EGUI, the current program is DFH0XVDS for all user IDs, and the response is 0k.

The screenshot shows the 'Filtering' tab of a configuration window. It is divided into three main sections:

- Application Context:** 'Define predicates to filter events.' It contains four rows:

Context	Operator	Value
Transaction ID	Equals	EGUI
Current Program	Equals	DFH0XVDS
User ID	All	
Response Code	Equals	Ok
- Application Command Options:** 'Define predicates for command options. Predicates marked with * should be specified to maintain CICS performance.' It contains one row:

Name	Operator	Value
FILE*	Equals	EXMPCAT
- Application Data:** 'Define predicates for application data. Whilst defining a predicate you may import a language structure and choose an item from it.' It contains a table:

Source	Container	Offset	Length	Operator	Value
FROM		53	4	Less Than	0024
FROM		57	3	Equals	000

At the bottom, there are navigation links: '<- Back: Capture Point' and 'Next: Information Sources ->'. On the right side of the Application Data table, there are buttons for 'Add...', 'Edit...', and 'Remove...'.

Figure 2-9 Catalog sample filtering

2.5.3 Information sources

If you want to supply a payload for the event, perform the following tasks:

- ▶ Add business information items to the event specification in the order you want them emitted.
- ▶ Define an information source for each item of business information in the capture specifications for the event.

Figure 2-10 shows information sources for the catalog sample.

Name	Type	Fo...	Source	Contal...	Offset	Capture Length	Capture Type
Program...	Text	8	PROGRAM				
Item_ref	Numeric	4	FROM		0	4	Character
Item_des...	Text	40	FROM		4	40	Character
in_stock	Numeric	4	FROM		43	4	Character
on_order	Numeric	3	FROM		57	3	Character

Figure 2-10 Catalog sample information sources

You add an information source by choosing from the following options:

- ▶ Application context
- ▶ Application command options
- ▶ Application data

Application context

Choose from USERID, PROGRAM, or TRANSID. CICS knows the format and length of these choices.

Application command options

These application command options are the values of options passed on the API command. For example, at the INVOKE SERVICE capture point, use these choices for the application data areas:

SERVICE	OPERATION	URI
CHANNEL	URIMAP	

CICS knows the format and length of these choices. So, if you chose SERVICE for an application data area, CICS supplies a 32-character value, which is the length of the SERVICE parameter on the INVOKE SERVICE call.

Application data

For capture points that pass data between the application and CICS, you can specify application data as the information source. For example, at the program initiation capture point, the choices for the application data areas are COMMAREA and CHANNEL. In this case, CICS does not know the format and length of the data area so you must supply it.

For example, if you choose CHANNEL, you must supply the name of the container holding the data, the type of data, its offset into the container, and the length of the data. If the data type is Packed Decimal or Zoned Decimal, you are given the option to supply the precision. If the data type is Character, you can choose an alternate code page from the default code page IBM037. You can import a language structure for the application data to help specify this information.

Figure 2-11 shows the information source editor.

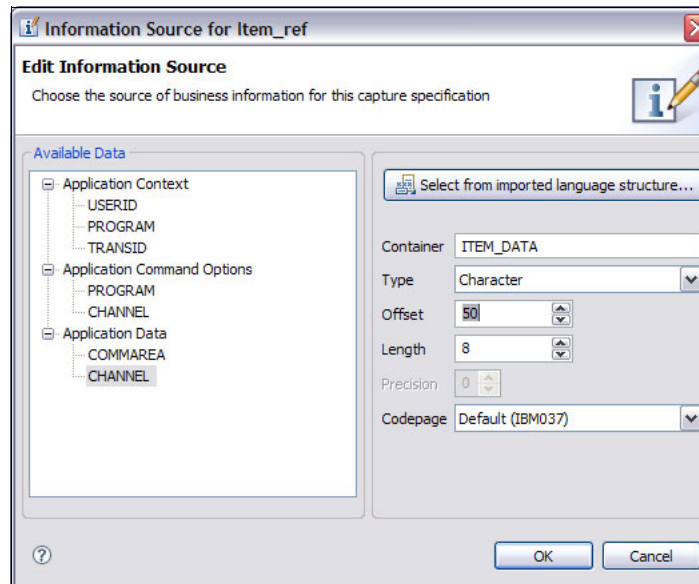


Figure 2-11 Editing an information source

2.6 Event binding

An *event binding* is an XML definition that defines one or more business events to CICS. An event binding consists of event specifications, capture specifications, and EP adapter and dispatcher information.

The event binding is the unit for installing, enabling, and disabling CICS events. The event binding groups together sets of events that are to be handled using the same EP adapter, configuration, and dispatching policy. Event bindings are deployed to CICS in a bundle that can contain other resources. All of the resources in a bundle can be enabled and disabled together.

You create an event binding in a CICS bundle project using the CICS Event Binding Editor. Figure 2-12 shows an event binding containing three event specifications.

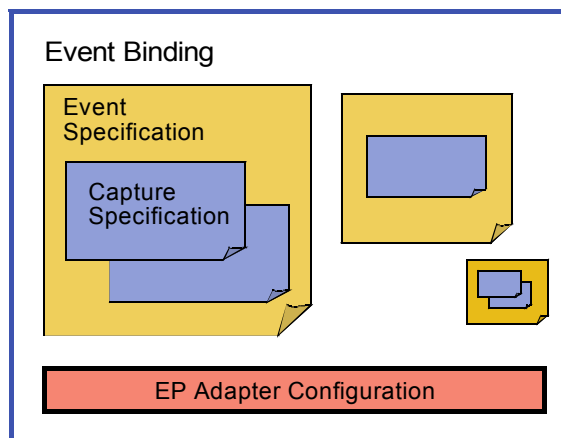


Figure 2-12 Event binding containing three event specifications

The event binding contains the following information:

- ▶ Description
- ▶ User tag

Tip: You can use the CICS Systems Management (SM) perspective, which is a part of CICS Explorer, to display the user tag for an event binding installed in CICS.

Figure 2-13 shows the event binding details for the catalog sample. The EP adapter information is shown on the Adapter tab.

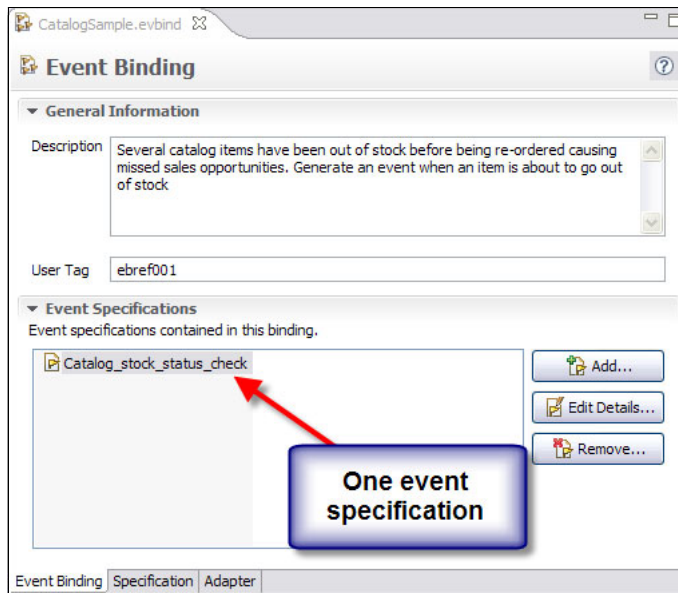


Figure 2-13 Catalog sample event binding

2.7 Non-invasive events or SIGNAL EVENT

CICS EP support allows existing business applications to be instrumented to emit events without altering the existing application code. We call this function *non-invasive event processing*.

If you want to identify explicitly a place in an application program where one or more events can be emitted, add an EXEC CICS SIGNAL EVENT call to your program. This call is invasively adding the opportunity for an event to be captured. The SIGNAL EVENT call is also useful, because you might need to collect the data available for the event payload from diverse sources. You can collect the information in containers on the FROMCHANNEL or pass a single data area using FROM.

The SIGNAL EVENT identifies a place in an application program where one or more events can be emitted. Events are emitted when the following conditions are satisfied:

- ▶ Event processing is active.
- ▶ There is at least one matching capture specification enabled. A capture specification matches if it has a capture point of SIGNAL EVENT, and all its predicates evaluate to TRUE.

SIGNAL EVENT has a primary predicate of EVENT and allows other predicates on the FROM data area or the FROMCHANNEL and its containers. You can also define predicates on the context data, which you can do for all capture points. The data in any CICS event emitted as a result of SIGNAL EVENT is defined in the business event that contains the matching capture specification.

SIGNAL EVENT works in the same way as any other capture point in CICS, except for the following differences:

- ▶ You can explicitly choose where in the application logic the capture point occurs.

The capture points that we provide might not meet your requirements.

- ▶ You provide the name for this capture point:
 - This name might be unique, or you can provide a common name, such as the application area.
 - Capture points can filter on this name with Equals or Begins With, for example.
- ▶ You can assemble the data that you want to make available from diverse sources.

A capture specification for this signal event can then select information sources from this data to build the event payload.

SIGNAL EVENT is also useful for application vendors to provide prepared capture points for their customers. Again, a range of useful information can be provided for event payloads.

2.7.1 Automatic capture specification for SIGNAL EVENT

If you create a SIGNAL EVENT using FROMCHANNEL and write the data into containers, you can use the editor to create a capture specification for the event automatically.

You add an automatic capture specification using the Event Binding editor by performing the following steps:

1. Add a new event specification with a name that matches the EVENT predicate.
2. Add the business data that you want to capture with the names of the containers that you are passing on the FROMCHANNEL.
3. Click **Add**, which is an automatic capture specification.

The editor creates a capture specification for a SIGNAL EVENT capture point with the following filter:

```
EVENT Equals <name of event specification>
```

It also adds information sources for each business data item:

- ▶ The information is in a container with the name of the business data passed on the FROMCHANNEL.
- ▶ The data is at offset 0 in the container.
- ▶ The length is the same as the length of the business data.

Figure 2-14 shows an event specification to be used with SIGNAL event.

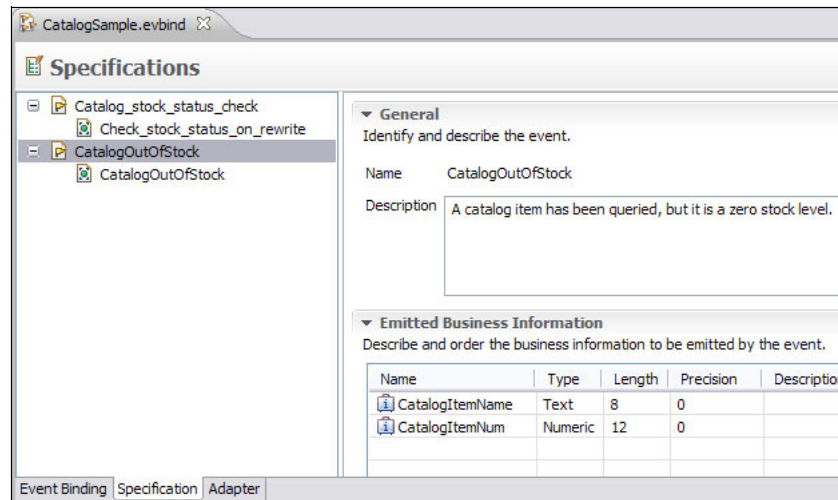


Figure 2-14 Event specification for a SIGNAL EVENT

Figure 2-15 on page 33 shows an automatically generated capture specification. Instead of defining the event specification to yield the most appropriate automatic capture specification, you can also use this option to generate a template, which can then be tailored as needed.

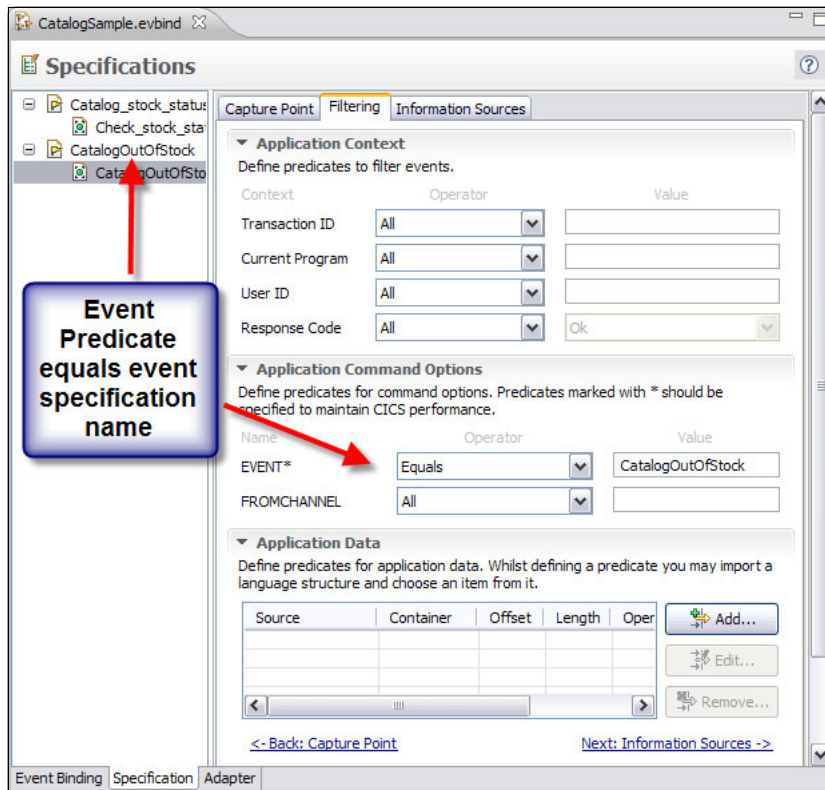


Figure 2-15 Automatically generated capture specification

Figure 2-16 shows the automatically generated information sources in the capture specification.

Name	Type	F...	Source	Container	Offset	Capture Length	Capture Type
Program...	Text	8	FROMCHANNEL	Program_name	0	8	Character
Item_ref	Numeric	4	FROMCHANNEL	Item_ref	0	4	Character
Item_des...	Text	40	FROMCHANNEL	Item_description	0	40	Character
in_stock	Numeric	4	FROMCHANNEL	in_stock	0	4	Character
on_order	Numeric	3	FROMCHANNEL	on_order	0	3	Character

Figure 2-16 Automatically generated information sources

2.8 Event processing adapters

Specify information in your event binding to control how CICS emits events produced by the event binding. Use the Adapter panel to define what will happen to events created by this binding. Select the EP adapter to emit events, then select options relevant to the EP adapter. Figure 2-17 on page 35 shows the adapter configuration for the catalog sample.

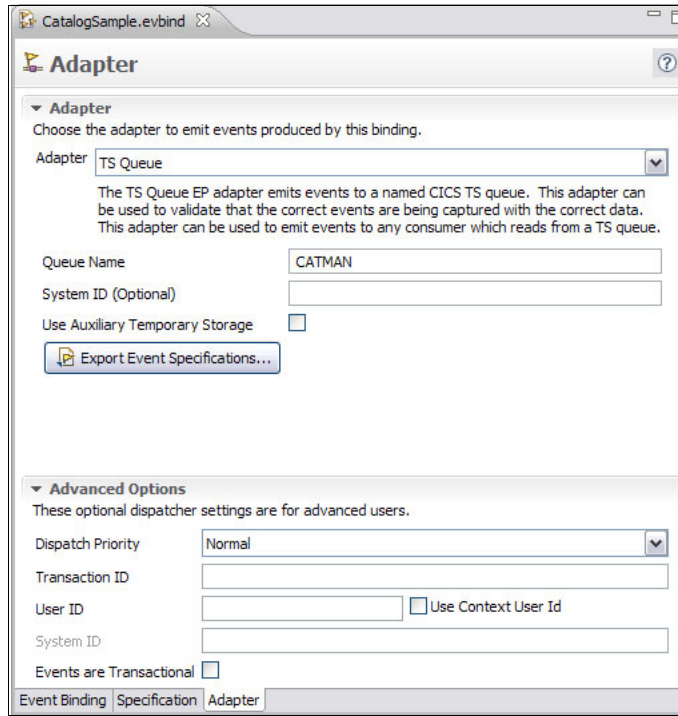


Figure 2-17 Adapter information for the catalog sample

Click the **Adapter** tab in the CICS Event Binding Editor, as shown. The Adapter pane is where you specify the type of EP adapter to use for this event binding, the parameters for the EP adapter, and any advanced information.

Choose the EP adapter type from the Adapter list. You can specify four EP adapter types, which are discussed in 2.8.1, “Temporary storage queue EP adapter” on page 36 through 2.8.3, “WebSphere MQ (WMQ)” on page 37.

You can export a description of your event as an XML schema or COBOL copybook. See 2.9, “Exporting event schema or copybook” on page 38.

You can optionally set advanced dispatcher settings. See 2.10, “EP adapter advanced options” on page 39.

2.8.1 Temporary storage queue EP adapter

This EP adapter emits events in the CICS flattened event (CFE) format to a named CICS temporary storage (TS) queue and can be used to perform the following tasks:

- ▶ Validate that the correct events are being captured with the correct data
- ▶ Emit events to any consumer that reads from a TS queue

This EP adapter is a good choice when initially implementing new events. You can use it as a temporary adapter, because you can browse TS queues with the CICS CEBR transaction easily to check the payload. You can refresh the view to check for new events in the queue.

However, TS queues are limited to a maximum of 32,767 entries. You can only delete the entire queue rather than deleting individual entries within the queue, so you will need a strategy for clearing out events that you have processed.

Specify the following options for the TS queue EP adapter:

- ▶ Specify the CICS queue name. You must specify a queue name.
- ▶ Specify the System ID, if your target queue is remote.
- ▶ Select **Use Auxiliary Temporary Storage**, if required.

2.8.2 Transaction start EP adapter

The transaction start EP adapter emits events in the CICS channel-based event (CCE) format to a named CICS transaction that consumes the event. You can specify the CICS system that will run the transaction. You can use an existing transaction, if the event data is not required.

Specify the following options for the transaction start EP adapter:

- ▶ Specify the transaction ID of the CICS application that runs as a result of the events. You must specify a transaction ID.
- ▶ Optionally, specify a transaction user ID.

If unspecified, the started transaction will run under the CICS region user ID.

The transaction that is started by the transaction start EP adapter will run under this user ID. The CICS region user ID needs to be defined as a surrogate of any user ID specified here.

2.8.3 WebSphere MQ (WMQ)

This EP adapter emits events to a WebSphere MQ queue either in the common base event (CBE) XML format for consumption by products that use the common event infrastructure, such as WebSphere Business Monitor, in the WebSphere business event (WBE) format for WebSphere Business Events, or in a non-XML character format.

Specify the following options for the WebSphere MQ (WMQ) EP adapter:

- ▶ Specify the queue name of the WebSphere MQ queue on which events emitted by this event binding are placed. You must specify a queue name.
Specify whether messages are persistent. Select one of the following values from the Persistent list:
 - No
Messages put on the queue by the WebSphere MQ (WMQ) EP adapter are non-persistent.
 - Yes
Messages put on the queue by the WebSphere MQ (WMQ) EP adapter are persistent.
 - Queue default
Messages put on the queue inherit the default persistence of the named queue.
- ▶ Specify the message priority. You can either select the queue default, or type a value in the Priority field, for the WebSphere MQ message priority, from 0–9.
- ▶ Specify the expiry time. You can either select Never Expire, or type a value for the WebSphere MQ message expiry in the Expiry Time field. This period of time is expressed in tenths of a second. A message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.
- ▶ Specify a data format for the event. Select one of the following values from the Data Format list:
 - CICS Flattened Event (CFE) format
Event data is in a non-XML character format.
 - WebSphere Business Events XML format
Messages are put on the queue in the XML format required by WebSphere Business Events.
 - CBE format for WebSphere Business Monitor

Messages are put on the queue in the CBE event format required by WebSphere Business Monitor.

2.8.4 Custom (user-written) EP adapter

This adapter emits events in any format that you require. A custom EP adapter is a CICS program that you write to provide a combination of formatting and routing of an event that is not supported by the CICS-provided EP adapters. It must not perform out any other processing, such as consumption of the event.

Specify the transaction ID for your user-written CICS application that formats, routes, and emits the event. You must specify a transaction ID.

Write the data to be passed to the custom EP adapter. Your custom EP adapter will receive this data, which can be used to configure the custom EP adapter.

HTTP EP adapter: An HTTP EP adapter is now available through an authorized program analysis report (APAR). New Function (NF) APAR PK94205 adds an event processing (EP) adapter that uses the HTTP transport mechanism to CICS Transaction Server for z/OS V4.1. This new EP adapter supports event formats of Common Base Event REST (CBER), WebSphere Business Event, and Common Base Event (CBE). APAR PK94205 closed on 29 April 2010.

2.9 Exporting event schema or copybook

You can export descriptions for one or more event specifications in this event binding as a schema or copybook for use elsewhere.

If you are using the WebSphere MQ (WMQ) EP adapter and the CBE format or the WebSphere Business Events (XML) format, the exported file will be an XML schema definition (.xsd) file.

If your chosen EP adapter and data format emit events to a system in a non-XML character format, the exported file will be a COBOL copybook (.cpy) file. The TS Queue adapter and the WebSphere MQ (WMQ) EP adapter using the CICS Flattened Event format emit events in a non-XML character format.

To export event schema or copybook, perform the following steps:

1. Click **Export Event Specifications**. The Export Event Specifications window opens.

2. Select the event specifications that you want to export.
3. Specify a directory to which to export the event specifications in the “To” directory field.
4. Click **Export**.

A file is created in the specified directory for each event specification that you selected.

For example, if you specify the WebSphere MQ adapter and the CBE format and select two event specifications called `example1` and `example2`, two XML schema files are created, which are called `example1.xsd` and `example2.xsd`. You can import these schema files to WebSphere Business Monitor to help define an inbound event.

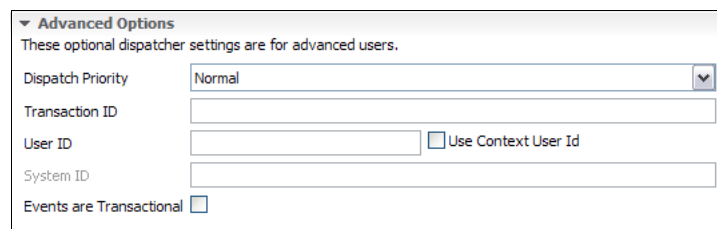
If you specify the WebSphere MQ adapter and the WebSphere Business Events (XML) format and select an event specification called `example1`, an XML schema file is created, which is called `example1.xsd`. You can use this schema file in the WebSphere Business Events Design Data tool to help define an event.

If you specify the TS Queue adapter and select two event specifications called `example1` and `example2`, two COBOL copybooks are created, which are called `example1.cpy` and `example2.cpy`. You can use these copybooks to process data in your own event consumer programs.

If you export an event specification and then export the same event specification again to the same directory, the CICS Event Binding editor prompts you to either overwrite the existing file or cancel the export operation.

2.10 EP adapter advanced options

Specify any required advanced dispatcher options, as shown in Figure 2-18. These options are for advanced users and control the way that the EP adapter is run in a CICS system. These options are not mandatory; default values will be assigned if you do not supply any values.



Advanced Options
These optional dispatcher settings are for advanced users.

Dispatch Priority: Normal

Transaction ID: [Empty field]

User ID: [Empty field] Use Context User Id

System ID: [Empty field]

Events are Transactional

Figure 2-18 Advanced options for dispatch

2.10.1 Dispatch priority

You can specify Normal or High priority to control the dispatching of events from which the event was captured. High priority events are emitted as soon as they are available based on the “Events are Transactional” setting. Normal priority events are emitted as soon as they are available, unless the “Events are Transactional” option is selected, but they are emitted after any outstanding high priority events.

2.10.2 Transaction ID

Specify the Transaction ID, which is not available for the CICS Transaction EP adapter or a custom EP adapter. The EP adapter program will run under this transaction ID. If you do not specify a transaction ID and you do not specify a user ID, the EP adapter is linked to under the dispatcher transaction.

2.10.3 User ID

You specify a user ID so that the EP adapter transaction runs with this user ID; otherwise, it runs under the CICS region user ID. If you select “Use context User ID”, the EP adapter runs with the user ID under which the event was captured.

If you specify a user ID, but you do not specify a transaction ID, the EP adapter will run under the default transaction for the EP adapter type:

- ▶ The WebSphere MQ (WMQ) EP adapter runs under the CEPQ transaction.
- ▶ The TS Queue EP adapter runs under the CEPT transaction.

When you install a BUNDLE resource that includes an event binding for which you specified a user ID in the Adapter tab of the CICS Event Binding Editor, CICS checks that the User ID performing the installation operation is authorized as a surrogate user of the user ID specified in the CICS Event Binding Editor. This check also applies to the CICS region user ID during the group list installation on a CICS “cold” or initial start.

2.10.4 System ID

Specify the system ID (only available for the CICS Transaction EP adapter). The transaction started by the EP adapter will run on the CICS system with this system ID.

2.10.5 Transactional events

Specify whether events are transactional:

- ▶ Select the check box if you want CICS to emit captured events only if the unit of work (UOW) associated with the event completes successfully. The events are either emitted when there is an internal sync point within the program or when the program ends and reaches an implicit sync point.

If the transaction fails and backs out, the events will be discarded.

- ▶ Clear the check box if you want CICS to process events associated with this event binding in near-real time regardless of whether the UOW commits.

2.11 Bundles

You can now deploy applications into CICS using bundles. A *bundle* is a collection of CICS resources, artifacts, references, and a manifest that represents an application. Use bundles to more easily manage the availability of an application and the life cycle of its resources.

For an example of a bundle containing three event bindings, see Figure 2-19.

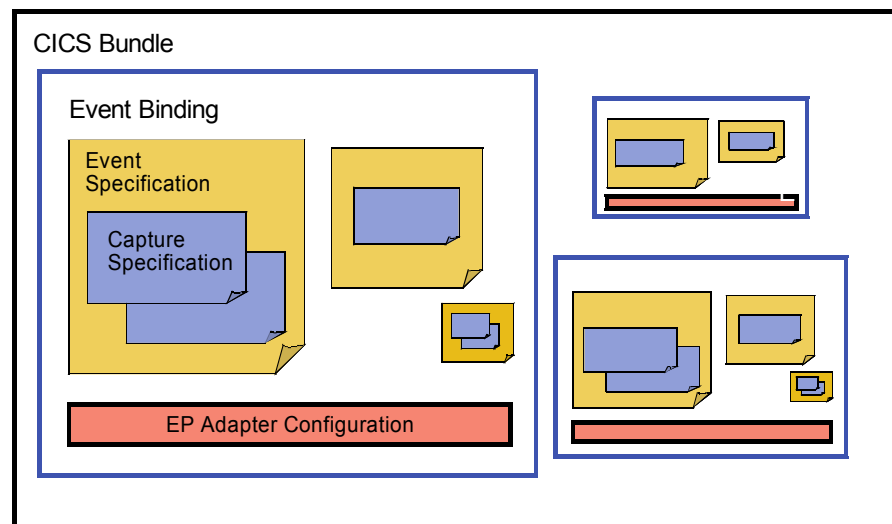


Figure 2-19 A CICS bundle containing three event bindings

Bundles are created by an application developer using a tool, such as Rational Developer for System z, the CICS XML assistant, or the IBM CICS Explorer. A bundle contains only the resources that are required by the application. The

system resources that the application requires might be defined as prerequisites, but they are not included in the bundle. This separation means that you can install the same application into multiple CICS regions without having to repackage or redeploy the bundle.

A bundle is defined in CICS using a BUNDLE resource. For information about how to define this resource and for more information about the format of its contents, see the following web page:

http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/topic/com.ibm.cics.ts.resourcedefinition.doc/bundles/defining_app_resources.html

The BUNDLE resource differs from a Resource Definition Online (RDO) group, because it maintains a relationship with all the resources after they are installed, so that you can manage all the related resources as a single entity. For example, if you disable a BUNDLE resource because you want to stop an application from running, CICS disables all of the related application resources for you. To view the contents of a bundle and the state of its resources, use the IBM CICS Explorer.

The types of applications that you can deploy as bundles include event processing, channel-based services, and XML-based services. Each of these application types is represented by one or more CICS resources. These resources are dynamically created as part of the bundle deployment.

If you deploy an application that uses event bindings from the CICS Event Binding Editor, installing the BUNDLE resource generates one or more EVENTBINDING and CAPTURESPEC resources. The resource signatures for each resource indicate that they were created during a bundle deployment and contain the name of the BUNDLE resource.

2.12 Deploy a bundle to zFS

Important: This option is shown as Export to System z HFS, as shown in Figure 2-20 on page 43.

You export the bundle to z/OS UNIX® (zFS) storage from which it can be installed into CICS by referencing the bundle directory in a CICS bundle resource definition. To deploy a bundle to zFS, right-click the bundle and select the **deploy** option, as shown in Figure 2-20 on page 43 (where the option is called Export to System z HFS).

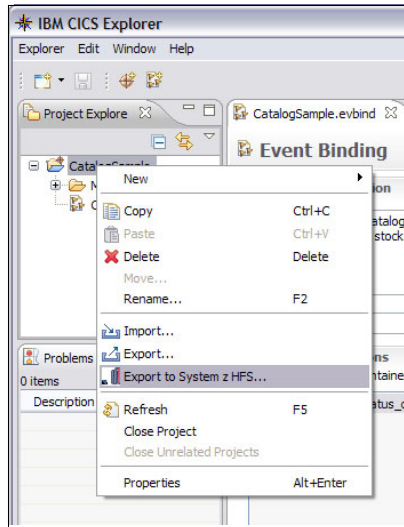


Figure 2-20 Selecting the export panel

Complete the information in the Export panel to provide the host, FTP port, user name, and password for the z/OS system. Enter the hierarchical file system (HFS) directory or browse for the desired directory. See Figure 2-21 on page 44.

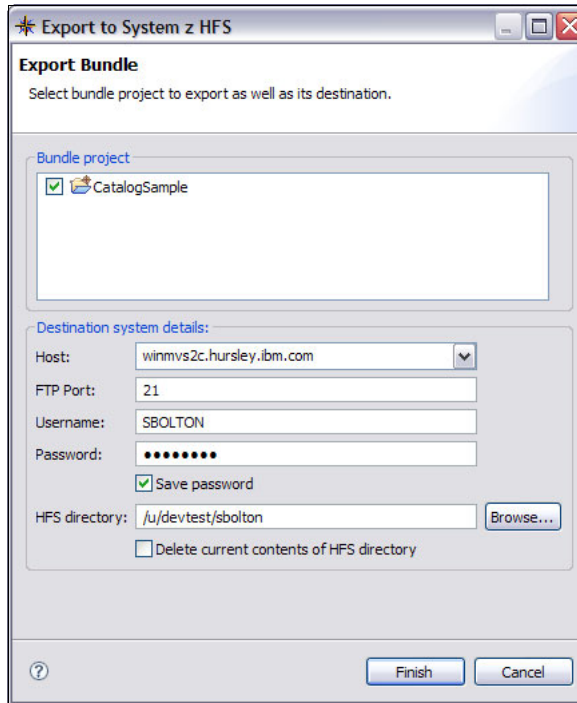


Figure 2-21 Export bundle panel



Integrating event processing and enterprise service bus

In this chapter, we describe the conceptual architecture for event processing systems, in addition to the three typical layers of an event-driven system:

- ▶ Event producer and associated components
- ▶ Event consumer and associated components
- ▶ Intermediary Event Bus layer

3.1 Conceptual architecture

The conceptual architecture builds on the concepts of the event processing network (EPN) by defining the components that can be involved in an event processing solution. Several of these components are equivalent to an event processing agent (EPA), or a set of connected EPAs, at the EPN level. Other components are more involved in the flow of events and are equivalent to channels in the EPN. Later in this section, we show how the conceptual architecture maps to the EPN (see Figure 3-3 on page 53).

Any system architecture that supports business event processing must enable the flexible definition of the event-processing logic: detection of event patterns, derivation of new events, transformation, and routing from producers to consumers based on the business logic required. Thus, businesses can react to changes, execute the relevant processes, and influence ongoing processes based on the changes. Furthermore, such a definition of event processing must be easy to modify and quick to deploy in accordance with business needs, such as changes in business processes and policies.

To understand how this business value can be derived from event processing systems, it is important to consider a further layer of granularity than the event processing network, to consider components that can be used to construct an event processing system, and their interactions. The outcome of this process is what we term a *conceptual architecture for event processing systems*. In addition to the three typical layers of an event-driven system (the event producer and associated components, event consumer and associated components, and an intermediary Event Bus layer), the conceptual architecture needs to include components for security, monitoring, analytics, and management of events and event flows.

At the simplest level, a minimal set of conceptual components required for an event processing system consists of an Event Emitter layer to emit events from event producers, an Event Bus, and an Event Handler layer to handle events for consumption by event consumers. Figure 3-1 on page 47 illustrates this architecture and also includes examples of event producers and event consumers.

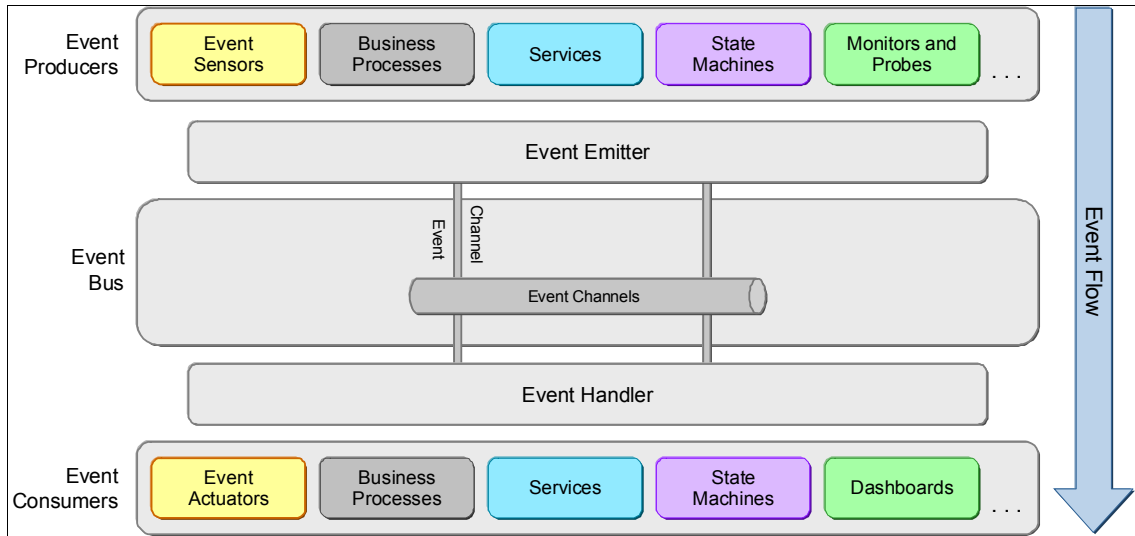


Figure 3-1 Minimal event processing conceptual architecture

There might be a need for additional capabilities within the Event Emitter layer, the Event Bus, and the Event Handler (or receiver) layer. In practice, generated events from an event producer cannot always be immediately shared with event consumers. Because the event producer does not require awareness of the event consumers in an event processing system, there is typically a need for a middleware layer between the producer and consumer. This middleware layer performs additional event-related tasks, as well as enabling the consumer to receive those events, or derivatives of them, that are of interest. Not all events are generated by the producer in the required format, and, in such cases, the events need to be transformed to the required (enterprise standard) format prior to being published to the intermediate layer. In certain cases, an ordinary event can be evaluated for notability by an event preprocessor (router or filter), resulting in the generation of a new notable event. Also, an event producer can choose not to emit all of the events. Event processing agents can filter and mediate raw events within the domain of the producer.

Similarly, all of the events received at the consumer side might not be in a ready-to-use form, so there might be processing and mediation required at the consumer end. An ordinary event might be evaluated for notability by an event preprocessor (filter) before detailed handling at the consumer end. The consumer might choose to ignore part of the events it receives. Event processing services fulfill these pre-publish and pre-receive event processing requirements at the event handler layer.

Figure 3-2 shows all the components of the event processing conceptual architecture. Any event processing implementation must be achievable with this architecture as the base set of components at the conceptual level, but not every component will be required in every implementation. Similarly, not all of the components will be required for any given scenario. When we look at the scenarios and see how they map to the conceptual architecture, we see that every component is not involved in every scenario.

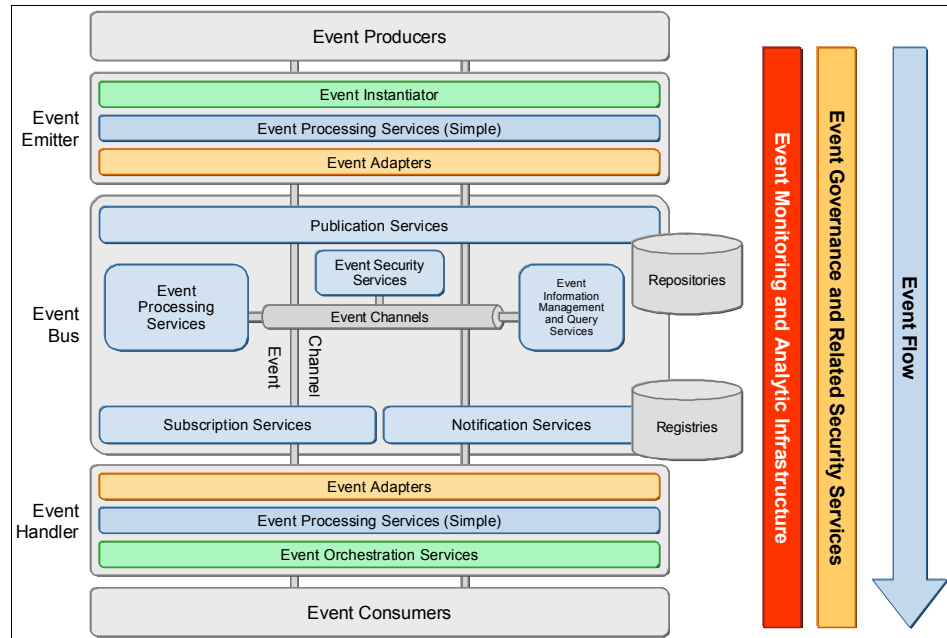


Figure 3-2 Event processing conceptual architecture: Components that can be involved in an event processing system

Architectural components

The event flow in the conceptual architecture is from producer to consumer, and the components shown in the conceptual architecture diagram are summarized in that order. At the implementation level, event consumers will often also be producers of events, but at a conceptual level, the roles of consumer and producer are separate.

► Event producer

The *event producer* emits an event when something of interest happens (or does not happen). The event producer does not include logic for manipulating events, nor any decision logic on what to emit when, and the events that are generated can be redundant or irrelevant.

Typical examples of event producers include:

- *Event sensors*, which detect situations (things that happen) and generate raw events, or originate events from data streams or business flows, for example, the transmission of temperature
- *Monitors and probes*, which produce events about availability and problems in systems, such as faults in an IT network
- *Business processes*, which produce events at significant points in the processing (for example, at milestones or checkpoints), or when a specific process task is reached or started
- *Services and applications*, which produce events at key points in the processing, such as when the service is invoked and completes, or when it fails
- *State machines*, which generate events when changing state

► Event emitter

The *event emitter* is logically (although not necessarily physically) associated with the event producer and is responsible for converting and packaging raw events from producers, for delivery to the Event Bus. The event emitter can include an event instantiator, which creates the instances of events; simple event processing services, such as filtering and mediation of events emitted by a single producer, enriching the event with information available at the time the event occurs; and event adapters. The event instantiator takes events from the producer and does whatever (if anything) is needed to make it available for further event processing or delivery, which can include aggregation, caching, and serialization of events. The event instantiator might be required to manipulate the event header to embed *semantic metadata* into the event message itself, and to make it self-describing (with information such as the time, date, instrument type, process ID, and so forth). The event emitter can provide optimization by carrying out simple event processing at this stage rather than after events reach the Event Bus. Event adapters can provide formatting and protocol conversion of the event into a form to be received by the event processing network, such as wrapping event records as event messages and sending the event messages to the Event Bus.

► Event Bus

The *Event Bus* receives events from event emitters, potentially at an extremely high volume of events, and invokes consumers through event handlers as a result of events. Among the capabilities of an Event Bus can be processing to derive a lower volume of more informative events from the incoming events. The components of the Event Bus do not have to be co-located. The section “Event Bus components” on page 51 gives further detail about the Event Bus.

► Event handler

This component prepares the events from the Event Bus for consumption by the event consumer, receiving events and deciding how to react. The event handler has event adapters to receive event messages from the Event Bus and unwrap them to get event records. The event handler can also provide simple event processing services, which carry out consumer-side processing to filter and mediate events received from the Event Bus. Event handlers can also determine the appropriate consumers to react to an event, and invoke the consumers with context derived from the event. Finally, an event handler can provide event orchestration services to manage the distribution of events to consumers.

► Event consumer

The *event consumer* performs tasks in reaction to an event. The event consumer has limited concern about the origins of the event and is merely aware that it is being invoked as a result of the event along with context about the event. The following examples are typical event consumers:

- *Event actuators*, which are invoked to perform physical tasks, such as operating valves, switches, or alarms
- *Operator dashboards*, which display information about the behavior of IT systems and affected services
- *Business dashboards*, which display information about the behavior of business processes
- *Business processes*, which can be initiated or resumed in response to an event
- *Services and applications*, which can be invoked in reaction to an event and can include external content management systems or event repositories
- *State machines*, whose state can be changed in reaction to an event

This view of the conceptual architecture is based on the roles of each component, but that is not to say that a particular participant in the architecture cannot perform more than one role: an event producer can also carry out event processing and can act as an event consumer. In particular, the publication and subscription services are only required where a Publish/Subscribe-style model is used.

The conceptual architecture can be regarded as “nested,” in that any participant can contain within it a network of further components. For example, an event producer might emit an event to the main Event Bus. In the process of producing that event, you can envision a *mini* version of the overall model, in which a

producer emits an initial simple event for pattern matching with other events in a *mini* Event Bus, residing logically within the overall event producer.

Event Bus components

The *Event Bus* transmits events from producers to consumers and can provide additional services for processing and routing events. The Event Bus can have an associated event registry and can have the capability to perform transactional storage of in-flight events (transient or persistent) by using an event repository.

The Event Bus can be local or implemented at an enterprise level, and the events received need to be processed based on the business requirement. This solution is achieved using simple and complex event processing services. These services are provided by event processing agents, which are wired through event channels.

The Event Bus can provide the following services or building blocks:

- ▶ Event channels, which transmit events from event emitters to the Event Bus, between components of the Event Bus, and onto event handlers.
- ▶ Publication services, to enable producers to send events to the appropriate channels.
- ▶ Subscription services, to enable the dynamic registration of producers and consumers, such as allowing event handlers to find the appropriate channels and subscribe to receive events from those channels.
- ▶ Notification services to notify subscribed event handlers when events are available, supporting both the push and pull of events.
- ▶ Query services to allow a repository to be queried for events (and metadata).
- ▶ Event security services, to control access and authority for events; for example, to control authorization for adding and removing events to and from the Event Bus, as well as privacy and non-repudiation of event contents.
- ▶ Event processing services, which provide filtering, transformation, and enrichment of events, and can also provide pattern matching and event derivation. These services can include complex event processing, which processes events from multiple sources, and perform long-running pattern matching among events.
- ▶ Event information services, which enable administrators to add, remove, and organize channels, to organize event type metadata (syntax and semantics) and to alternatively store event data in a relational format rather than using persistence based on an event message, that is, atomic persistence.
- ▶ An event registry, to provide a taxonomy of event types and an ontology of event relationships.

- ▶ An event repository, to store events for medium-term to long-term event persistence.

The following function types are the most significant function types that must be provided by processing within the Event Bus:

- ▶ Transformation: Function that transforms the incoming event by translating or splitting it
- ▶ Enrichment: Function that enriches the content of events with reference data from multiple possible sources
- ▶ Validation: Function to provide validation against required criteria
- ▶ Pattern detection: Function that recognizes actual and retrospective patterns; a combination from possibly multiple events, characterizing a significant business situation
- ▶ Filtering: Stateless function that filters events based on their content; that is, the information that is carried by the message generated when the event happened
- ▶ Aggregation: Function that can group events as necessary
- ▶ Routing: Function that routes events to the destination based on various possible routing patterns, such as pre-established itinerary, calendar-based, subscription, or “intelligent” routing decisions

The conceptual architecture also includes event governance and security services, to manage and control the life cycle of events and event metadata. Event monitoring and analytic infrastructure are needed for mainly administrative purposes, to notify users of failures in the event infrastructure, and to gather and display statistics about the event flow. These capabilities must cover the full event flow and are, therefore, shown on the right side of the conceptual model diagram.

Thus, the conceptual architecture represents event producers emitting events to the Event Bus, where they might be processed, and where they are finally consumed by event consumers. A consumer can, as a result of an event, produce another event, or react in other ways with another component, which itself produces an event as a result.

Figure 3-3 on page 53 shows an example of how the conceptual architecture builds on the concept of an EPN, by illustrating components that are equivalents of event processing agents (EPAs), or sets of connected EPAs, and other components that provide event channel services.

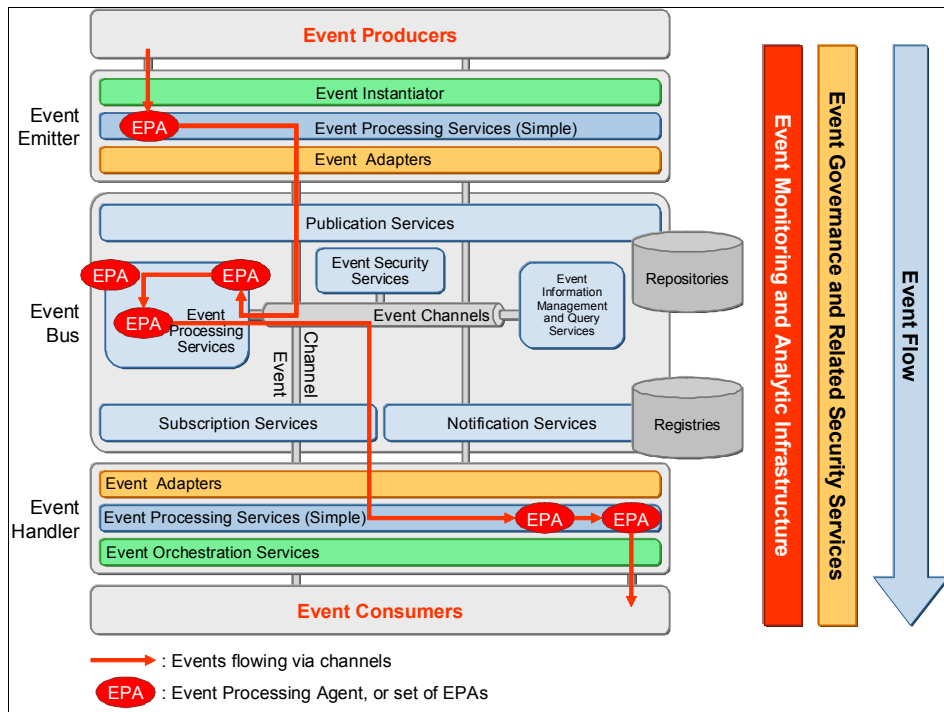


Figure 3-3 An example of EPN used by the event processing conceptual architecture

3.2 Why use an enterprise service bus as the Event Bus

The following function types are the most significant function types that must be provided by processing within the Event Bus:

- ▶ Transformation: Function that transforms the incoming event by translating or splitting it
- ▶ Enrichment: Function that enriches the content of events with reference data from multiple possible sources
- ▶ Validation: Function to provide validation against required criteria
- ▶ Pattern detection: Function that recognizes actual and retrospective patterns; a combination from possibly multiple events, characterizing a significant business situation
- ▶ Filtering: Stateless function that filters events based on their content; that is, the information that is carried by the message generated when the event happened

- ▶ Aggregation: Function that can group events as necessary
- ▶ Routing: Function that routes events to the destination based on various possible routing patterns, such as pre-established itinerary, calendar-based, subscription or “intelligent” routing decisions

In this book, we show examples of using an enterprise service bus (ESB). See Chapter 5, “WebSphere Enterprise Service Bus business scenario” on page 91, Chapter 7, “DataPower business scenario” on page 185, and Chapter 6, “WebSphere Message Broker business scenario” on page 131 to transform and enrich an event received from CICS Transaction Server.

The ESB receives the common base event (CBE) and then fires off a Web Service call, `inquireSingle`, which is located in box `WS2`, to enrich the payload with the product item price. The price is in the VSAM inventory file available to CICS. See Chapter 4, “Overview of the application and business scenarios” on page 57 for more details.

The ESB transforms the input event changing the format from a CBE format to a WebSphere business event (WBE) format and sends the enriched event to WebSphere Business Events. See Chapter 4, “Overview of the application and business scenarios” on page 57 for more details.



Part 2

Environment

In this part of the book, we describe the application that we used, and we review our business scenario.



Overview of the application and business scenarios

In this chapter, we describe our environment and the application we chose to use to build our scenarios.

4.1 Objectives

For this book, we wanted to build a test application with several event-based scenarios to demonstrate ideas and techniques about integrating an enterprise service bus (ESB) into your WebSphere Business Events solutions.

We wanted a test environment with which CICS clients were familiar and that they might be able to utilize for demonstration purposes; therefore, we chose to use the CICS Transaction Server-supplied *catalog manager example application*.

4.2 Overall architecture

IBM provides three ESB solutions: WebSphere Enterprise Service Bus, WebSphere Message Broker, and DataPower. We incorporated all three solutions into our testing and came up with a flexible environment where we independently tested each of the three ESBs with the same application and product mix.

Figure 4-1 on page 59 shows a CICS region communicating to WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server through an ESB. In the ESB diagram, we configure one of the three ESBs for each test scenario.

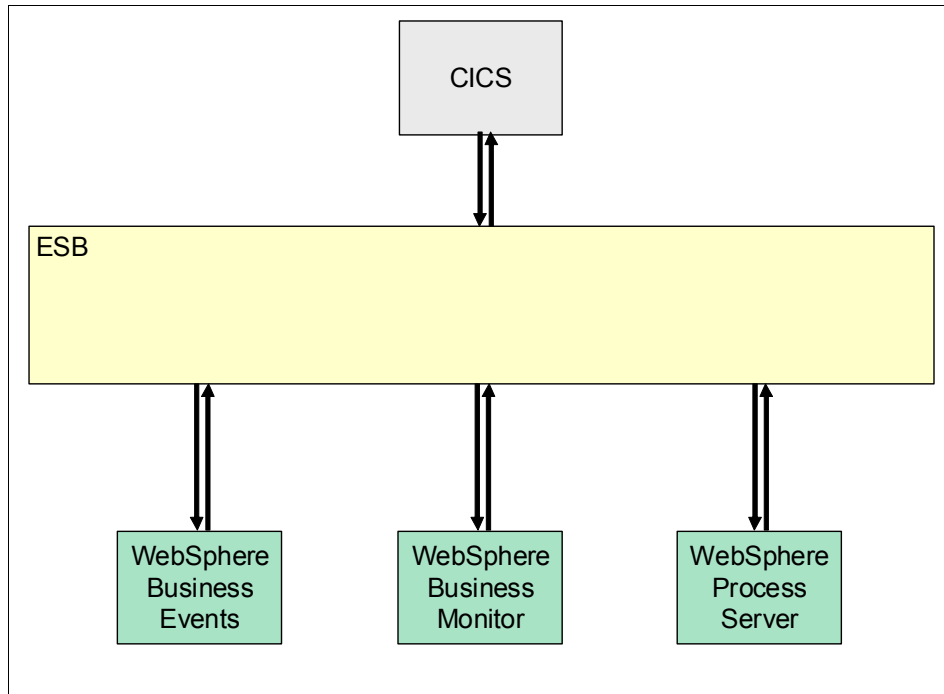


Figure 4-1 Overall architecture diagram

4.3 The catalog manager example application

CICS Transaction Server V3.1 ships with a sample application called the *CICS catalog manager example application* to help you set up, configure, and test connecting CICS applications to external clients and servers.

For this book, we run the catalog manager example application in our CICS region to drive our event scenarios using the ESB as an integration platform.

We summarize the catalog manager application. For complete details, consult the CICS Transaction Server V3.1 Information Center or another information center by searching *CICS Catalog Manager Example Application*.

Figure 4-2 on page 60 shows the components that make up our implementation of the catalog manager example application running in CICS. We used the application as supplied; therefore, we did not modify any of the supplied modules or components.

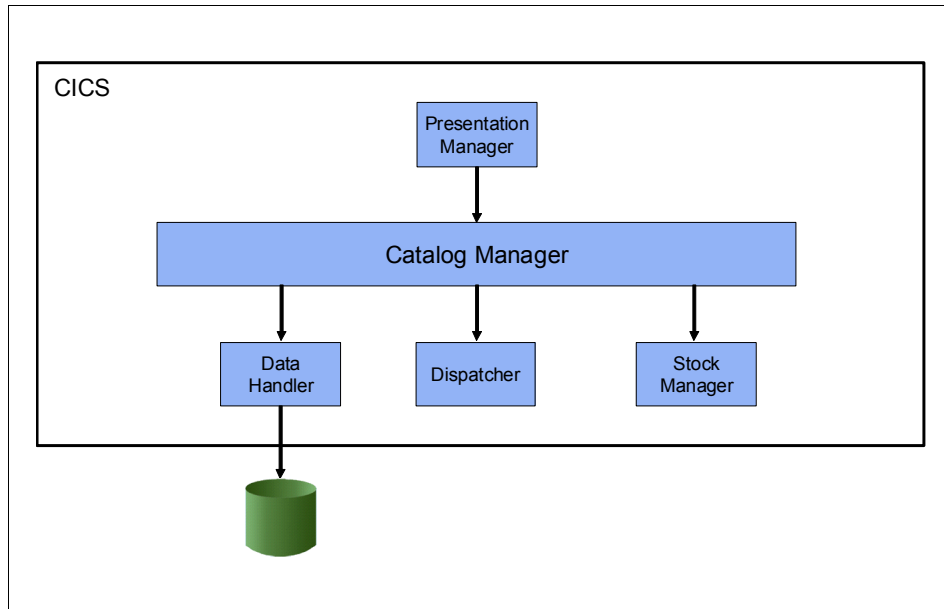


Figure 4-2 Catalog manager example application components diagram

The catalog manager example application is an office supply sales catalog and order processing application that allows you to list, inquire about, and order products out of an office items catalog that is stored in a VSAM file.

The application is made up of a Presentation Manager module, DFH0XGUI, that communicates to a Catalog Manager interface module, DFH0XCMN, which is a mainline driver routine that buffers the front-end and back-end components. The data is stored in a VSAM file and is accessed by one of the back-end modules: the File Manager, DFH0XVDS, the Dispatch Manager, DFH0XSOD, or the Stock Manager, DFH0XSSM.

In our diagram, the file is only attached to the Data Handler module, because our Dispatcher and Stock Manager modules are dummy stubs and do not perform any function. If desired, you can modify them to interact with the EMPCAT file.

There are a several ways to configure the application. We use the configuration transaction, ECFG, as shown in Figure 4-3 on page 61.

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 ==> myserver:99999
Outbound WebService URI ==> http://myserver:80/exampleApp/dispatchOrder
==>
==>
==>
==>
==>
```

Figure 4-3 Catalog manager example application configuration parameters from transaction ECFG

The Datastore Type lets you choose to configure a STUB program, DFHOXSDS, as the Data Handler program to run as an IEFBR14 style program or to configure the VSAM module, DFHOXVDS, which uses an actual VSAM file to provide sample stock inventory data.

We left all of the options as defaults, which means that the Dispatcher and Stock Manager are both dummy stubs that do nothing but return to the caller.

This configuration allows us to get a functional application up and running that gives us the ability to order products and affects the actual stock available (or on hand) values stored in our EXMPCAT VSAM stock file.

Our plan is to utilize the catalog manager example application as the major driver for our testing of our event scenarios. One problem that we ran into is restocking our data in the inventory file. When you install the application, the EMPCAT file is loaded with initial stock values and can start running out of stock extremely quickly. Because we configure the application using the default Stock Manager module, DFHOXSSM, there is no provision to restock our inventory.

Obviously, we might have modified the Stock Manager module to provide this function. However, because we are demonstrating adding or modifying the functionality of an existing application without touching it, for example, in our

fictitious sample, we assume that the catalog manager example application has been installed and running for years. Perhaps our programming staff is not trained in the complexities of the application design. Or, perhaps part of the source code has been lost and we are adding new functionality to the application without touching the original application.

To solve our problem, we create a restock program that, when given a stock item reference number, simulates ordering and restocking that specific item. Figure 4-4 shows our Restock Item module added to our application diagram, accessing the EMPCAT VSAM file.

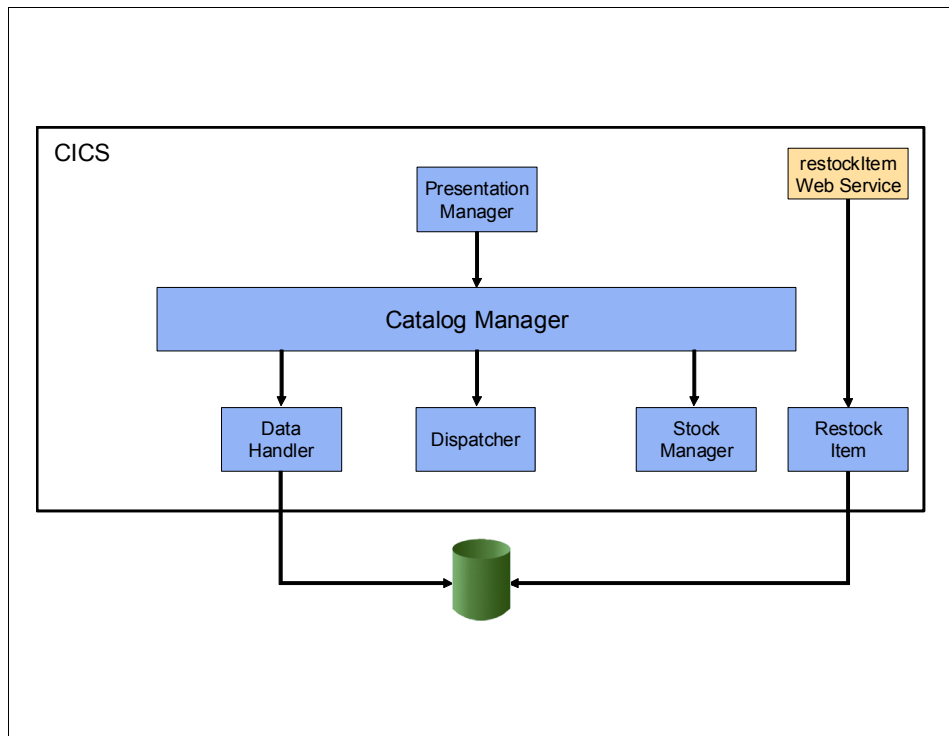


Figure 4-4 Adding the Restock Item component

We simulate a modification to an existing application by using our business process management suite of products and the integration with our ESB. By using event processing and the integration functionality of our ESBs, we successfully add and modify the functionality of our existing application without altering the original code. The catalog manager example application runs untouched as though you unloaded and installed it directly from CICS. We add new features and functionality to it, demonstrating the power of event processing

and ESB technologies. We include the source for our Restock module in Appendix A, “Additional material” on page 401.

The catalog manager example application provides Web Service front ends to the various functions, List, Inquire, and Place Order. Figure 4-5 shows how to access our application back-end components, along with our Restock Item module, as a Web Service.

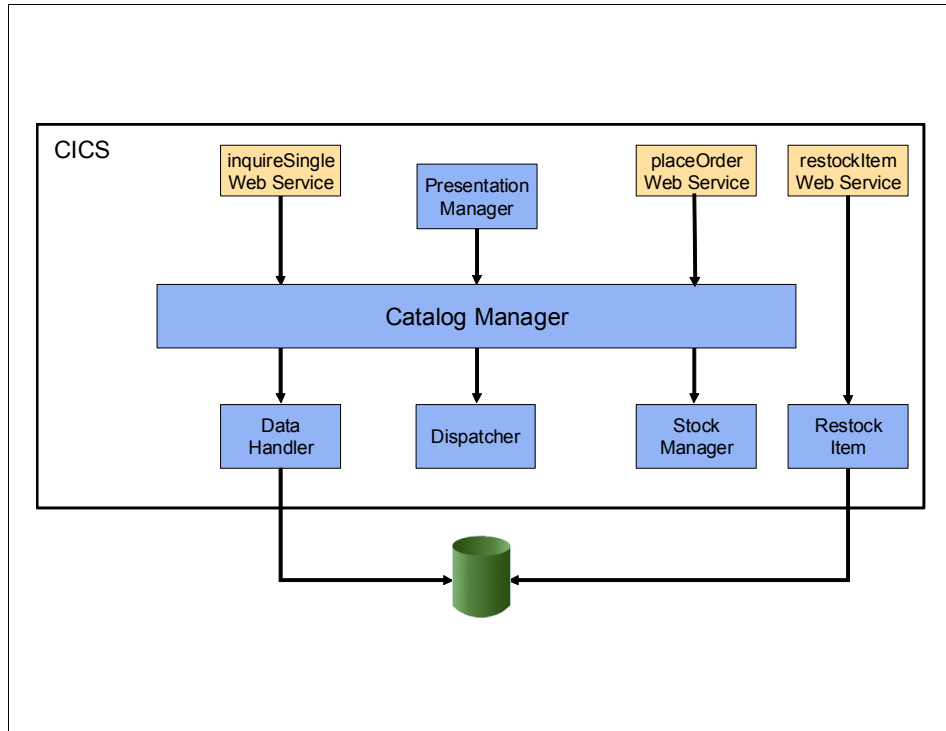


Figure 4-5 Web Service front-ends diagram

We chose to use the interfaces that are shown in Figure 4-5, but CICS also provides three additional wrapper programs that present a much cleaner interface to these same functions. Figure 4-6 on page 64 shows how we can alternatively access our application back-end components as Web services that invoke a wrapper program to provide a cleaner interface to the Catalog Manager COBOL program interface module, DFH0XCMN.

Typically, you do not use the cleaner interfaces that are provided by the wrapper programs, but in our example, we show you how the ESB can buffer the interfaces of your back-end applications. Either interface can work with our ESB. We chose one interface, but the alternative wrapper program interfaces can work, too.

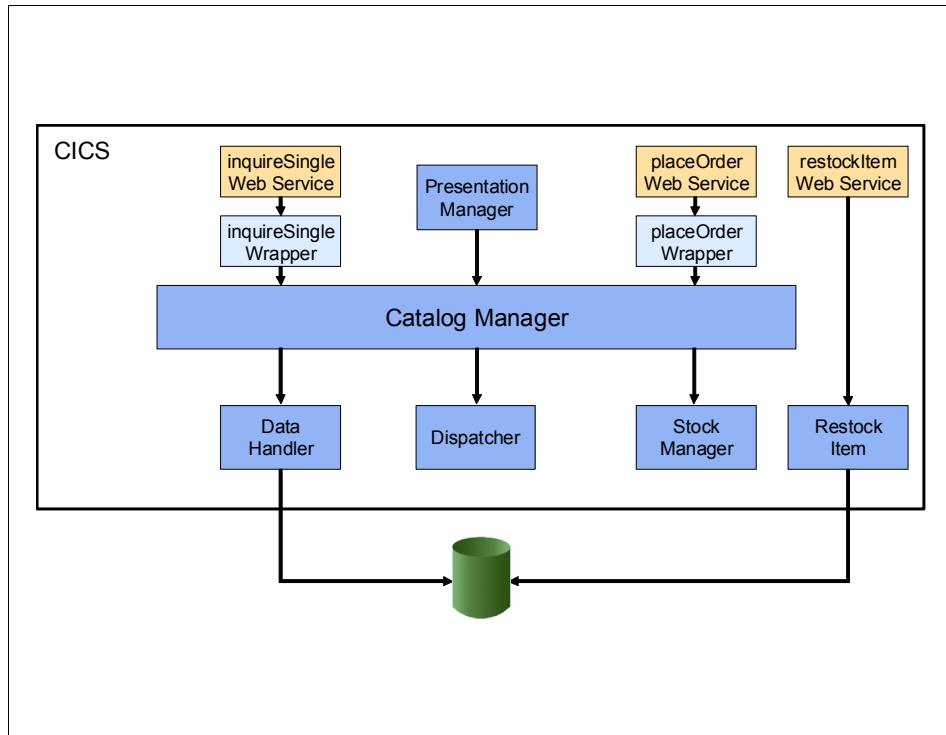


Figure 4-6 Improved Web Service front-ends diagram

If you have an application that uses an older style interface similar to our previous example and you do not want to clean up that interface in CICS, you can use an ESB to provide a separate interface without having to write a wrapper program in CICS. The disadvantage is that if the original interface in CICS changes, you now have a second interface to update. You must consider this disadvantage in your governance and production turnover procedures. Also, if response time is critical, you have introduced additional code. When problems occur, you have an additional component that might be the problem.

Our objective is to show you the flexibility that is available. You can configure your environment to take advantage of the flexibility to combine products using the ESB to hide or buffer certain interfaces.

4.4 ESB structure

We chose to use all three ESB products in our testing scenarios: WebSphere Enterprise Service Bus, DataPower, and WebSphere Message Broker. Also, we wanted to work WebSphere Process Server into the testing.

We decided to build a flexible test environment that allowed us to plug in one of the three ESB products, with the capability to have one or more tests running at the same time. We built the architecture that is shown in Figure 4-7.

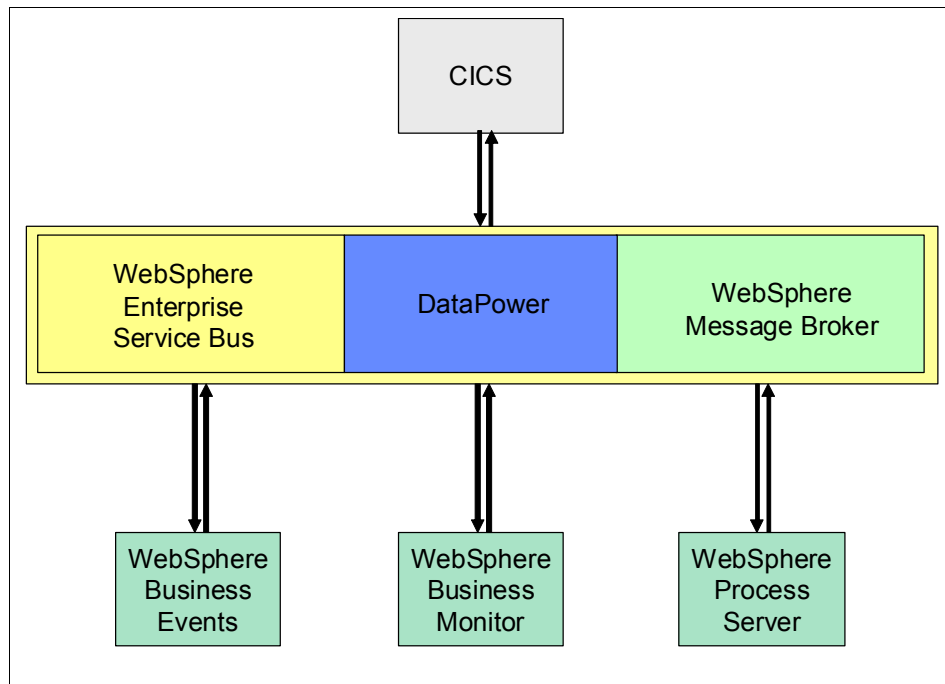


Figure 4-7 ESB structure diagram

In the ESB structure diagram that is shown in Figure 4-7, we show a CICS Transaction Server V4.1 test region interfacing with an ESB, which can be any of the three ESB products: WebSphere Enterprise Service Bus, DataPower, or WebSphere Message Broker. Then, the diagram shows the ESB interface with our three business process management products: WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server.

CICS is running our catalog manager example application, which generates events to be handled by WebSphere Business Events, WebSphere Business

Monitor, or WebSphere Process Server. We further explain these event scenarios in 4.8, “Event scenarios used by our team” on page 68.

In our test environment, we configure a single ESB product into the model that we have described and create three copies that allow us to test the same scenario using all three ESB products separately. In an actual implementation, you might have a more complex environment with one or more ESBs and many instances of our products, along with many other products, plugging into the ESB. But, we kept our test simple.

4.5 Using WebSphere Enterprise Service Bus as the ESB

Using the model outlined in our ESB structure diagram in Figure 4-7 on page 65, we configured a CICS Transaction Server V4.1 region to communicate to an instance of WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server through a WebSphere Enterprise Service Bus, as shown in Figure 4-8.

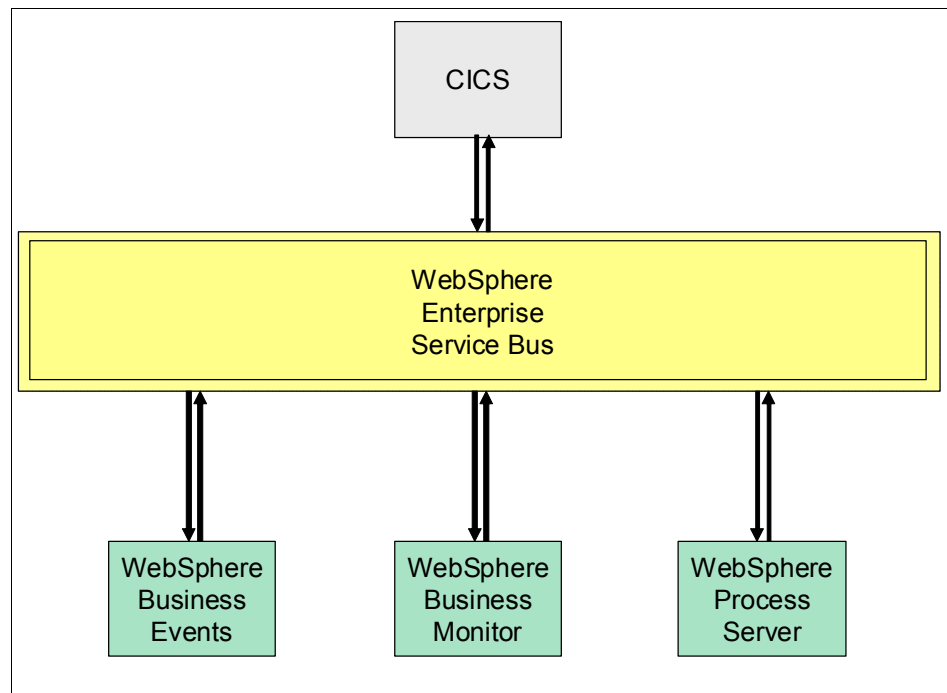


Figure 4-8 WebSphere Enterprise Service Bus diagram

We configured our catalog manager example application to run the same event scenarios, communicating to the same product mix and setup that were used in all our other ESB setups.

4.6 Using DataPower as the ESB

Using the model outlined in our ESB structure diagram in Figure 4-7 on page 65, we configured a CICS Transaction Server V4.1 region to communicate to an instance of WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server through a DataPower XI50, as shown in Figure 4-9.

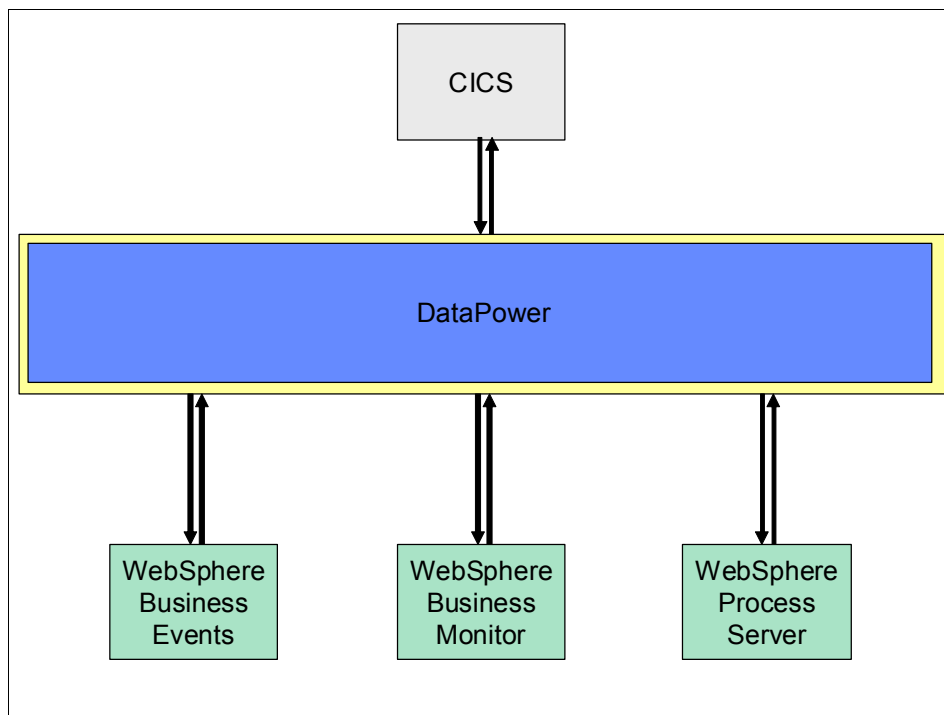


Figure 4-9 DataPower diagram

We configured our catalog manager example application to run the same event scenarios, communicating to the same product mix and setup that were used in all our other ESB setups.

4.7 Using WebSphere Message Broker as the ESB

Using the model outlined in our ESB structure diagram in Figure 4-7 on page 65, we configured a CICS Transaction Server V4.1 region to communicate to an instance of WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server through a WebSphere Message Broker, as shown in Figure 4-10.

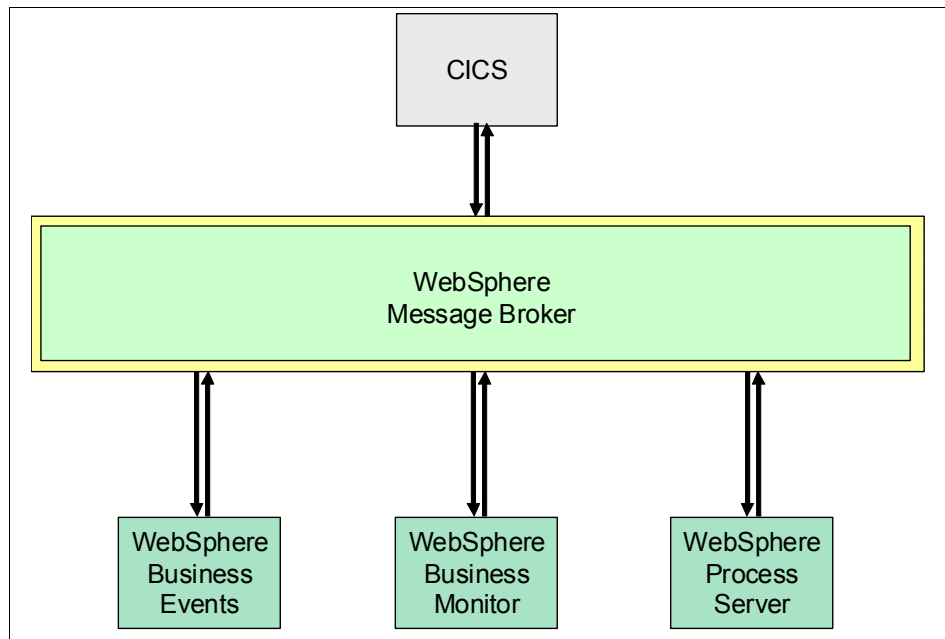


Figure 4-10 WebSphere Message Broker diagram

We configured our catalog manager example application to run the same event scenarios, communicating to the same product mix and setup that were used in all our other ESB setups.

4.8 Event scenarios used by our team

Our major focus was integrating an ESB with our existing applications and products to show how you can enhance and integrate your existing environment with event-based processing technology. Because our focus was not on the actual event processing setup but on integration with an ESB, we tried to keep our actual event scenarios simple and limited our testing to the four event scenarios that are listed in Figure 4-11 on page 69.

For more information about event processing with CICS using WebSphere Business Events and WebSphere Business Monitor, see the companion book to this publication called *Implementing Event Processing with CICS*, SG24-7792.

For more information about the CICS Explorer, which we used to set up events in CICS and to monitor and manage the CICS environment, see *CICS Explorer*, SG24-7778.

Scenarios	Actions
<p>① Successful Order</p>	<p><i>Enrichment, WBE Analysis, Log</i></p>
<p>② Multi High Value Orders in 3 days</p>	<p><i>Email Coupon, Log</i></p>
<p>③ Failed Order - Insufficient Stock</p>	<p><i>Drive Re-Stock Order, WBE Analysis, Log</i></p>
<p>④ Multi Insufficient Stock Failures</p>	<p><i>Increase Re-Order Size Point, Log</i></p>

Figure 4-11 Event scenario chart

The CICS region running the catalog manager example application created two events, one successful event and one failure event, which both relate to ordering products from our catalog.

We generated a total of four events, two events from CICS, one event representing a positive situation, and one event representing a failure situation. We then complemented each of those events with events that were generated from WebSphere Business Events based on multiple CICS events being received and combined in a complex scenario.

Additionally, not listed in the diagram, we also added a message to a Publishing/Subscription queue to notify any interested party when a High Value order takes place.

4.8.1 Successful events

Scenario 1 is a successful event that is generated by CICS when a product is ordered. Refer to Figure 4-12 and Figure 4-13 on page 71 to see where in the processing cycle the event is captured.

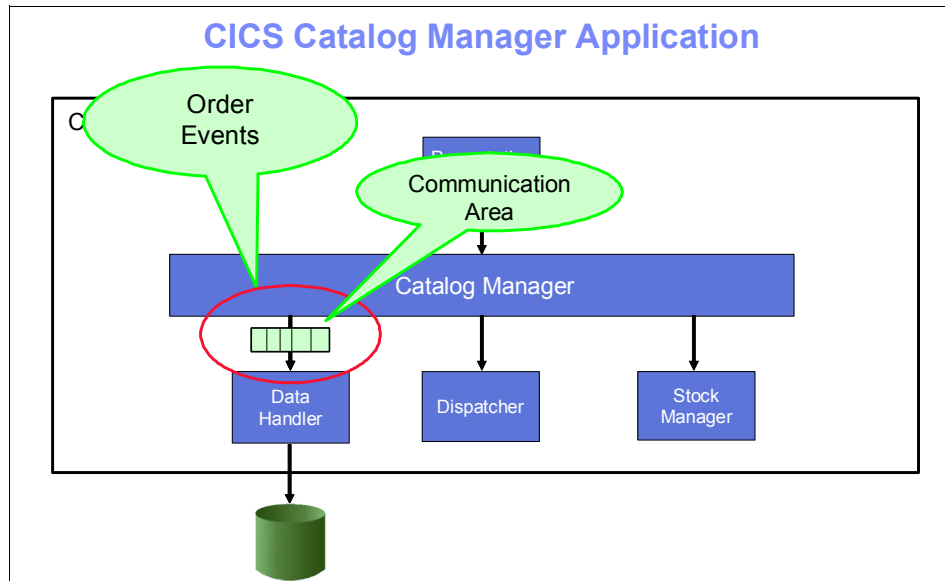


Figure 4-12 Emitting successful order events diagram

The emitting successful order events diagram in Figure 4-12 shows the data flowing in the communications area (COMMAREA) from the Catalog Manager module to the Data Handler module to trigger and build our event payload.

In Figure 4-13 on page 71, we show the expanded structure of the COMMAREA, identifying the fields inside the COMMAREA that are used for event processing and capture.

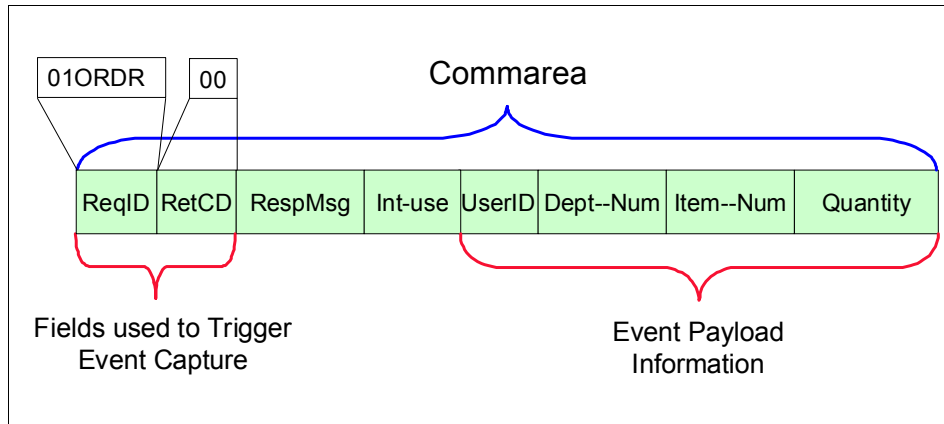


Figure 4-13 Successful order COMMAREA diagram

The successful order event is triggered by the ReqID Field being equal to 010RDR and a RetCD value of 00. When the event is triggered, we capture and build the event payload using the UserID, Dept--Num, Item--Num, and Quantity fields.

Scenario 2, depicted in Figure 4-14 on page 72, is an event generated by WebSphere Business Events when multiple orders have been received, exceeding a set value over a defined period of time.

Description: This scenario looks for three orders that are over USD200 for the same customer in four days.

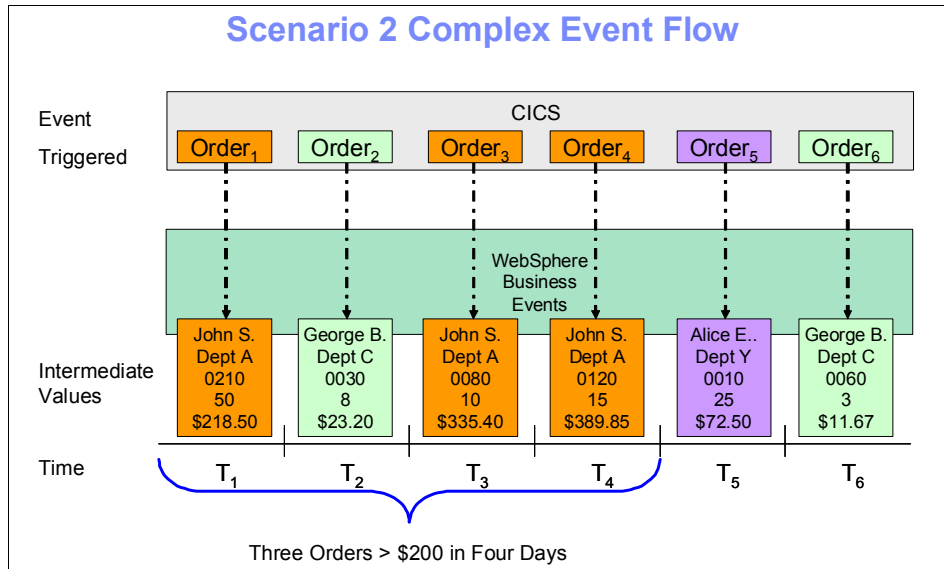


Figure 4-14 Scenario 2 complex event flow

In scenario 2, complex event flow, in Figure 4-14, we show time units of T_1 , T_2 , T_3 , and so on where the time units can be days, weeks, or months.

In our actual testing, we configure WebSphere Business Events to monitor all successful orders, looking for a pattern of orders placed by the same customer where the orders total over a set value and all orders are placed within a three-day time frame.

4.8.2 Failure events

Scenario 3 is a failure event generated by CICS when a product is ordered but fails due to an insufficient stock available condition. See Figure 4-15 on page 73 and Figure 4-16 on page 73 to see where in the processing cycle the event is captured.

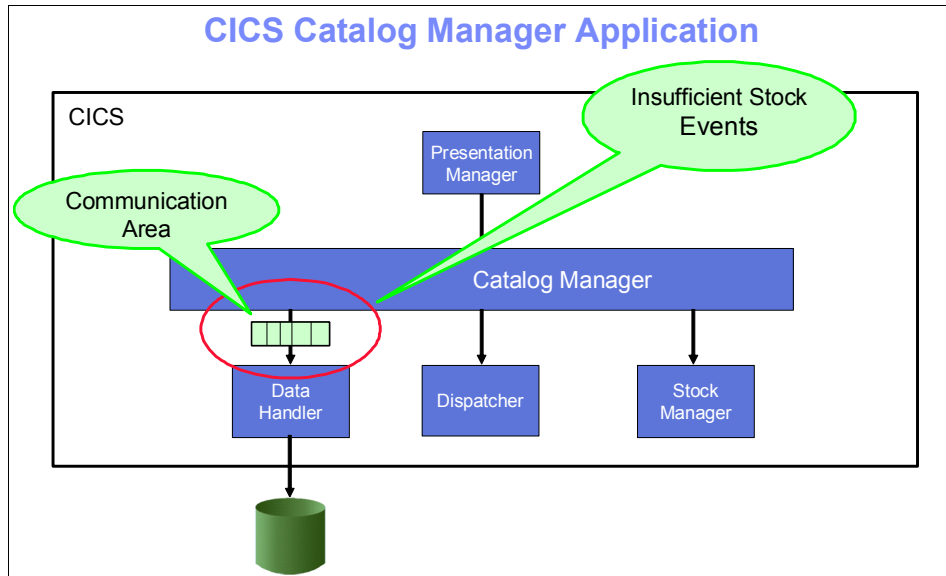


Figure 4-15 Emitting failed order events diagram

The emitting failed order events diagram that is shown in Figure 4-15 shows that we are looking at the data flowing in the COMMAREA from the Catalog Manager module to the Data Handler module to trigger and build our event payload.

In Figure 4-16, we show the expanded structure of the COMMAREA, identifying the fields inside the COMMAREA that are used for event processing and capture.

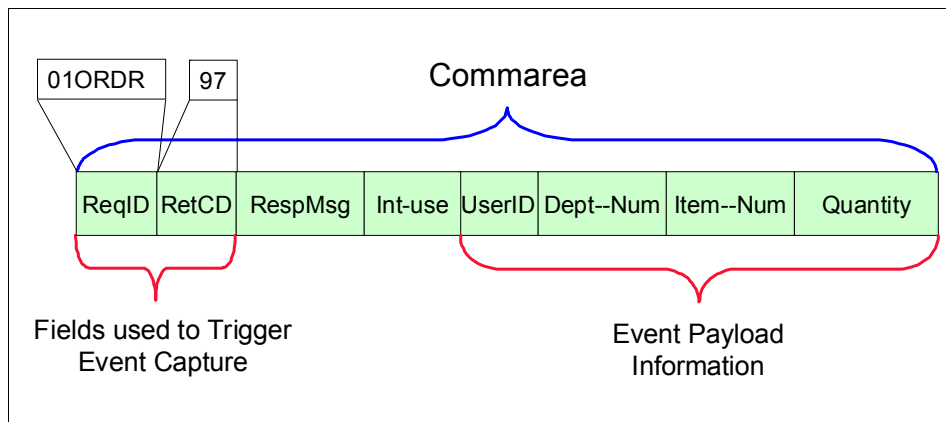


Figure 4-16 Insufficient stock COMMAREA diagram

The failed order event is triggered by the ReqID Field being equal to 010RDR and a nonzero RetCD value of 97. When the event is triggered, we capture and build the event payload using the UserID, Dept--Num, Item--Num, and Quantity fields.

Scenario 4, which is depicted in Figure 4-17, is an event that is generated by WebSphere Business Events when multiple orders have failed repeatedly for the same product over a defined period of time.

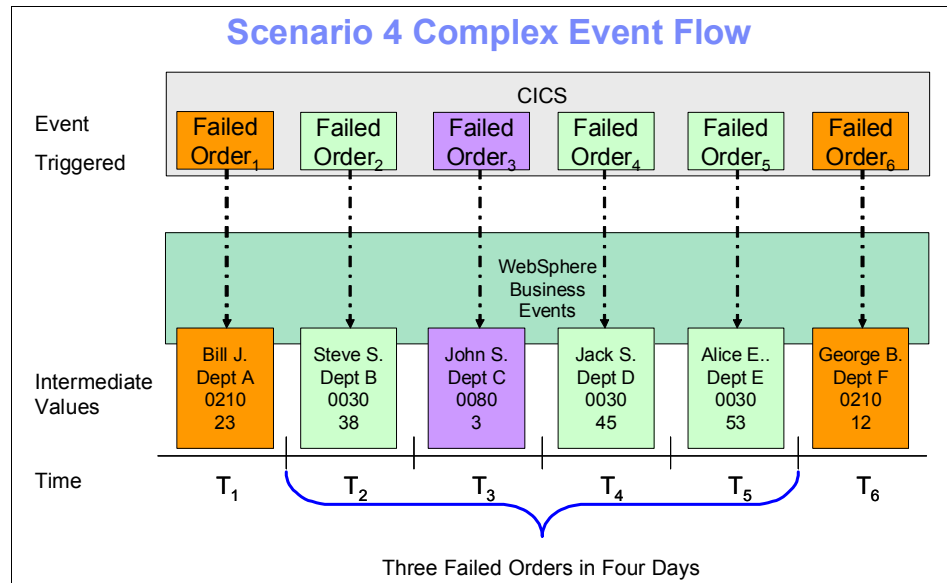


Figure 4-17 Scenario 4 complex event flow

In our scenario 4 complex event flow diagram in Figure 4-17, we show time units of T₁, T₂, T₃, and so on where our Time units can be days, weeks, or months.

In our testing, we configured WebSphere Business Events to monitor all failed orders, looking for a pattern of repeat failed orders for the same product Item--Num over a period of three days. We assumed that this repeat failure situation indicated that our Restock Item process was deficient and that we needed human interaction to adjust the restock process.

4.8.3 Scenario 1: Successful order event

The successful order event is generated from the catalog manager example application transaction, EGUI, running in our CICS region.

Whenever a successful product order is placed, CICS emitted an event on the WebSphere MQ (WMQ) EP adapter in common base event (CBE) XML format.

We emitted the event to our ESB so that it was able to perform enrichment, transformations, and routing.

An alternative is to emit the event directly to WebSphere Business Events; however, because this book is about integration with an ESB that allows us to perform the enrichment and transformation, we send all messages to the ESB for processing. This approach allows us to manage a flexible environment. For instance, we plug additional products into the bus and integrate them into our existing event processing mix with minimal configuration.

Event setup

We use the CICS Event Binding editor to configure CICS to emit an event whenever the Data Handler program, DFH0XVDS, is called with a COMMAREA parameter equal to 010RDR and successfully completes with a valid return code of zero.

Information emitted

The successful order event emits the following four fields that are shown in Example 4-1 in a CBE format to our ESB.

Example 4-1 Successful order event-emitted business information

Order Item Number
Order Quantity
User Name
Charge Dept

We describe the full details about the event setup and configuration in 6.2.1, “Configuring CICS” on page 133.

Event flow

Using the diagram in Figure 4-18 on page 76, we explain the event flow.

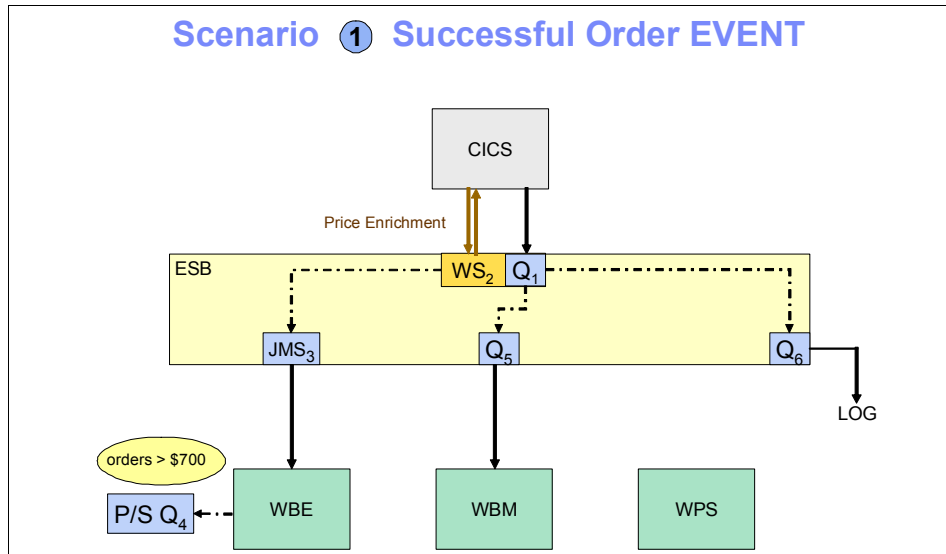


Figure 4-18 Scenario 1: Successful order event diagram

The following steps describe the successful order event scenario:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter onto a queue that is received by the ESB, identified as Q₁ in the diagram.
2. The ESB receives the CBE event and then fires off a Web Service call, inquireSingle, located in box WS₂, to enrich the payload with the product item price. The price is in the VSAM inventory file that is available to CICS.
3. The ESB transforms the input event, changing the format from a CBE format to a WebSphere business event (WBE) format, and sends the enriched event to WebSphere Business Events runtime through a Java Message Service (JMS) topic, JMS₃.
4. WebSphere Business Events receives and evaluates the event. If the event represents an order over USD700, the information is placed on a publish/subscribe (Pub/Sub) queue for interested parties through Q₄.
5. The ESB transfers the original CBE-formatted event to Q₅ for consumption by WebSphere Business Monitor.
6. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₆.

Managing traffic flow

Our environment is not necessarily realistic. One of our goals was to create three identical test environments to plug in and test each of the three ESB products. We set up our architecture to use MQ Series queues to communicate among all our products. We developed a queue naming standard, which is not usable outside of our environment, but it works for us.

You must create an architecture and establish naming conventions to allow all your applications and products to communicate and to allow plug-in compatibility.

Figure 4-19 shows the scenario diagram, along with our MQ queue names.

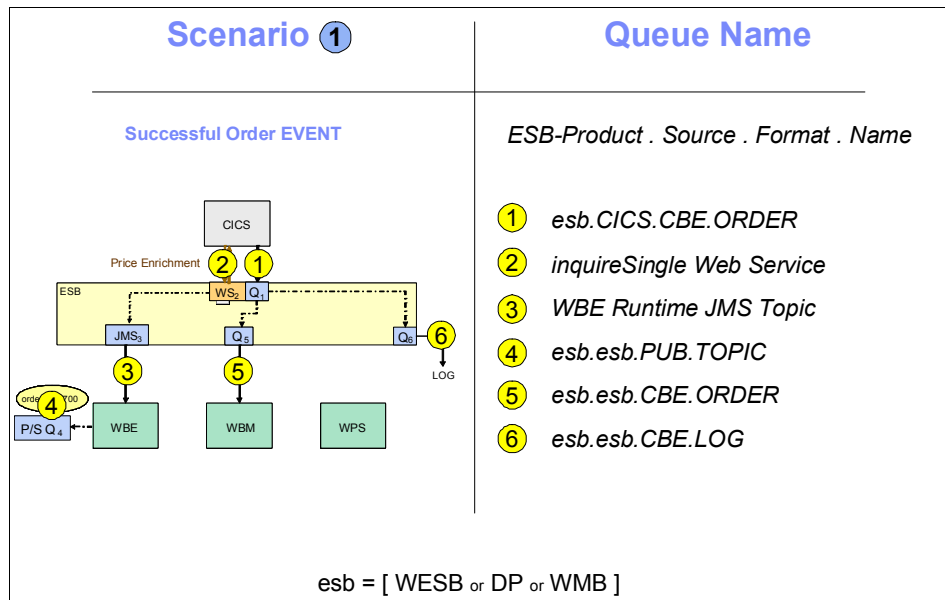


Figure 4-19 Scenario 1: Successful order event queue mapping diagram

Our queue names consist of four parts:

- ▶ The ESB product that we used or tested for this run
- ▶ Where the message came from
- ▶ The data type format of the data flowing over the queue
- ▶ The queue name

We also included the Web Service resource names in Figure 4-19 to account for all the parts.

4.8.4 Scenario 2: Multiple high value orders in three days event

The multiple high value order event is created by WebSphere Business Events by reviewing each CICS event from scenario 1 and matching orders placed by the same person over a defined period of time whose total order value exceeds a trigger value, indicating a high value customer.

Whenever a successful product order is placed, CICS emits an event on the WebSphere MQ (WMQ) EP adapter in CBE format. We emit the event to our ESB so that it can perform enrichment, transformations, and routing.

Event setup

We used the WebSphere Business Events Design tool to create an interaction set to execute the business logic for this flow. WebSphere Business Events normalizes the incoming event data for use in the evaluation of the interaction blocks contained within the interactions set.

Information emitted

In the scenario, when the system identifies a high value customer, the company sends that customer a coupon; thus, the output is an email.

We describe the scenario event setup and configuration in detail in Chapter 9, “WebSphere Business Events scenario” on page 249.

Event flow

Using the diagram in Figure 4-20 on page 79, we explain the event flow.

Scenario ② Multi High Value Orders in 3 days EVENT

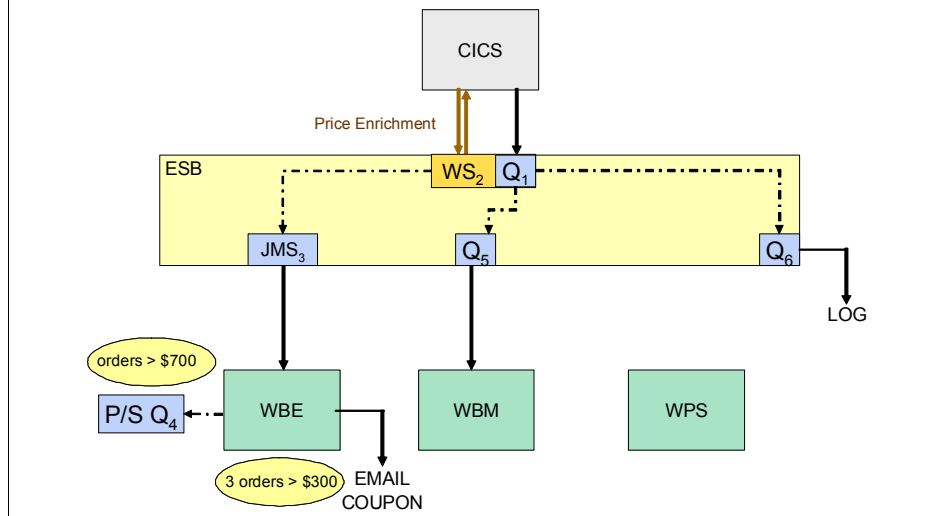


Figure 4-20 Scenario 2: Multiple high value orders in three days event diagram

Scenario 2 consists of these steps:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter onto a queue that is received by the ESB, identified as Q₁ in the diagram.
2. The ESB receives the CBE event and then fires off a Web Service call, inquireSingle, which is located in box WS₂, to enrich the payload with the product item price. The price is in the VSAM inventory file available to CICS.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, JMS₃.
4. WebSphere Business Events receives and evaluates the event. If the event represents an order over USD700, the information is placed on a Pub/Sub queue for interested parties through Q₄.
5. WebSphere Business Events reviews the order, comparing it with previous orders by the same customer, checking to see if the total value exceeds a trigger value during a specified time window. If true, a complex scenario has occurred and WebSphere Business Events generates an event that results in sending the customer a coupon through email.
6. The ESB transfers the original CBE-formatted event to Q₅ for consumption by WebSphere Business Monitor.

7. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₆.

Managing traffic flow

Figure 4-19 on page 77 shows the scenario diagram along with the MQ queue names used for scenario 2.

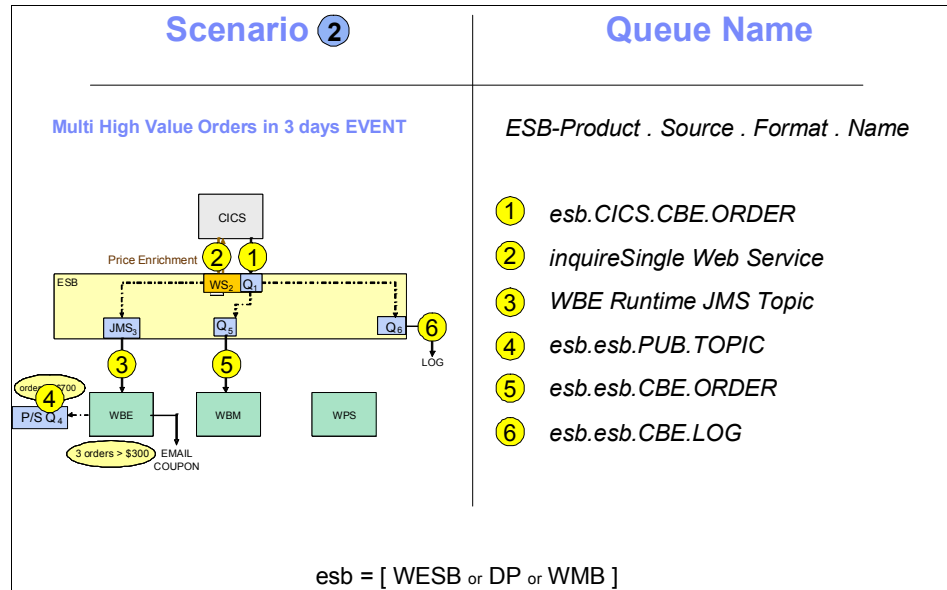


Figure 4-21 Scenario 2: Multiple high value orders in three days event queue mapping

Our queue names consist of four parts:

- ▶ The ESB product that we used or tested for this run
- ▶ Where the message came from
- ▶ The data type format of the data flowing over the queue
- ▶ The queue name

We also included the Web Service resource names in Figure 4-21 to account for all the parts.

4.8.5 Scenario 3: Failed order due to insufficient stock event

The catalog manager example application transaction, EGUI, running in our CICS region, generates the failed order event.

Whenever a product order is placed and insufficient stock exists to fill the order, the catalog manager example application causes the order to fail and CICS emits

an event on the WebSphere MQ (WMQ) EP adapter in CBE format. We emit the event to our ESB so that it can perform enrichment, transformations, and routing.

We also can emit the event directly to WebSphere Business Events. However, this book is about integration with an ESB; therefore, we are sending all messages to the ESB for processing. This action allows us to manage a flexible environment. For instance, we can install multiple products and run them even if we have more products to install.

Event setup

We used the CICS Event Binding editor to generate a CICS event whenever the Data Handler program, DFH0XVDS, is called with a COMMAREA parameter equal to 010RDR and fails to fill the order due to an insufficient stock condition, which is indicated by a return code of 97.

Information emitted

The failed order event emits the following four fields that are shown in Example 4-2 in a CBE format to our ESB.

Example 4-2 Failed order event-emitted business information

Order Item Number
Order Quantity
User Name
Charge Dept

We describe the event setup and configuration in 6.2.1, “Configuring CICS” on page 133.

Event flow

Using the diagram in Figure 4-18 on page 76, we explain the event flow.

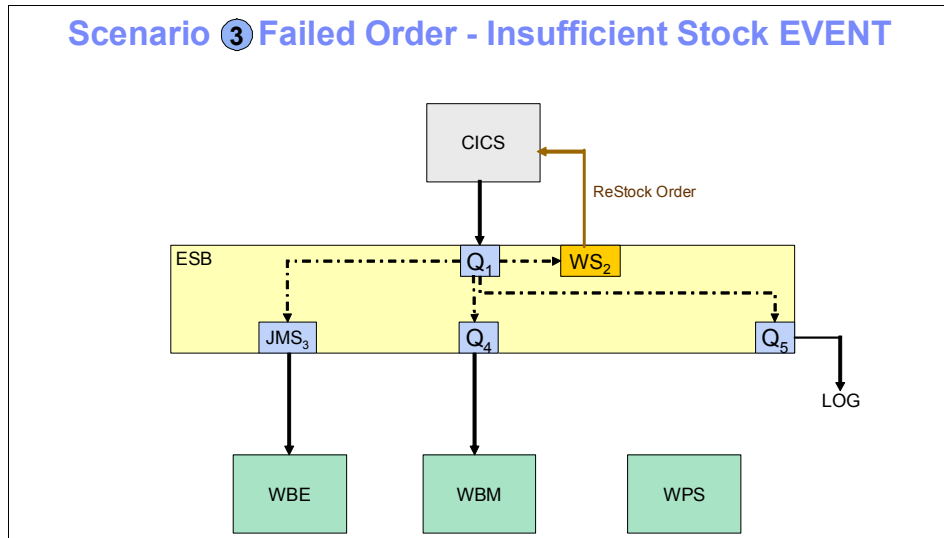


Figure 4-22 Scenario 3: Failed order due to insufficient stock event diagram

Figure 4-22 shows the following steps:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter onto a queue that is received by the ESB, identified as Q₁ in the diagram.
2. The ESB receives the CBE event and then fires off a Web Service call, restockItem, which is located in box WS₂, to generate a reorder of the stock to meet future orders. Our simple application performs a basic stock order. We did not build in intelligence to calculate how much to order. We always order a predefined amount, such as 50 units.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, JMS₃.
4. The ESB transfers the original CBE-formatted event to Q₄ for consumption by WebSphere Business Monitor.
5. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₅.

Managing traffic flow

Our environment is not realistic, because it was designed for demonstration purposes. We wanted to create three identical test environments to plug in and test each of the three ESB products. We set up our architecture to use MQ

Series queues to communicate among all our products. Therefore, we created a queue naming standard, which works for us.

You must create an architecture and naming conventions to allow all your applications and products to communicate, and also allow plug-in compatibility.

Figure 4-19 on page 77 shows the scenario diagram along with our MQ queue names.

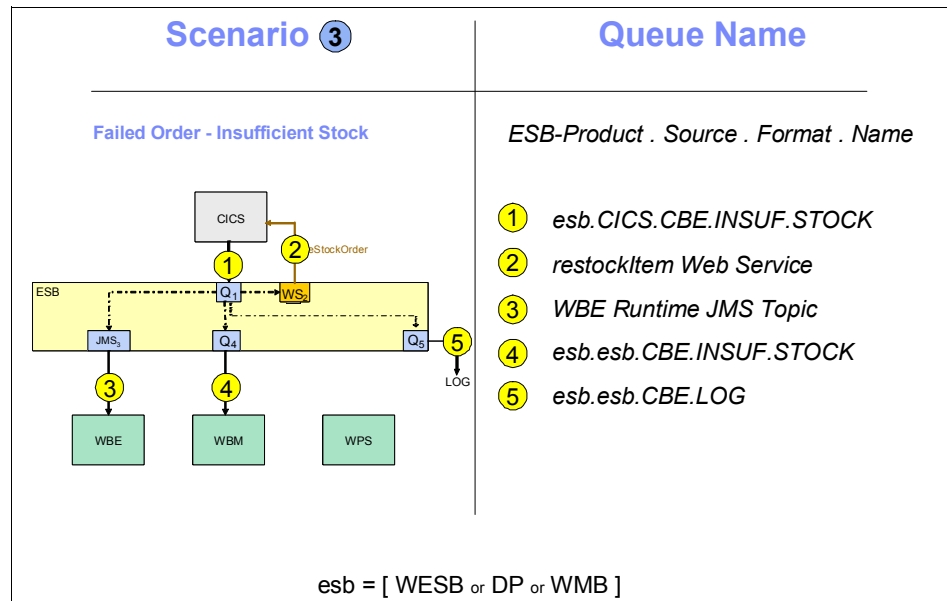


Figure 4-23 Scenario 3: Failed order due to insufficient stock event queue mapping

Our queue names consist of four parts:

- ▶ The ESB product that we used or tested for this run
- ▶ Where the message came from
- ▶ The data type format of the data flowing over the queue
- ▶ The queue name

We also included the Web Service resource names in Figure 4-23 to account for all the parts.

4.8.6 Scenario 4: Multiple insufficient stock failures event

The multiple insufficient stock failure event is an event created by WebSphere Business Events by reviewing each CICS event from scenario 3 and matching failed orders for the same product item number over a defined period of time.

This task indicates that the reorder process does not work and needs human intervention to identify why orders continue to fail.

WebSphere Process Server enriches the information that it receives with the item prices. If the failed orders are greater than a predetermined value, the system contacts a manager along with dispatching a person to review the process.

Event setup

We used the WebSphere Business Events Design tool to create an interaction set to execute the business logic for this flow. WebSphere Business Events normalizes the incoming event data for use in the evaluation of the interaction blocks contained within the interaction set.

Information emitted

The failed order event sends the following four fields that are shown in Example 4-3 to WebSphere Process Server through a Web Service invocation for further processing.

Example 4-3 Failed order event-emitted business information

Order Item Number
Order Quantity
User Name
Charge Dept

We describe the event setup and configuration in detail in Chapter 9, “WebSphere Business Events scenario” on page 249.

Event flow

Using the diagram in Figure 4-24 on page 85, we explain the event flow.

Scenario 4 Multi Insufficient Stock Failures EVENT

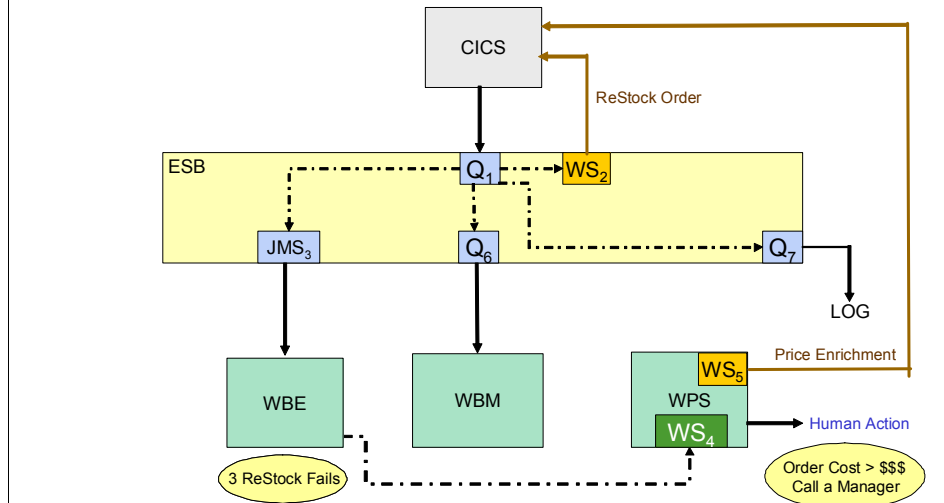


Figure 4-24 Scenario 4: Multiple insufficient stock failures event diagram

The following steps describe the event flow:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter onto a queue, which is received by the ESB, identified as Q₁ in the diagram.
2. The ESB receives the CBE event and then fires off a Web Service call, restockItem, which is located in box WS₂, to generate an order and replenish the available or on-hand inventory for the item that was involved in the failed order.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, JMS₃.
4. WebSphere Business Events reviews the failed order, comparing it with previous failed orders by the same product item number, and checking to see if the number of failures exceeds a trigger value within a specified time window. If true, a complex scenario has occurred, and WebSphere Business Events generates and sends a request to WebSphere Process Server through a Web Service call, requestFailedOrderIntervention, which is located in box WS₄. WebSphere Process Server requests that a person get involved to review the reorder process to make the necessary adjustments to prevent future failed orders. Additionally, if the failed orders are over a specific value, WebSphere Process Server also makes a request to get a manager involved.

5. The ESB transfers the original CBE-formatted event to Q₆ for consumption by WebSphere Business Monitor.
6. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₇.

Managing traffic flow

Figure 4-19 on page 77 shows the scenario diagram, along with the MQ queue names used for scenario 4.

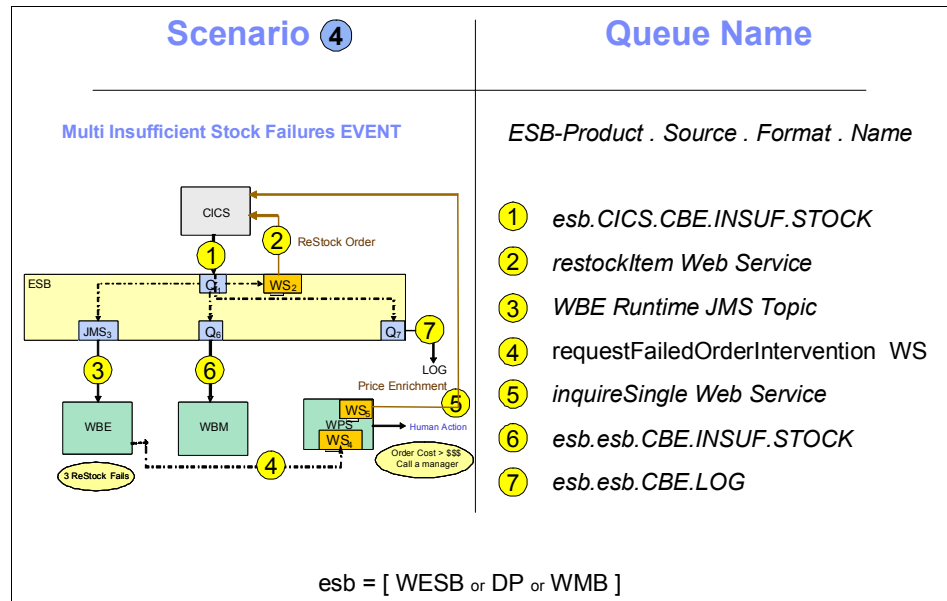


Figure 4-25 Scenario 4: Multiple insufficient stock failures event queue mapping diagram

Our queue names consist of four parts:

- ▶ The ESB product that we used or tested for this run
- ▶ Where the message came from
- ▶ The data type format of the data flowing over the queue
- ▶ The queue name

We also included the Web Service Enrichment resource names in Figure 4-25 to account for all the parts.

4.9 Testing each scenario

Our implementation of the catalog manager example application is terminal-based and driven by a person logging onto CICS and executing the EGUI transaction. Because this book is not a performance-based book, our testing was low volume and limited to driving our test scenarios only. Therefore, you will not see any high volume scenarios.

One change that we made was to simulate events over time, such as three high value orders over three days. We were unable to wait days or weeks to produce results, so we adjusted our actual parameters to force the environment to actually trigger and emit several events. Figure 4-26 shows the input sources to our scenarios.

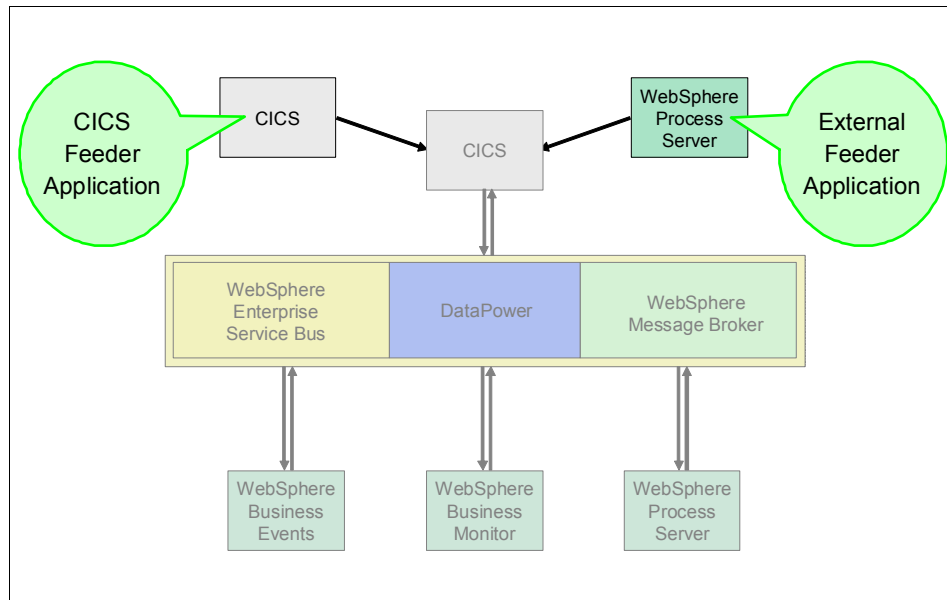


Figure 4-26 Sources of Input to the CICS catalog manager example application

We considered having WebSphere Process Server send events; however, we did not have enough time to implement this part.



Part 3

Scenarios

In this part of the book, we show you several enterprise service bus (ESB) scenarios. We have included the Customer Information Control System (CICS) environment setup in each scenario chapter. However, with regard to the setup for WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server, unless it is specific to the ESB, it is in a separate chapter. Therefore, we have three chapters for these products in this sequence:

- ▶ WebSphere Enterprise Service Bus business scenario
- ▶ WebSphere Message Broker business scenario
- ▶ DataPower business scenario
- ▶ Scenario flow
- ▶ WebSphere Business Events
- ▶ WebSphere Business Monitor
- ▶ WebSphere Process Server



WebSphere Enterprise Service Bus business scenario

In this chapter, we discuss the scenarios from the point of view of WebSphere Enterprise Service Bus acting as the enterprise service bus (ESB). We examine details of the following topics:

- ▶ Overview of the environment with WebSphere Enterprise Service Bus
- ▶ Configuration of the environment with WebSphere Enterprise Service Bus
- ▶ Implementation and test of the first and third scenarios

5.1 Environment overview

We perform all of our development work for the WebSphere Enterprise Service Bus solution on Microsoft® Windows XP using WebSphere Integration Developer V7. Included in the WebSphere Integration Developer tooling is a WebSphere Enterprise Service Bus server that we configure and use for our unit and functional testing. After we test the WebSphere Enterprise Service Bus artifacts, we can deploy them to a WebSphere Enterprise Service Bus server running in an integration test environment. This server can be on a separate platform or might be configured in a more robust configuration, such as a network deployment topology.

When considering the environment for WebSphere Enterprise Service Bus development and functional test, we generally have two options for most projects. We can attempt to simulate the systems driving requests into WebSphere Enterprise Service Bus and build stubbed implementations of the services that will be needed, and, in certain cases, these mechanisms might be necessary. However, the second option is by far the better option and assumes that we can physically connect to the driving systems and required services. This preferred option is more efficient. We do not need to spend time developing test drivers or stubbed services, and we know that our testing is valid because we have used the actual systems and services.

So, our approach is to have the following connectivity for our development environment:

- ▶ Connectivity to WebSphere MQ in order to receive the event that Customer Information Control System (CICS) has written to a queue, as well as to place the event on the queue for WebSphere Business Monitor
- ▶ Connectivity to CICS Web services in order to invoke the services required to enrich the event or reorder stock
- ▶ Connectivity to WebSphere Business Events in order to send the event

In our environment, we do not involve security for the initial development and testing.

5.2 Environment configuration

We now describe the configuration details of the environment for WebSphere Enterprise Service Bus development. We have specific considerations for the following products:

- ▶ CICS: We need to call two Web services provided by CICS.
- ▶ WebSphere Business Events: We need to send events to the Java Message Service (JMS) topic to which WebSphere Business Events has subscribed.
- ▶ WebSphere Business Monitor: We need to consume the CICS event messages from WebSphere MQ and produce messages for consumption by WebSphere Business Monitor and the Audit service.

See Figure 5-1 for a topological overview of our environment.

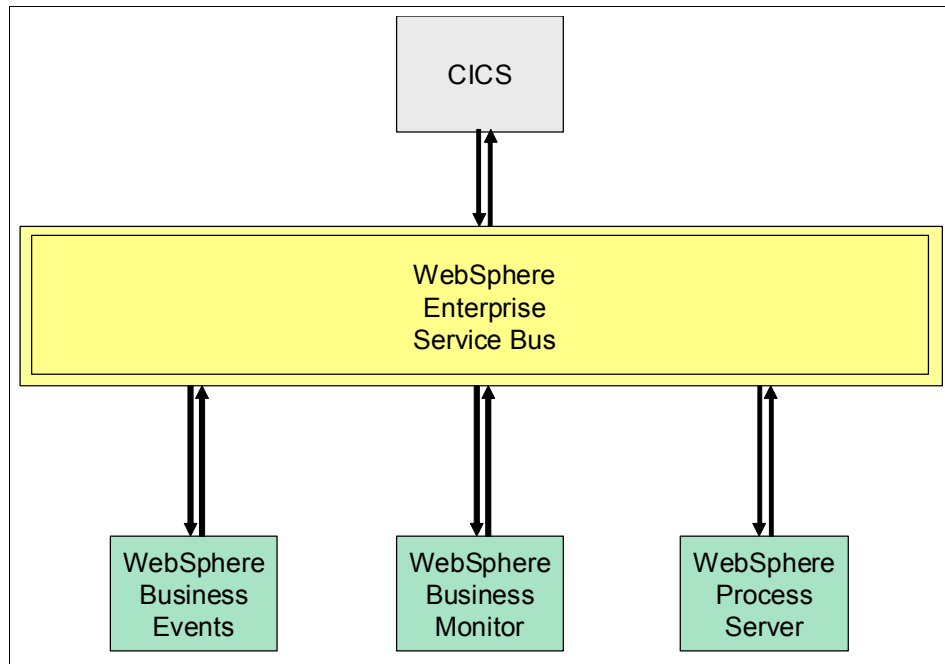


Figure 5-1 Environment topology

5.2.1 CICS configuration

First, we configure CICS to emit events.

Creating a bundle project

To configure CICS to emit events, we first create a new CICS bundle project in the IBM CICS Explorer. The bundle project will contain the event binding (evbind) files and other metadata that will be deployed to CICS.

We follow these steps to create a new bundle project in the CICS Explorer:

1. Click Explorer on the menu bar.
2. Hover the mouse over New Wizards.
3. Click CICS Bundle Project.

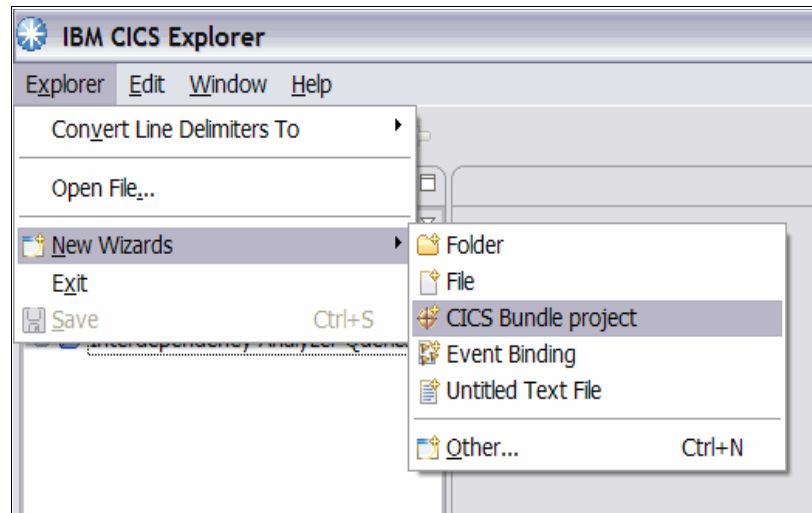


Figure 5-2 Creating a CICS bundle

After we click CICS Bundle Project, we are prompted to name the project. For our project, we named it CatalogManager. After we name our project, it shows under the Project Explorer.

Creating an event binding file

After the project has been created, you can create the event binding file. The event binding is an XML definition that defines one or more business events to CICS. It consists of the event specifications, capture specifications, and event processing (EP) adapter and dispatcher information. We follow these steps to create the event binding file:

1. Right-click the bundle project name.
2. Hover the mouse over New.
3. Click Event Binding.

See Figure 5-3 on page 95.

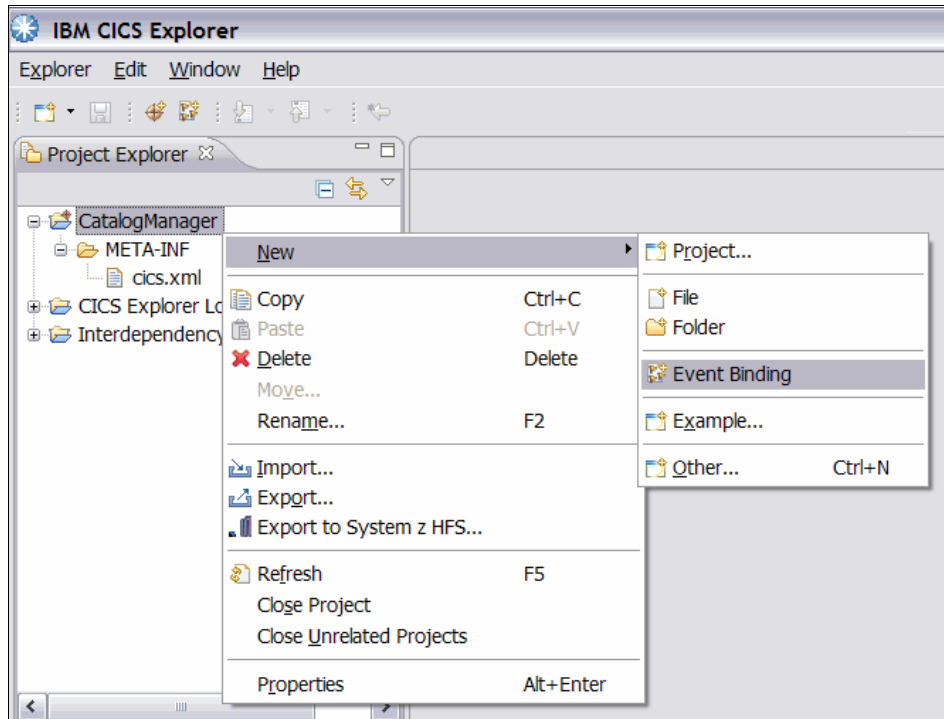


Figure 5-3 Creating an event binding

You will be prompted to enter a name for the event binding file. In our scenario, we named our first event binding file `SuccessfulOrder` and the second event binding file `InsufficientStock`. We chose these names, because we are capturing two events in CICS and want the bind file names to have a name representing each event.

Creating an event specification

After the event binding file is created, the event binding tab editor is displayed. On this window, you can add the event specifications. An *event specification* describes an event and its processing. In the event binding tab, we click Add to create an event specification. See Figure 5-4 on page 96.

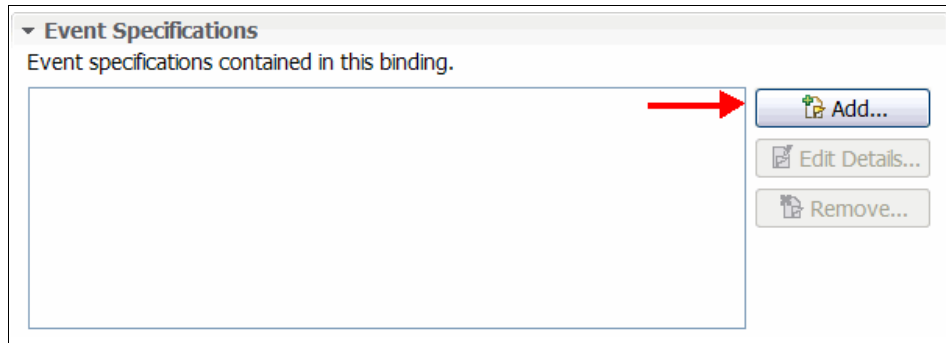


Figure 5-4 Creating an event specification

For our successfulorder event, we created a specification called ItemOrder.

After the specification has been created, we select it in the table, and we click Edit Details, which opens the Specification tab window.

Data to be emitted

We start by defining the information that we want CICS to emit when the event is triggered. For our scenario, we want to emit four pieces of data when the event is triggered. We describe the fields that we want to emit in the DFH0XCP1 copybook. See Example 5-1 on page 100 for the copybook layout. We want to add the following fields to the emitted event:

- ▶ userid (text field)
- ▶ charge_dept (text field)
- ▶ item_ref_number (numeric field)
- ▶ quantity_req (numeric field)

At this time, we do not need to specify a length or precision value. The emitted business information section looks like Figure 5-5 when completed.

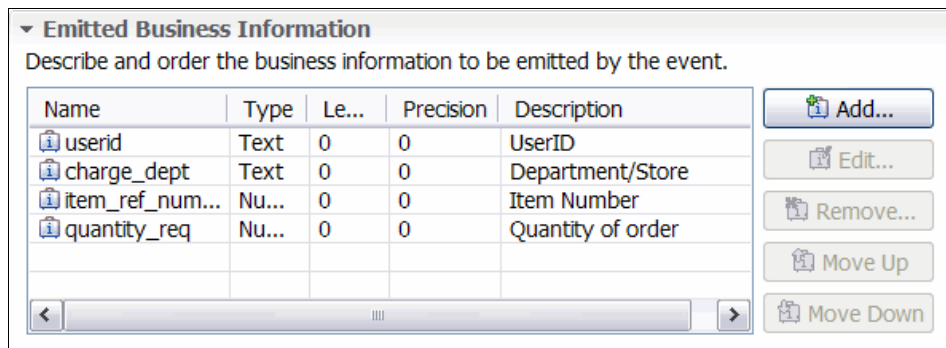


Figure 5-5 Emitted business information

Adding a capture specification

Now that you have defined what you want to emit, you need to set up when you want the event to be triggered. You can define this trigger by adding a capture specification to the event binding.

For this scenario, we named our capture specification `OrderSuccess`, because when an item is ordered successfully, we want to trigger an event. We add a capture specification by clicking “Add a Capture Specification.”

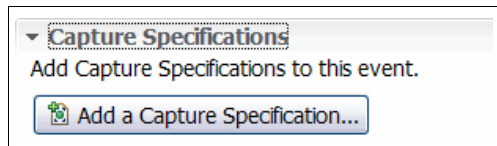


Figure 5-6 Creating a capture specification

When the capture specification is selected, you will see three additional tabs at the top of the editor window when you have the Specifications editor tab open. Using these three additional tabs at the top, you can configure under what conditions you want to trigger the event (Figure 5-7 on page 98).

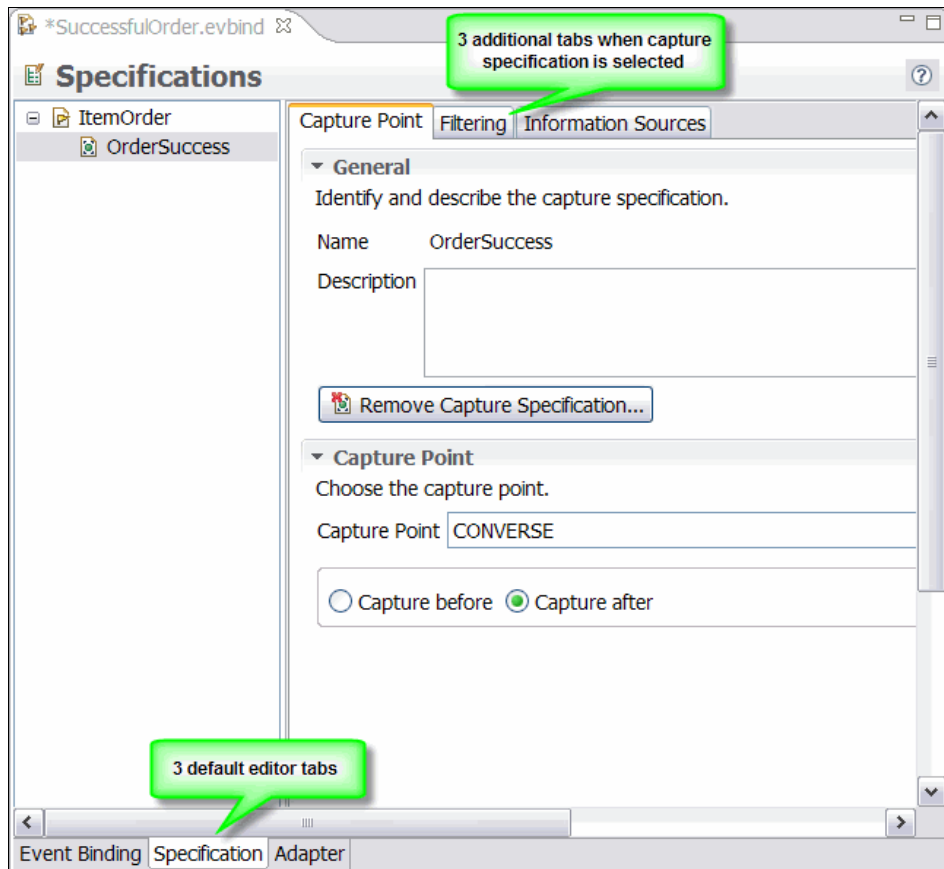


Figure 5-7 Tab layout

When to trigger the event

After creating the capture specification, determine when you want to trigger the event. You can select the EXEC CICS command to use as the capture point by clicking the drop-down menu next to the Capture Point.

In our scenario, we want to trigger the event on the EXEC CICS LINK PROGRAM command.

Note: We used the “Capture after” option, which allows us to filter on the return code.

Creating event filters

After selecting a capture point, we go to the second tab to set up filters. In our scenario, we set the Operator for the response code to Equals. We also set up for an event to be triggered only when the link is to DFH0XVDS. We need to define predicates for application data, because we only want to capture successful orders. Breaking down the COMMAREA, when the first six bytes are 010RDR and the next two bytes are 00, we know that the order was successful. We use this application data as a filter. We click Add. A new window titled Application Data Predicate opens. We fill in the operator and value fields, as shown in Figure 5-8, and we click “Select from imported language structure.”

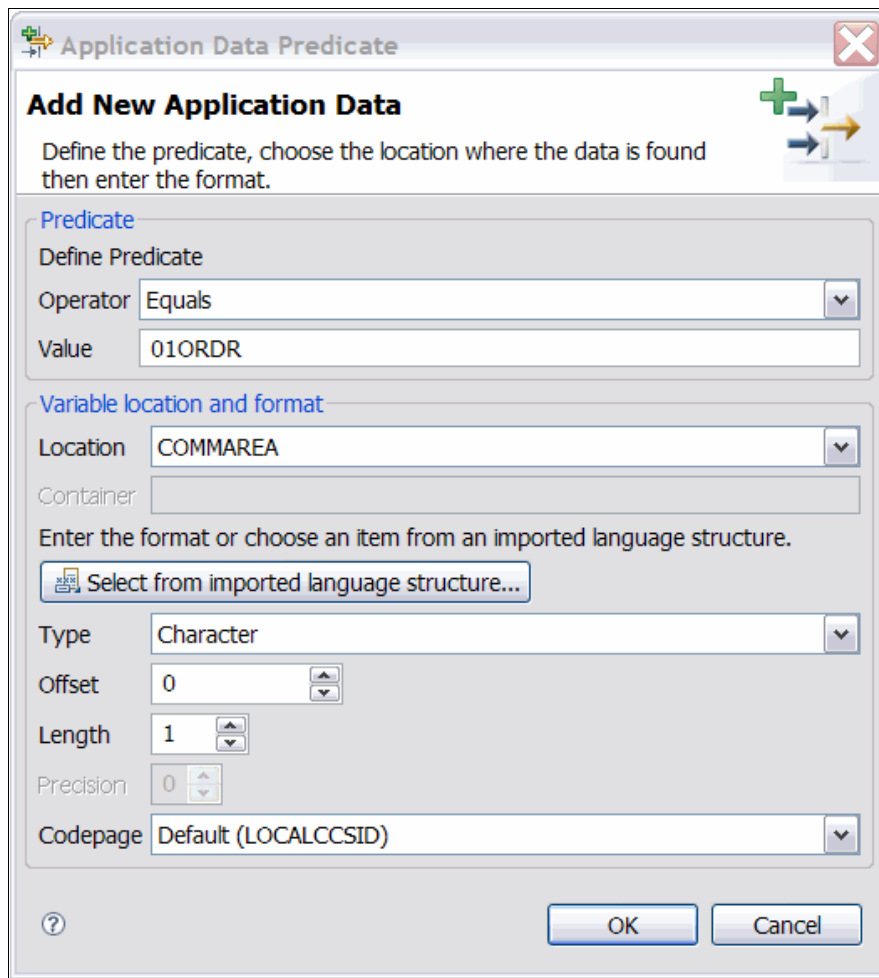


Figure 5-8 Application Data Predicate

We export a COBOL copybook named DFH0XCP1 from the enterprise server to our desktop. This step enables us to use the copybook as input to the CICS Explorer Event Binding editor so that we can map the COMMAREA accurately. Example 5-1 shows the copybook that we import.

Example 5-1 Portion of DFH0XCP1

```

03 CA-REQUEST-ID          PIC X(6) .
03 CA-RETURN-CODE        PIC 9(2) .
03 CA-RESPONSE-MESSAGE   PIC X(79) .
03 CA-ORDER-REQUEST .
    05 CA-USERID          PIC X(8) .
    05 CA-CHARGE-DEPT    PIC X(8) .
    05 CA-ITEM-REF-NUMBER PIC 9(4) .
    05 CA-QUANTITY-REQ   PIC 9(3) .
    05 FILLER             PIC X(888) .

```

After the CICS Explorer parses the copybook, the window shown in Figure 5-9 opens. The first field in which we are interested is `ca_request_id`, because this field must have 010RDR to signify an order. We select the row for `ca_request_id`, and then, click OK. This step returns to the Application Data Predicate window where you notice that the type, offset, and length fields were updated by the CICS Explorer based on the values in the copybook.

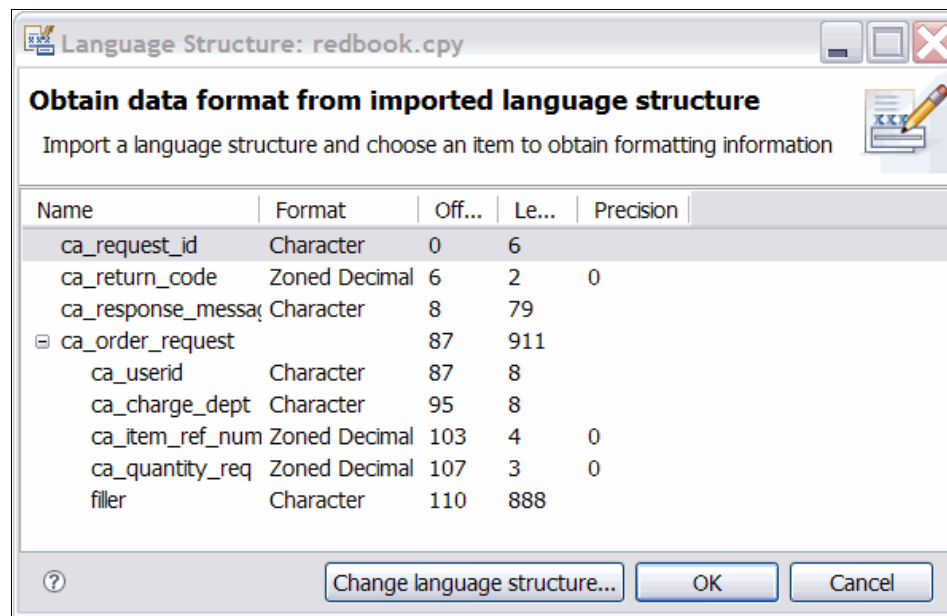


Figure 5-9 Language Structure input

After creating a filter on 01ORDER, we created an additional filter on the `ca_return_code` field in our copybook. We took the same steps as previously described to create a filter for 00. When these two conditions are met, we know that we have a successful order and an event is to be emitted. The Filtering tab looks like Figure 5-10 now.

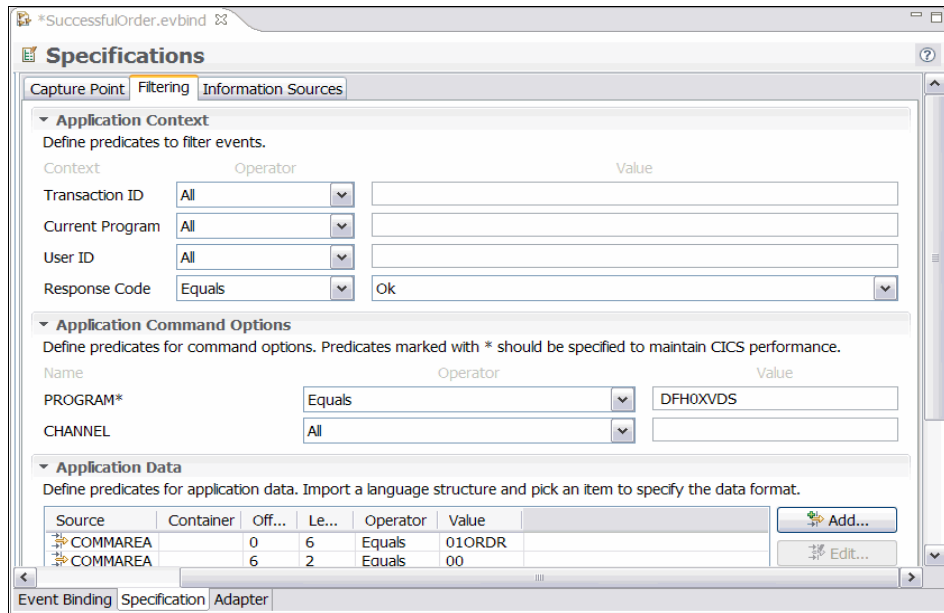


Figure 5-10 Completed Filters view

Now that the filters are set up, the last task in the specification tab is to define from where the emitted business information is obtained. Under the Information Sources tab, we see the four fields that we defined earlier. We can use the copybook that we exported earlier to specify where these fields are located in the COMMAREA. We select `userid`, and we click Edit, which opens a window titled “Information Source for `userid`”.

Because the emitted information is application data, we select `COMMAREA` under the application data tree. On the right panel, we click “Select from imported language structure.” We take the same steps as before when setting a filter on the `ca_request_id`, except that this time, we select the fields that we want emitted during the event. When completed, the information sources look like Figure 5-11 on page 102.

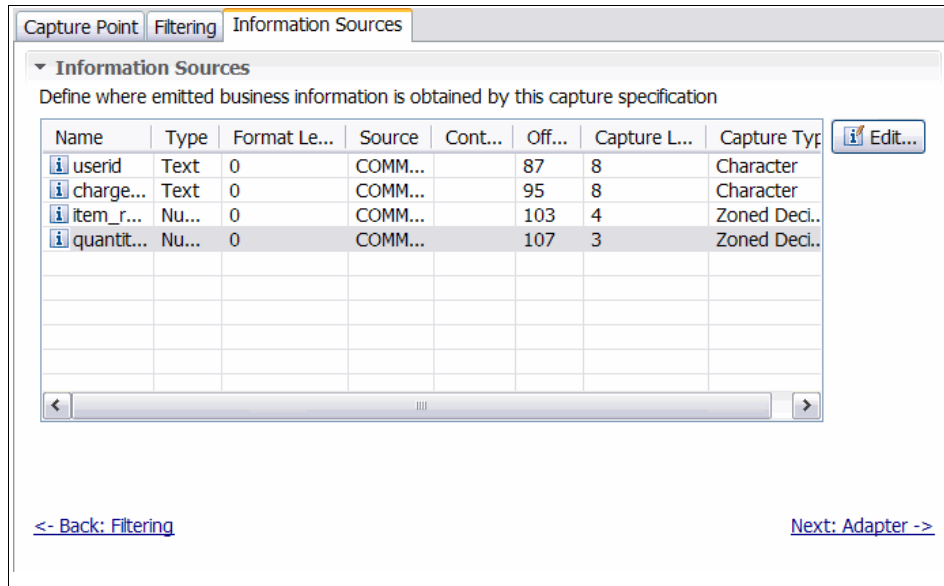


Figure 5-11 Information Sources tab

Event adapter

The last step in creating the evbind file is to choose the EP adapter through which you want the events emitted.

For our scenario, we use a WebSphere MQ adapter that allows the events to be emitted to a queue that is used as input into WebSphere Enterprise Service Bus. In the editor, we select WMQ Queue as the adapter. For the queue name, we specified WESB.CICS.CBE.ORDER. For our scenario, we set the data format to Common Base Event (CBE) (XML).

Exporting schemas

Now that the event binding file is configured to capture the successful order event, we click Export Event Specifications. This step creates an xsd schema file that describes the format of the payload of the event that will be emitted by CICS. During the WebSphere Enterprise Service Bus configuration, we import this schema into the tooling. Because we use a CBE-formatted event, there are two additional schemas that are needed to describe the entire message. The schema generated by the explorer is considered the payload or the dynamic portion of the event. The other portion of the event is the static portion. The static XML schema ships with CICS and is in this location:

```
/usr/lpp/cicsts/cicsts41/schemas/eventprocessing/cics_cbe_static.xsd
```


The last schema needed describes the entire CBE envelope in which the event will be included, which is at this website:

http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf

We use these three schema definitions later as input into the WebSphere Enterprise Service Bus tools.

Insufficient stock event

The only difference between the successful order and insufficient stock event is the return code and the queue to which the event is written. Everything else remains the same, so we can take the same steps previously described to create a second event. There are two changes to make:

1. When setting up the filter predicates, filter on a 97 for `ca_return_code`.
2. Define a separate queue for the event; we called our queue `WESB.CICS.CBE.INSUF.STOCK`.

Deploying the events to CICS

Now that the event binding files have been created, the next step is to deploy these artifacts into CICS. As long as you have connected the CICS Explorer to a host system, you can perform this step within the CICS Explorer.

We right-click the project name under the Project Explorer in the left pane. We select “Export to System zFS” and fill in the destination system detail credentials, as necessary. We determine the location on the System z file system (zFS) where we want to export the bundle.

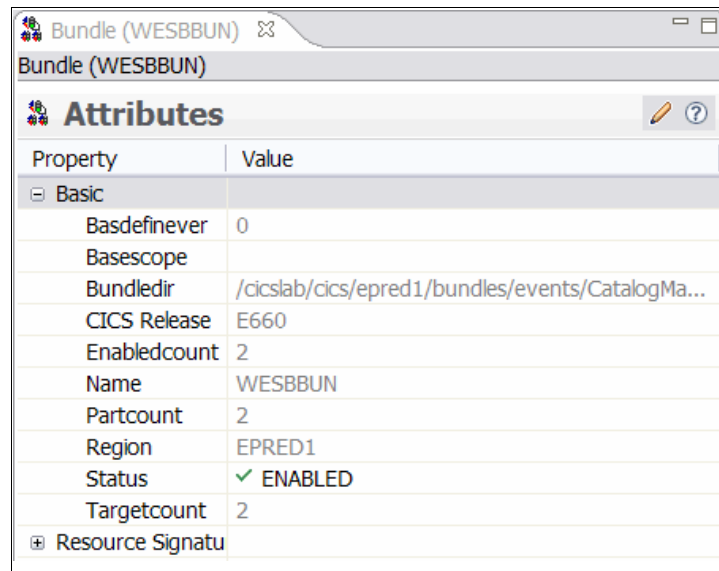
You want to organize the zFS in a way that works for your environment, for example, production versus test, or by region. You probably do not want to use your home (`/u/userid`) directory.

For this scenario, we use the `/cicslab/cics/epred1/bundles/events/` directory. The CICS Explorer appends the name of the project to the directory. In our case, the directory is `/cicslab/cics/epred1/bundles/events/CatalogManager`. When we browse that directory, we see the `META-INF` directory and the two event binding files.

Defining the bundle resource

Now that we have exported the event binding files into zFS, we need to create a bundle definition. If we have a CICS management client interface (CMCI) connection, we can create a bundle definition directly through the explorer. We switch to the CICS Systems Management perspective (which is part of CICS Explorer), select the CICS region where we want to install the bundle, and click

Administration → Bundle Definitions. When we define our bundle definition, we need to specify the bundle directory. In our scenario, the bundle directory is /cicslab/cics/epred1/bundles/events/CatalogManager. If we view the attributes for the new bundle definition that we created, it looks like Figure 5-12.



Property	Value
Basic	
Basdefinever	0
Basescope	
Bundledir	/cicslab/cics/epred1/bundles/events/CatalogMa...
CICS Release	E660
Enabledcount	2
Name	WESBBUN
Partcount	2
Region	EPRED1
Status	✓ ENABLED
Targetcount	2
Resource Signatu	

Figure 5-12 Bundle attributes

After defining the bundle definition, we install it, and then, the events portion of the CICS configuration is complete.

5.2.2 WebSphere Business Events configuration

We used essentially the same configuration of the WebSphere Business Events Server in our WebSphere Enterprise Service Bus development and test environment that was used with the other ESB products. The standard WebSphere Business Events configuration includes the following resources that we used:

- ▶ WbeBus: A service integration bus (SIBus) using the default JMS provider that comes as a standard part of WebSphere Application Server
- ▶ WbeTopicSpace: A destination on the WbeBus that is used for Publish/Subscribe (Pub/Sub) interactions with message producers and consumers
- ▶ jms/WbeTopicConnectionFactory: A JMS resource bound into the Java Naming and Directory Interface (JNDI) namespace allowing connection to the WbeBus and WbeTopicSpace

- ▶ `.jms/eventTopic`: A JMS resource that is also bound into the JNDI namespace where nondurable messages can be published

In addition to these standard resources, we added a foreign bus definition from the WbeBus to the SCA.Application bus in the WebSphere Enterprise Service Bus server. Figure 5-13 is the view of the foreign bus from the WebSphere Business Events server.

Select	Name	Routing type	Description	Configuration status
You can administer the following resources:				
<input type="checkbox"/>	SCA.APPLICATION.qcell.Bus	Direct, service integration bus link		Current
Total 1				

Figure 5-13 Foreign bus in WebSphere Business Events for WebSphere Enterprise Service Bus

The heart of the foreign bus is the bus link that defines the physical connection between the SIBus in WebSphere Business Events and the SIBus in WebSphere Enterprise Service Bus. Figure 5-14 is the view of the active bus link from the WebSphere Business Events server.


Select	Name	Description	Local messaging engine	Foreign messaging engine	Status
You can administer the following resources:					
<input type="checkbox"/>	BusLinkToWESB		kcgl6hkNode01.server1-WbeBus	qnode.server1-SCA.APPLICATION.qcell.Bus	 CWSIP09201: Status OK
Total 1					

Figure 5-14 Bus link between WebSphere Business Events and WebSphere Enterprise Service Bus

This configuration is required for WebSphere Business Events to enable the interaction with the WebSphere Enterprise Service Bus server. See “WebSphere Business Events configuration in WebSphere Enterprise Service Bus” on page 109 for a description of a similar set of resources that must be defined in WebSphere Enterprise Service Bus to complete the configuration.

See 5.7.2, “WebSphere Business Events plug-in for WebSphere Integration Developer” on page 125 for a description of the Jython script that is delivered with the WebSphere Business Events plug-in for WebSphere Integration Developer.

You can use the script for reference if you define these resources manually, or you can execute the script with input parameters appropriate to your environment to automate the configuration task.

5.2.3 WebSphere Business Monitor configuration

The WebSphere Business Monitor configuration is identical for all ESB solutions. We describe it in detail in Chapter 10, “WebSphere Business Monitor” on page 271.

5.2.4 WebSphere Process Server configuration

There is no WebSphere Process Server configuration required for any of the ESB products. We defined scenario 4, which involves WebSphere Process Server, so that WebSphere Business Events makes a Web services call directly to WebSphere Process Server. As a result, there is no ESB function required to complete scenario 4.

5.2.5 WebSphere Enterprise Service Bus configuration

When we installed WebSphere Integration Developer, we had an option to create a profile for WebSphere Enterprise Service Bus, which we chose. The profile provides a stand-alone server topology that has been augmented with the WebSphere Enterprise Service Bus runtime components. Global Security is enabled by default, but we disabled it. We made significant configuration changes to this server, so, in an environment where many WebSphere Enterprise Service Bus projects are being developed, we create a new WebSphere Enterprise Service Bus server profile for this project. We can use a pop-up menu item from the server.

After we start the WebSphere Enterprise Service Bus server, we launch the Administrator Console and begin to add the needed configuration updates.

There are no special steps needed to access Web services from WebSphere Enterprise Service Bus. We have already verified that we can ping the CICS host system. We still have more tasks to perform to access WebSphere MQ and WebSphere Business Events.

WebSphere MQ configuration in WebSphere Enterprise Service Bus

In our environment, WebSphere MQ is running on the z/OS host system, and we have installed, customized, and started the Client Attach Feature. We can

connect to the WebSphere MQ system from our WebSphere Enterprise Service Bus development environment using a Client Attachment.

We will read events from the two WebSphere MQ queues where CICS has placed the events, so we have two inbound queues. We also have two outbound queues to which we will write for WebSphere Business Monitor and the Audit service.

In WebSphere Enterprise Service Bus, therefore, we use Service Component Architecture (SCA) components that have MQ bindings. Our import components (those components that will write the messages) use the JNDI names for the connection factory and the queue. Our export components (those components that receive inbound messages) also use the JNDI names for the queues and need activation specifications to drive the message into WebSphere Enterprise Service Bus.

In order to configure the WebSphere Enterprise Service Bus, we need the information in Table 5-1 about WebSphere MQ.

Table 5-1 WebSphere MQ resources

Resource	Resource name
Queue manager name	MQCR
Queue manager host name	wtsc66.itso.ibm.com
Queue manager host port	1415
Server connection channel	SYSTEM.DEF.SVRCONN
CICS order event queue name	WESB.CICS.CBE.ORDER
CICS failed order event queue name	WESB.CICS.CBE.INSUF.STOCK
WebSphere MQ queue name	WESB.WESB.CBE.ORDER
Audit queue name	WESB.WESB.CBE.LOG

With the information in Table 5-1, we can configure the needed resources in WebSphere Enterprise Service Bus (notice that these resources use the MQ Messaging Provider). Table 5-2 on page 108 shows the export components.

Table 5-2 Export resources

Resource name	Resource type	JNDI name
MQCR	Connection factory	jms/CICS_WBE_CF
CBE_LOG	Queue	jms/CBE_LOG
CBE_ORDER	Queue	jms/CBE_ORDER
CICS_CBE_Failed	Queue	jms/CICS_CBE_Failed
CICS_CBE_ORDER	Queue	jms/CICS_CBE_ORDER
AS_FAILED	Activation specification	jms/AS_FAILED
AS_ORDER	Activation specification	jms/AS_ORDER

You can define these resources at any scope that makes sense in your environment.

We defined them at the node scope. When the configuration definitions are complete, we have resources that match those resources shown in Figure 5-15 and Figure 5-16.

Select	Name	JNDI name	Provider	Description	Scope
You can administer the following resources:					
<input type="checkbox"/>	MQCR	jms/CICS_WBE_CF	WebSphere MQ messaging provider		Node=qnode

Figure 5-15 Connection factory

Select	Name	JNDI name	Provider	Description	Scope
You can administer the following resources:					
<input type="checkbox"/>	CBE_LOG	jms/CBE_LOG	WebSphere MQ messaging provider		Node=qnode
<input type="checkbox"/>	CBE_ORDER	jms/CBE_ORDER	WebSphere MQ messaging provider		Node=qnode
<input type="checkbox"/>	CICS_CBE Failed	jms/CICS_CBE_Failed	WebSphere MQ messaging provider		Node=qnode
<input type="checkbox"/>	CICS_CBE_ORDER	jms/CICS_CBE_ORDER	WebSphere MQ messaging provider		Node=qnode

Figure 5-16 Queues

For the queues in Figure 5-16 on page 108 that are used as input, we need to add the following custom properties to their definitions in WebSphere Enterprise Service Bus:

- ▶ MDWRITE = YES
- ▶ MDREAD = YES
- ▶ MSGBODY = MQ

If you forget to add these custom properties, WebSphere Enterprise Service Bus issues a descriptive error message so that you can resolve the problem easily (Figure 5-17).

Select	Name	JNDI name	Provider	Description	Scope
You can administer the following resources:					
<input type="checkbox"/>	AS_FAILED	jms/AS_FAILED	WebSphere MQ Resource Adapter		Node=qnode
<input type="checkbox"/>	AS_ORDER	jms/AS_ORDER	WebSphere MQ Resource Adapter		Node=qnode

Figure 5-17 Activation specification

WebSphere Business Events configuration in WebSphere Enterprise Service Bus

We use the Default Messaging Provider for the resources that need to interact with WebSphere Business Events. The WebSphere Business Events server uses the JMS Publish/Subscribe (Pub/Sub) capabilities built into WebSphere Application Server V7. It has a topic space and two topics (a durable and a nondurable topic) to which it has subscribed. WebSphere Enterprise Service Bus will need to publish the events to either of those topics.

Because our WebSphere Enterprise Service Bus server is in a separate cell than the WebSphere Business Events server, we need to configure a bus link between the two servers. A *foreign bus* is the abstraction WebSphere Application Server uses to identify the SIBus in the other cell. When we define the foreign bus, we can specify it to support Pub/Sub, as well as point-to-point messaging. In WebSphere Enterprise Service Bus, we create a destination on our SIBus that is a topic space. The foreign bus definition has a topic space map entry that maps the topic space in WebSphere Enterprise Service Bus to the topic space in WebSphere Business Events. We also need to define the two topics in WebSphere Enterprise Service Bus.

If you use the WebSphere Business Events plug-in for WebSphere Enterprise Service Bus (see 5.7.2, “WebSphere Business Events plug-in for WebSphere Integration Developer” on page 125, you notice that it provides a Jython script to automate this configuration. The script is handy, yet it might be best to perform

the configuration work manually the first time so that it is well understood and the values of the parameters that are required by the script are clear. Then, you can configure additional development servers or the WebSphere Enterprise Service Bus servers that will be created for the various stages of the test cycle much more efficiently by using the Jython script supplied by the WebSphere Business Events plug-in.

It is best to start with the configuration definitions needed on the SIBus. In WebSphere Enterprise Service Bus, we have three SIBuses already configured, as shown in Figure 5-18.

Select	Name	Description	Security
You can administer the following resources:			
<input type="checkbox"/>	CEI.qcell.BUS	CommonEventInfrastructure Bus	Enabled
<input type="checkbox"/>	SCA.APPLICATION.qcell.Bus	Messaging bus for Service	Disabled
<input type="checkbox"/>	SCA.SYSTEM.qcell.Bus	Messaging bus for Service	Enabled

Figure 5-18 SIBuses in WebSphere Enterprise Service Bus

We use the SCA_Application bus to perform the needed configuration for WebSphere Business Events.

It is best to start by creating a new destination on the bus, which is a topic space as shown in Figure 5-19.

<input type="checkbox"/>	WBETopicSpace	SCA.APPLICATION.qcell.Bus	Topic space
--------------------------	-------------------------------	---------------------------	-------------

Figure 5-19 Topic space

Next, we can define the foreign bus. Figure 5-20 shows the result.

Select	Name	Routing type	Description	Configuration status
You can administer the following resources:				
<input type="checkbox"/>	WbeBus	Direct, service integration bus link		Current

Figure 5-20 Foreign bus

If you have configured the foreign bus correctly, you will see the bus link show with a Started status, as shown in Figure 5-21 on page 111.

Select	Name	Description	Local messaging engine	Foreign messaging engine	Status	Current outbound messages	Messages sent	Current inbound messages	Messages received
You can administer the following resources:									
<input type="checkbox"/>	BusLinkToWESB		qnode.server1-SCA.APPLICATION.qcell.Bus	kcg16hkNode01.server1-WbeBus	CWSIP0920I: Status OK	0	3	0	3

Figure 5-21 Bus link in Started state

Notice that the buses have exchanged messages. These messages indicate, along with the messages in the server logs, that the Pub/Sub topology has been successfully updated between the two servers.

When defining the foreign bus, make sure that you have selected to include support for Pub/Sub and that you were given an opportunity to create the topic space mapping, as shown in Figure 5-22.

Select	Local topic space	Remote topic space
You can administer the following resources:		
<input type="checkbox"/>	WBETopicSpace	WbeTopicSpace
Total 1		

Figure 5-22 Topic space mapping

We have completed the configuration needed based on the SIBus topology. Now, we need to configure the JMS resources. We need to have a topic connection factory, as shown in Figure 5-23.

<input type="checkbox"/>	WbeTopicConnectionFactory	jms/WbeTopicConnectionFactory
--------------------------	---	-------------------------------

Figure 5-23 Topic connection factory

Finally, we need to define the topics for durable and non-durable subscriptions, as shown in Figure 5-24.

<input type="checkbox"/>	durableEventTopic	jms/durableEventTopic	Default messaging provider
<input type="checkbox"/>	eventTopic	jms/eventTopic	Default messaging

Figure 5-24 Topics

We have completed the configuration of the WebSphere Enterprise Service Bus server for development. We can now build the mediation modules required for our solution.

5.3 Scenario 1

Virtually all WebSphere Integration Developer development begins by creating a library project. *Libraries* are where we want to keep the XML Schema Definitions (xsd) and Web Services Description Language (wsdl) that are required for the solution. Therefore, we can share the definitions among multiple projects. For WebSphere Enterprise Service Bus, any project that is not a library will be a mediation module. Our solution for both scenarios includes the projects illustrated in Figure 5-25.

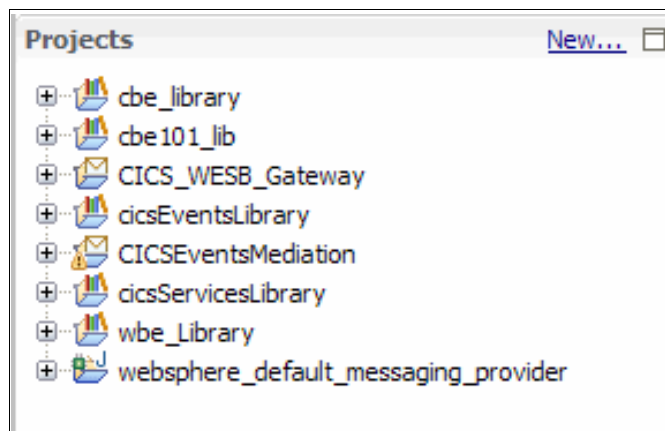


Figure 5-25 WebSphere Integration Developer projects in WebSphere Enterprise Service Bus solution

We describe the content of each library and its origin:

- ▶ **cbe_library:** We imported the `cbe101.xsd` schema into this library. It is the same as `cbe101_lib` except that the namespace was changed, which we will explain later.
- ▶ **cbe101_lib:** We imported the `cbe101.xsd` schema and left the namespace in its original form. We actually do not use this library at this time.
- ▶ **cicsEventsLibrary:** We imported the `cics_cbe_static.xsd`, which defines the static portion of the CBE event from CICS. We also imported the `ItemOrder.xsd` and `FailedOrder.xsd` into this library, which define the payload carried by the static portion of the cbe.
- ▶ **cicsServicesLibrary:** We imported the `inquireSingle.wsdl` and the `restockItem.wsdl` into this library. These files contain the definitions that are required to interact with the CICS Web services.
- ▶ **wbe_Library:** Here we imported both `Event_FailedOrder-esb.xsd` and `Event_ItemOrder-esb.xsd`, which define the contents of the event sent to

WebSphere Business Events in each of the two cases. We also defined an interface for each of the two event types in this library.

We examine the mediation modules next.

5.3.1 ESB transformation

As we began development of our mediation module, we first want to be certain that WebSphere Enterprise Service Bus was able to successfully consume the CBE event coming from CICS. We found this action to be a problem, and our implementation had to change somewhat to accommodate the unexpected behavior.

CBE uses the `http://www.ibm.com/AC/commonbaseevent1_0_1` namespace, which is special in WebSphere Application Server. Because the logging framework in WebSphere Application Server uses this namespace, a static Eclipse Modeling Framework (EMF) model was developed for it. WebSphere Enterprise Service Bus uses dynamic models for the schema that it consumes. Dynamic models in WebSphere Enterprise Service Bus allow us to convert easily between XML and the Service Data Objects (SDO) framework. Unfortunately, after WebSphere Application Server loads the static model, WebSphere Application Server will not attempt to use the dynamic model. The effect of the limitation is that WebSphere Enterprise Service Bus cannot use SDOs to process any XML using the CBE namespace. Our workaround addresses that problem by splitting the mediation function into two modules.

For now, we have the following solution diagram in WebSphere Integration Developer (Figure 5-26).

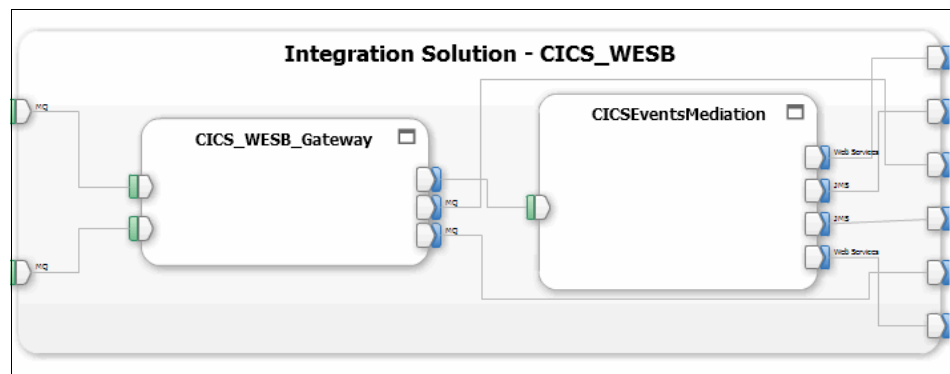


Figure 5-26 WebSphere Enterprise Service Bus solution diagram

All processing of events with the CBE namespace is performed in the first module, which essentially implements the routing requirements. The second module is responsible for additional routing, enrichment, and transformation. This approach is a reasonable division of the required functions. We might have arrived at this implementation without the limitation that forced us to adopt it.

CICS_WESB_Gateway

This module processes all incoming CBE events from CICS but without attempting to parse the XML. We do not create business objects (BOs) in WebSphere Enterprise Service Bus from the incoming events. This module also places the event on the WebSphere Business Monitor and audit queues with the original CBE namespace. In the assembly editor, the module looks like Figure 5-27.

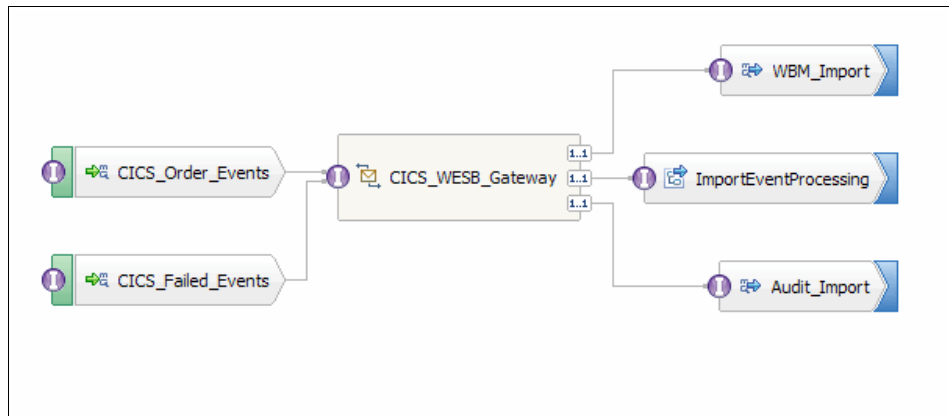


Figure 5-27 CICS_WESB_Gateway assembly

We initially generated the module using the Static Service Gateway pattern dialog in WebSphere Integration Developer. We remove the Request/Response operation and make several other modifications for the final implementation, yet the pattern was a helpful starting point. The essential aspect of developing this module is its use of the predefined resources, as shown in Figure 5-28 on page 115.

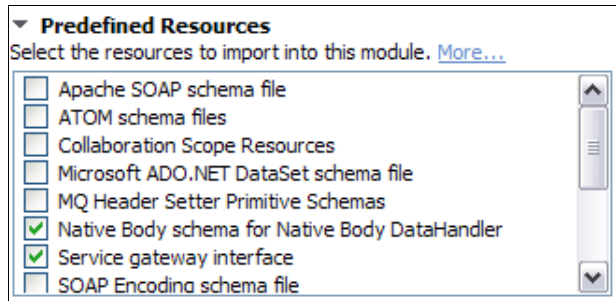


Figure 5-28 Predefined Resources used by CICS_WESB_Gateway

The Native Body schema and the service gateway interface allow us to process the CBE event without actually creating business objects from the elements in the message. Looking at the SCA components individually, we can better understand what this module contributes to our solution.

CICS_Order_Events

This SCA export has an MQ binding and uses the service gateway interface. The component receives the incoming ItemOrder event from the queue where CICS has written it.

CICS_Failed_Events

This SCA export also uses an MQ binding and the same service gateway interface. The component receives the incoming FailedOrder event from WebSphere MQ.

WBM_Import

This SCA import uses an MQ binding and the ServiceGateway interface to send the CBE event unmodified to the WebSphere Business Monitor queue.

Audit_Import

This SCA import also uses an MQ binding and the ServiceGateway interface, which allows us to send the CBE event unmodified to the audit queue.

ImportEventProcessing

This SCA import has an SCA binding to an export in the CICSEventsMediation module. It uses the cicsEvent interface. Before sending the event, we have changed the namespace.

CICS_WESB_Gateway

The SCA mediation flow component is responsible for routing and a small amount of transformation. We look inside the component to understand its function in more detail (Figure 5-29).

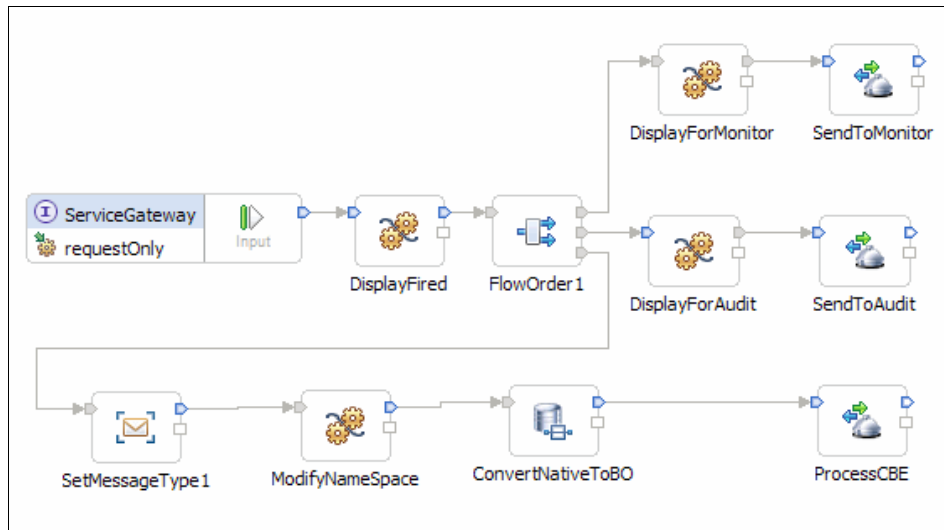


Figure 5-29 CICS_WESB_Gateway mediation flow component

The mediation flow component has four custom primitives. The first three custom primitives with names beginning with “Display” simply print a message to the system log. The custom primitive named `ModifyNameSpace` modifies the namespace. It looks at the message text for the string “`http://www.ibm.com/AC/commonbaseevent1_0_1`” and, if it finds the string, replaces it with “`http://www.ibm.com/AC/commonbaseevent1_0_A`”. This change allows us to circumvent the limitation discussed earlier and to send the message across an SCA interface to the `CICSEventsMediation` module.

The remaining primitives in the flow are fairly self-evident. The `FlowOrder` primitive allows us to fire each of its output terminals in order. The `SetMessageType` primitive essentially recasts the `xsd:any` that is the message body to a `TextBody` type for the downstream primitives to process. The three Service Invoke primitives call the reference partners that we saw in the assembly (the three SCA imports).

CICSEventsMediation

This module processes the CBE with the slightly altered namespace, so that we can perform the necessary processing and send the WebSphere Business Events event. Because the WebSphere Business Events event does not use the

CBE namespace, we have no problems with our dynamic model and can easily consume the XML, creating the BOs as needed. Figure 5-30 shows the structure of the module in the assembly editor.

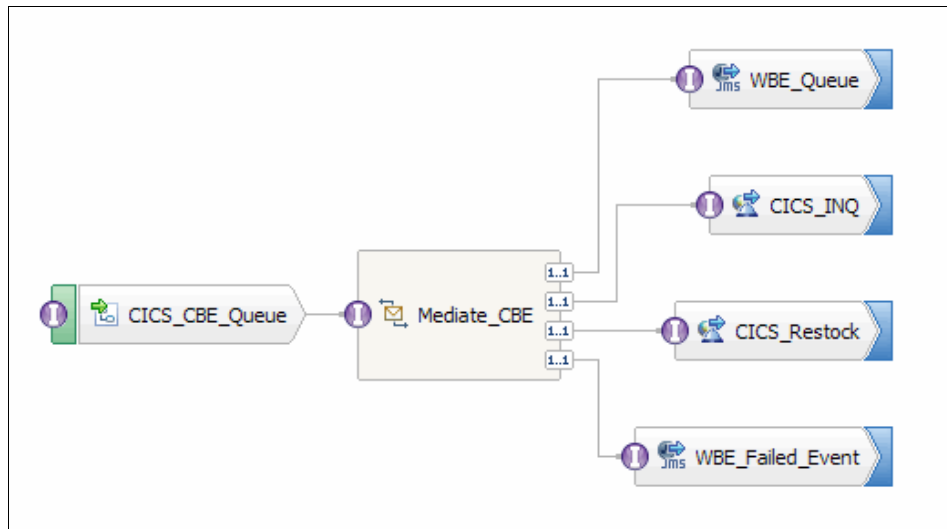


Figure 5-30 CICSEventsMediation assembly

To understand the module fully, we start by looking at each of the SCA components in the assembly.

CICS_CBE_Queue

This SCA export uses an SCA binding and the `cicsEvent` interface, which allows the `CICS_WESB_Gateway` to access the module.

WBE_Queue

This SCA import uses a JMS binding and the Pub/Sub protocols to send a WebSphere Business Events event to a non-durable topic (`.jms/eventTopic`). We needed to create the interface, `WBE_Ops`, after we imported the schema for the WebSphere Business Events event.

WBE_Failed_Event

This SCA import also uses a JMS binding configured for Pub/Sub. It publishes the event to the same topic, yet it uses the `WBE_Failed_Ops` interface, which we also created.

CICS_INQ

This SCA import has a Web Service binding and uses the `DFH0XCMNPort` interface, which was created when we imported the `inquireSingle.wsdl`.

CICS_Restock

This SCA import also has a Web Service binding and uses the CATREODRPort INterface, which was created when we imported the restockItem.wsdl.

Mediate_CBE

This SCA component is the mediation flow component that implements the needed processing. We explain the implementation of the component (Figure 5-31) now.

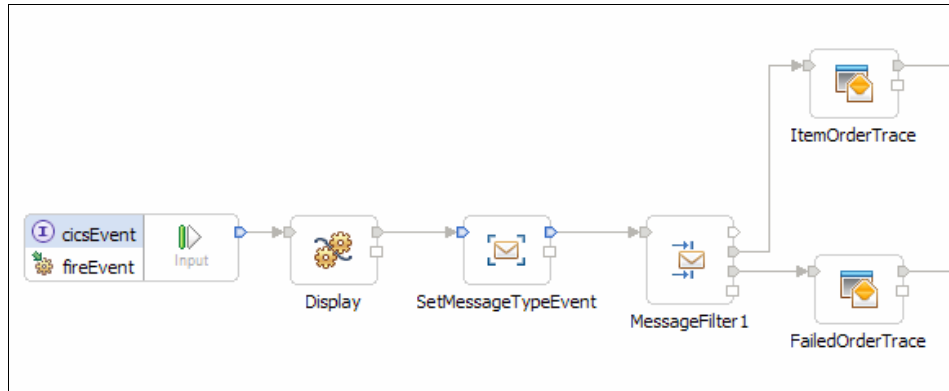


Figure 5-31 Mediate_CBE flow component: Part 1

We look at the flow in three parts and focus on the ItemOrder path. After the Display custom primitive, we need to set the abstract (xsd:any) element, which is the static portion of the cbe, to its actual type, which, in our case, is an event BO. This event BO was defined when we imported the cics_cbe_static.xsd into the cicsEventsLibrary.

The MessageFilter primitive looks at the context-info/eventname element and routes accordingly.

We then use a Trace primitive to log which path through the flow has been taken (Figure 5-32 on page 119).

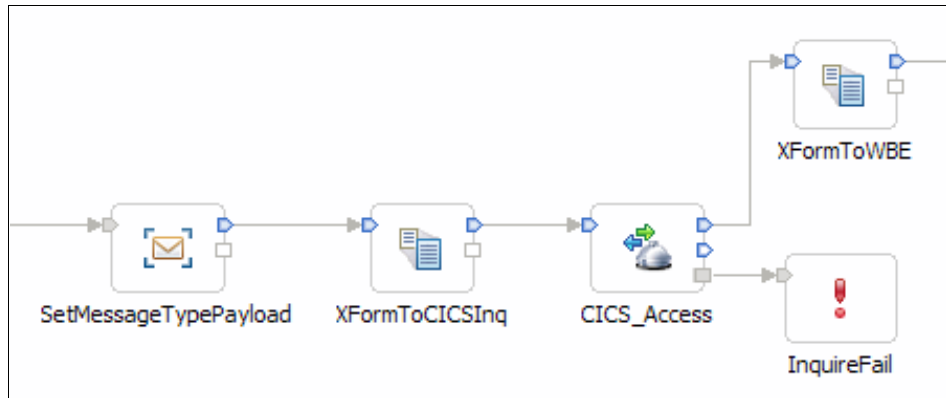


Figure 5-32 Mediate_CBE flow component: Part 2 (ItemOrder)

Now that we have an ItemOrder event, we must set the actual type of the payload-data element. We set the actual type in the SetMessageTypePayload primitive.

The Extensible Stylesheet Language Transformation (XSLT) primitive that is next in the flow actually has two important functions. It transforms the body of the service message object (SMO) to the format that is required to call the InquireSingle CICS Web Service. Yet, it also must store the elements of the original message that we will need to construct the WebSphere Business Events event. We have created a private BO in this module that we are using as the Transient Context object. So, the XFormToCICSInq also sets the needed fields into the Transient Context.

We then use a Service Invoke primitive to call the partner reference, and if the Web Service is successful, we will flow to the XFormToWBE XSLT primitive. Here, we store the item's unit cost into the Transient Context and construct the WebSphere Business Events event from the fields that are already stored there (Figure 5-33).

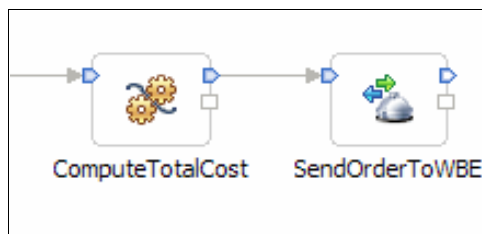
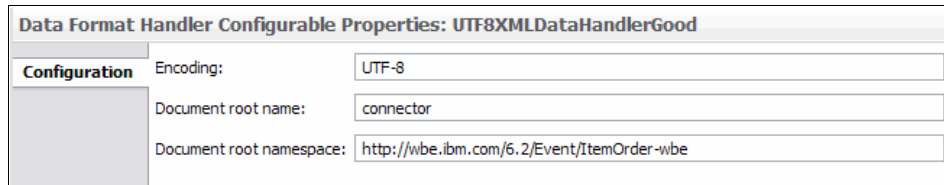


Figure 5-33 Mediate_CBE flow component: Part 3 (ItemOrder)

The final part of the flow consists of computing the total cost and sending the event. The computation is done using Java in a custom primitive. We then use a Service Invoke to call our partner reference, which causes the event to be published to the eventTopic in WebSphere Business Events. Notice that we set the Document root name and the Document root namespace properties in our DataHandler (Figure 5-34), which allows us to suppress the operation element from the XML passed to WebSphere Business Events.



Data Format Handler Configurable Properties: UTF8XMLDataHandlerGood	
Configuration	Encoding: UTF-8
	Document root name: connector
	Document root namespace: http://wbe.ibm.com/6.2/Event/ItemOrder-wbe

Figure 5-34 Data Handler properties

Testing in WebSphere Integration Developer

Testing in WebSphere Integration Developer is facilitated by the use of the Integrated Test Client. When working with messaging systems, the client can be launched in Attach mode. Figure 5-35 is the Session contents from a successful ItemOrder CBE from CICS as it flows through the CICS_WESB_Gateway module.

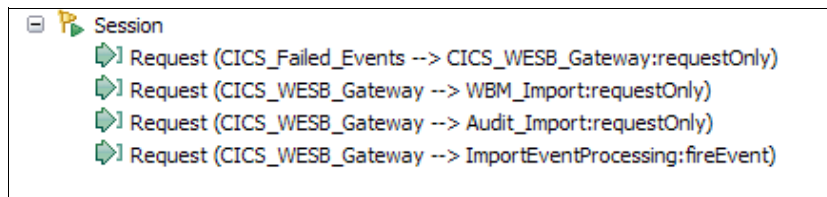


Figure 5-35 Integrated Test Client trace of CICS_WESB_Gateway module

We have not included the details, but you can examine each of the requests in the Integrated Test Client and display the contents.

Figure 5-36 on page 121 shows a similar trace as the message flows through our second mediation module. The Fine-Grained Trace shows each primitive that was traversed, and we can display the details of the SMO after each primitive has been executed.

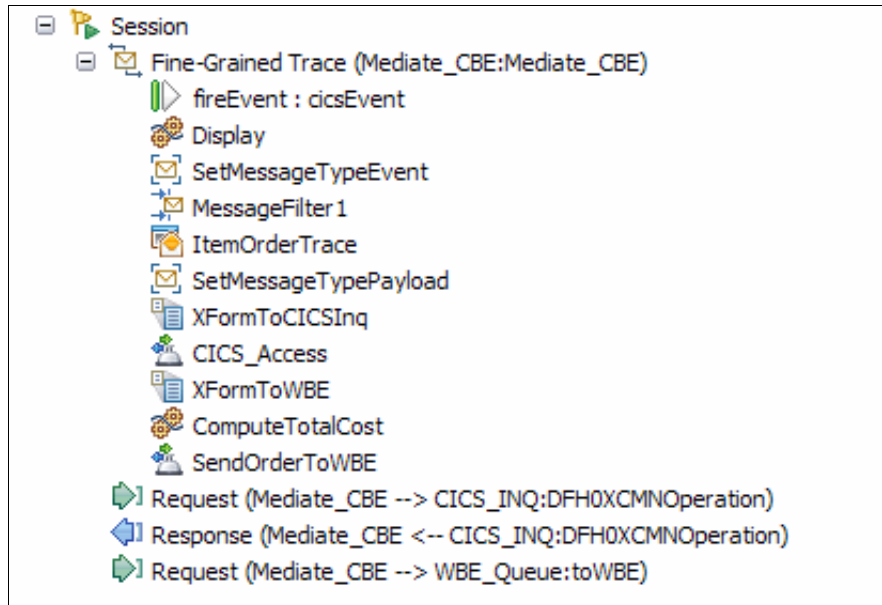


Figure 5-36 Integrated Test Client trace of CICSEventsMediation module

5.3.2 Test results

From the perspective of CICS, WebSphere Business Monitor, and WebSphere Business Events, there are no differences in the test results due to the specific ESB product in the solution. Chapter 10, “WebSphere Business Monitor” on page 271 describes the test results, where we have treated the ESB layer generically.

5.4 Scenario 2

The implementation of scenario 2 does not require any ESB function.

5.5 Scenario 3

We have already completed many of the activities that we need to do for scenario 3. This scenario uses many of the same libraries and mediation modules that are used in scenario 1. Our CICS_WESB_Gateway module does not need to change, because it does not rely on the type of event that is sent from CICS. We

need to look at the mediation flow component in the CICSEventsMediation module and add the functions that are required for the FailedOrder event.

5.5.1 ESB transformation

From our discussion in scenario 1, we have a good understanding of what the mediation flow component does for the ItemOrder event. Now, we work with the FailedOrder event.

The first part of the flow made a determination about which kind of event was received from CICS. Now, we focus on the primitives in the flow after the FailedOrderTrace primitive (Figure 5-37).

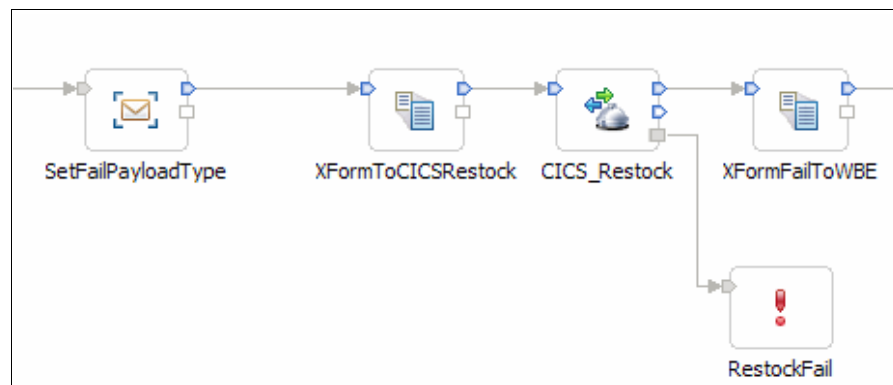


Figure 5-37 Mediate_CBE flow component: Part 2 (FailedOrder)

The SetFailPayloadType primitive must set the actual type of the xsd:any element representing the payload-data element.

Next, we use an XSLT primitive, named XFormToCICSRestock, to transform the SMO body to the request message required by the Restock service. Again, this primitive is performing double duty, because it also stores the fields from the incoming event BO into the Transient Context object in the SMO. We will use these fields later in the flow to construct the WebSphere Business Events event.

The Service Invoke primitive is used to call the partner reference, which represents the CICS Restock Web Service. If the Web Service fails, we raise an exception and terminate the flow.

Assuming that the Web Service call was successful, the XSLT named XFormFailToWBE constructs a new SMO body according to the WebSphere Business Events event schema definition. We use the fields that are stored in the Transient Context object to set the contents of the event.

Figure 5-38 shows the final primitive in the flow.



Figure 5-38 Mediate_CBE flow component: Part 3 (FailedOrder)

This Service Invoke causes the WebSphere Business Events event to be published to the eventTopic in WebSphere Business Events, which completes the processing.

Testing in WebSphere Integration Developer

The Integrated Test Client, running in Attach mode, is helpful during our testing. Figure 5-39 shows the trace of the CICSEventsMediation as a FailedOrder event is processed by WebSphere Enterprise Service Bus.

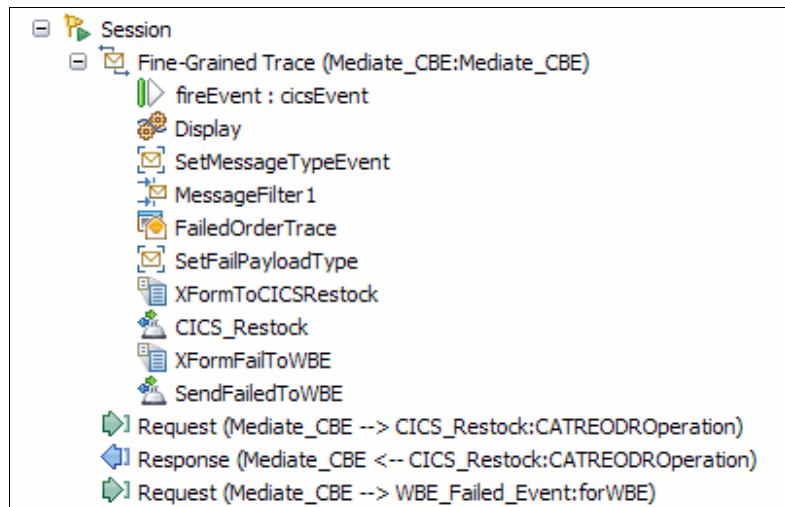


Figure 5-39 Integrated Test Client trace of the CICSEventsMediation module

5.5.2 Test results

From the perspective of CICS, WebSphere Business Monitor, and WebSphere Business Events, there are no differences in the test results due to the specific ESB product in the solution. See Chapter 10, “WebSphere Business Monitor” on

page 271 for a description of the test results, where we have treated the ESB layer generically.

5.6 Scenario 4

The implementation of scenario 4 does not require any ESB function.

5.7 Problems encountered, hints, and tips

We made a number of discoveries during the development and testing of the solution for WebSphere Enterprise Service Bus. We did have one problem that we discuss next that dealt with cross-cell Pub/Sub. The remaining sections offer suggestions for alternative ways of implementing the solution and tips for the developer.

5.7.1 Cross-cell Pub/Sub in WebSphere Application Server V7

We configured a foreign bus in each of the WebSphere Enterprise Service Bus and WebSphere Business Events servers. With an active bus link and the appropriate topic spaces and topics defined, the idea was to publish the event in WebSphere Enterprise Service Bus to the local eventTopic and to let the foreign bus configuration, along with the topic space mapping configured in WebSphere Enterprise Service Bus, be responsible for forwarding the published message to subscribers in the WebSphere Business Events server.

Unfortunately, and although the configuration was correct and operational, the event published in WebSphere Enterprise Service Bus never arrived at subscriber destinations in WebSphere Business Events.

To work around this problem, we made a simple configuration change. In WebSphere Enterprise Service Bus, instead of defining the eventTopic to use the SCA.Application Bus, we pointed the eventTopic directly at the bus in WebSphere Business Events (WbeBus). We also had to specify the name of the topic space in WebSphere Business Events, rather than the name in WebSphere Enterprise Service Bus.

As a result of making this single configuration change, we were able to publish events from WebSphere Enterprise Service Bus to WebSphere Business Events.

It is likely that the topic space mapping between the two cells was not functioning properly, and it is also likely that by the time that this book is published, that problem will be resolved.

5.7.2 WebSphere Business Events plug-in for WebSphere Integration Developer

WebSphere Business Events supplies runtime and tooling support for WebSphere Integration Developer and WebSphere Enterprise Service Bus that is helpful if you work with WebSphere Business Events events frequently. We did not originally use this support to develop our mediations, primarily because we thought it was important to understand the details of interacting with WebSphere Business Events from WebSphere Enterprise Service Bus.

The instructions for installing the plug-in in WebSphere Integration Developer and then updating the WebSphere Enterprise Service Bus run time with the provided jar file are located at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/index.jsp?topic=/com.ibm.wbe.integrating.doc/doc/installingsibx.html>

In WebSphere Integration Developer, the plug-in provides three new mediation primitives that simplify interactions with WebSphere Business Events. In our scenarios, the WebSphere Business Events EventEmitter primitive is useful.

Figure 5-40 is the final part of the ItemOrder flow in the CICSEventsMediation module that has been rewired to use the primitive.

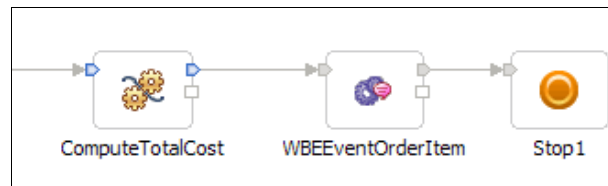


Figure 5-40 Using the WebSphere Business Events EventEmitter primitive in the ItemOrder flow

Similarly, we can rewire the FailedOrder portion of the flow to use another WebSphere Business Events EventEmitter primitive, as shown in Figure 5-41 on page 126.

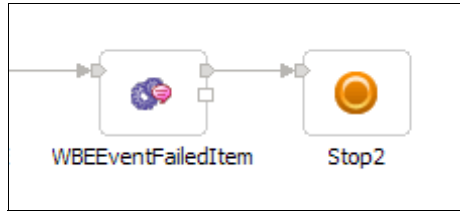


Figure 5-41 Using the WebSphere Business Events EventEmitter primitive in the FailedOrder flow

The primitive provides property details that you can use to customize its behavior. For our flows, we point the primitive to the correct portion of the SMO body element to pass to WebSphere Business Events, as shown in Figure 5-42.

Event properties	JMS related properties
Root:*	/body/toWBE/connector
Transaction mode:	Existing
Durability:	Non durable

Figure 5-42 WebSphere Business Events EventEmitter primitive event properties

The JMS-related properties allow us to define the topic factory and topics to use when sending the event to WebSphere Business Events, as illustrated in Figure 5-43.

Event properties	JMS related properties
Topic factory:	jms/WbeTopicConnectionFactory
Topic for non-durable events:	jms/eventTopic
Topic for durable events:	jms/durableEventTopic

Figure 5-43 WebSphere Business Events EventEmitter primitive JMS properties

When you build the Assembly, you do not need to wire the references to import components, because the WebSphere Business Events EventEmitter is configured to include the information otherwise specified on the import. You need a reference in the mediation flow component that has the operation used to send the event.

The effect in our scenarios is an assembly diagram with two warnings that did not exist before, because we have not wired the references (Figure 5-44).

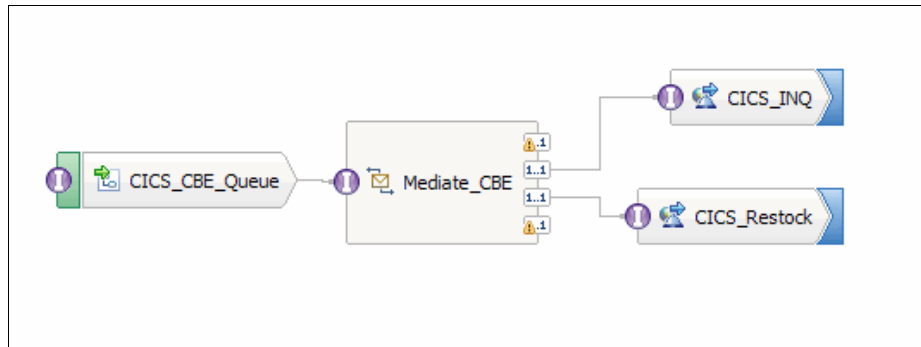


Figure 5-44 *CICSEventsMediation* assembly using WebSphere Business Events tooling enhancements

If you use these primitives in WebSphere Integration Developer, you must add the jar file included in the package to your WebSphere Enterprise Service Bus server.

You must complete the JMS configuration that we did in WebSphere Enterprise Service Bus and WebSphere Business Events, regardless of your choice to use the WebSphere Business Events plug-in in WebSphere Integration Developer. However, the package provides well-documented Jython scripts that make the required work significantly easier.

We used the scripts as reference when we made the manual updates.

Decide if the enhancements have value in your environment. For the `EventEmitter` primitive, the incremental value might be small. However, if you send or receive actions from WebSphere Business Events, the primitives can simplify your development effort.

5.7.3 Modifications to WebSphere Business Events event schema

We discussed early in this chapter the various libraries that we created and the XML schema definitions that we imported. We needed to make a small modification to the xsds that were imported into the `wbe_Library` in order for the events that we generated to be consumed properly by WebSphere Business Events.

As imported, the schema causes the creation of a connector BO. The connector is an element and not a type, however; so, in the imported form, WebSphere

Enterprise Service Bus automatically generates an `xsi:type` definition in the XML that is sent to WebSphere Business Events. This `xsi:type` definition appears to cause WebSphere Business Events problems consuming the XML, so we changed the top-level element in both schemas to a `complexType`, and WebSphere Business Events readily accepted the messages.

5.7.4 When WebSphere Business Monitor and WebSphere Enterprise Service Bus are in the same cell

In many cases, a topology with a single WebSphere Application Server cell is preferred as a means to ease administration. It is likely that the cell that contains WebSphere Enterprise Service Bus also contains WebSphere Business Monitor, although commonly these products are in separate clusters within the cell.

In such cases, after the event arrives in WebSphere Enterprise Service Bus from WebSphere MQ, it makes little sense for WebSphere Enterprise Service Bus to write the event back to a WebSphere MQ queue for WebSphere Business Monitor to retrieve it. A more likely approach is for WebSphere Enterprise Service Bus to write the message to the destination on the Common Event Infrastructure (CEI) bus where it otherwise arrives from WebSphere MQ. In our example, you can use a JMS binding on an SCA import and point to the JMS queue that was defined during the WebSphere Business Monitor configuration for this purpose. This JMS queue is the queue that is being mediated by the handler that was installed (MQtoCEIMediation EAR file). Be sure to write this message as a JMSText message.

5.7.5 CBE details in the WebSphere Application Server run time

During our development effort, we took time to more fully understand how the WebSphere Application Server foundation deals with CBEs. There is a bit of a collision between WebSphere Application Server (a CBE is generally an instance of `org.eclipse.hyades.logging.events.cbe.CommonBaseEvent`) and WebSphere Enterprise Service Bus, which treats the CBE coming from CICS as a business object using the SDO framework. Our solution illustrates a simple way to manage the limitation. We think the article at the following link is helpful to build a better understanding of the CBE processing in WebSphere Application Server:

<http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/autonomic/books/cbeppractice/index.htm>

5.7.6 Deploying to the next test stage

After the solution is verified to function properly in the WebSphere Integration Developer test environment, you are ready to deploy it to the next stage in the test cycle. You export the two projects as EAR files and deploy them using the Administrator Console for WebSphere Enterprise Service Bus or through an automated script. Be aware of the following two points:

- ▶ **Web services endpoints:** The CICSEventsMediation application has fixed references to the two CICS Web services endpoints. Often, you use a separate CICS region in a more advanced test stage, so you might have to change these endpoints. You can use a Jython script to change these endpoints easily by using the Jython script as a part of your automated deployment processing. Or, you can use the Administrator Console to update the endpoint, as shown in Figure 5-45.

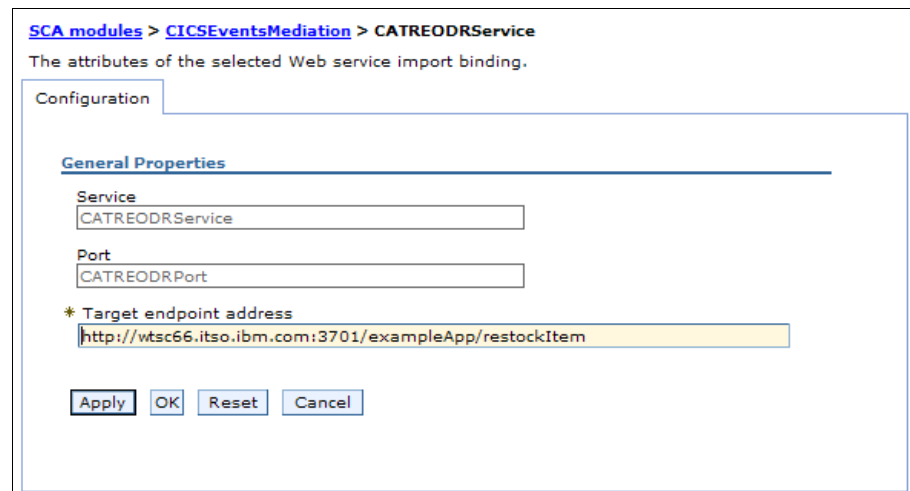


Figure 5-45 Verify target endpoint address

- ▶ **WebSphere MQ Resource JNDI names:** All of our SCA components used JNDI names for the resources that they accessed. This approach allows for much easier deployment to an environment where the resource names have changed. When we select to use JNDI names in our components, WebSphere Integration Developer generates the associated resource reference in the deployment descriptor for the module. As a result, you only need to map the JNDI name used in the module to the actual JNDI name of the resource during deployment. You can perform this mapping manually from the Administrator Console or automatically with scripts.

5.8 Summary

In this chapter, we looked at the detail of the WebSphere Enterprise Service Bus implementation of scenarios 1 and 3. We discussed the configuration of the development environment, including CICS, WebSphere MQ, WebSphere Business Events, and WebSphere Enterprise Service Bus. We examined the two mediation modules that together perform the needed functions for the solution. We looked at the mediation flow components and how they route, transform, and enrich the event payload for downstream consumers.

If we plan to continue to build our solution, we consider security and the transactional behavior of the services involved in the flow, as well as the requirements for persisting the events.

If you are interested in the fine-grained details of the WebSphere Enterprise Service Bus implementation, download the project interchange file that is available with the source materials for this book and import the project into WebSphere Integration Developer. A PDF file, which was generated by WebSphere Integration Developer and that fully documents the implementation, is available.



WebSphere Message Broker business scenario

In this chapter, we perform the following tasks:

- ▶ Present an overview of the environment
- ▶ Set up the environment:
 - CICS setup
 - WebSphere Message Broker setup
 - WebSphere Business Events setup
 - WebSphere Business Monitor setup
 - WebSphere Process Server setup
- ▶ Show how each scenario works and how each product in our environment handles the event

6.1 Environment overview

The environment that we use in this scenario is IBM Customer Information Control System Transaction Server (CICS TS) V4.1 with WebSphere MQ 7.0.1 on z/OS, WebSphere Message Broker 7.0 with WebSphere MQ V7.0 on Microsoft Windows XP, WebSphere Business Events V7.0 running on Microsoft Windows XP, and WebSphere Business Monitor V7.0 running on Microsoft Windows XP.

Figure 6-1 shows the topology of our environment.

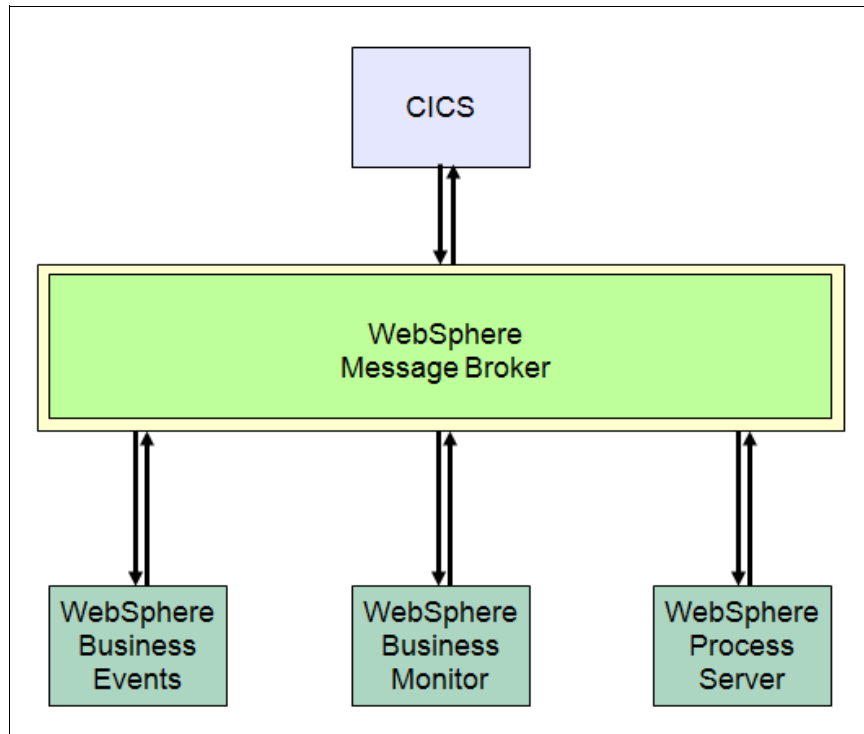


Figure 6-1 Topology

6.2 Configuring the environment

In the following sections, we discuss the configuration for these products:

- ▶ CICS
- ▶ WebSphere Message Broker

- ▶ WebSphere Business Events
- ▶ WebSphere Business Monitor
- ▶ WebSphere Process Server

6.2.1 Configuring CICS

Next, we step through the CICS configuration.

Creating a bundle project

To configure CICS to emit events, the first task to take is to create a new bundle project in the CICS Explorer. The bundle project will contain the evbind files and other metadata that will be deployed into CICS.

We perform these steps to create a new bundle project in the CICS Explorer (Figure 6-2):

1. Click Explorer on the menu bar.
2. Hover the mouse over New Wizards.
3. Click CICS Bundle project.

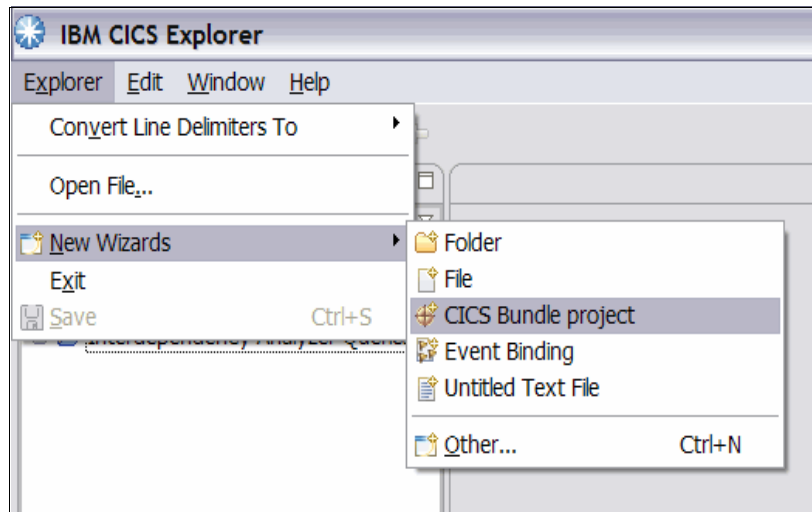


Figure 6-2 Creating a CICS bundle

After clicking CICS Bundle Project, we are prompted to name the project. We name our project CatalogManager. After we enter the project name, the project shows under the Project Explorer.

Creating the event binding file

After the project has been created, we can now create the event binding file. The event binding is an XML definition that defines one or more business events to CICS. It consists of the event specifications, capture specifications, and event processing (EP) adapter and dispatcher information. We follow these steps to create the event binding file:

1. Right-click the bundle project name.
2. Hover the mouse over New.
3. Click Event Binding.

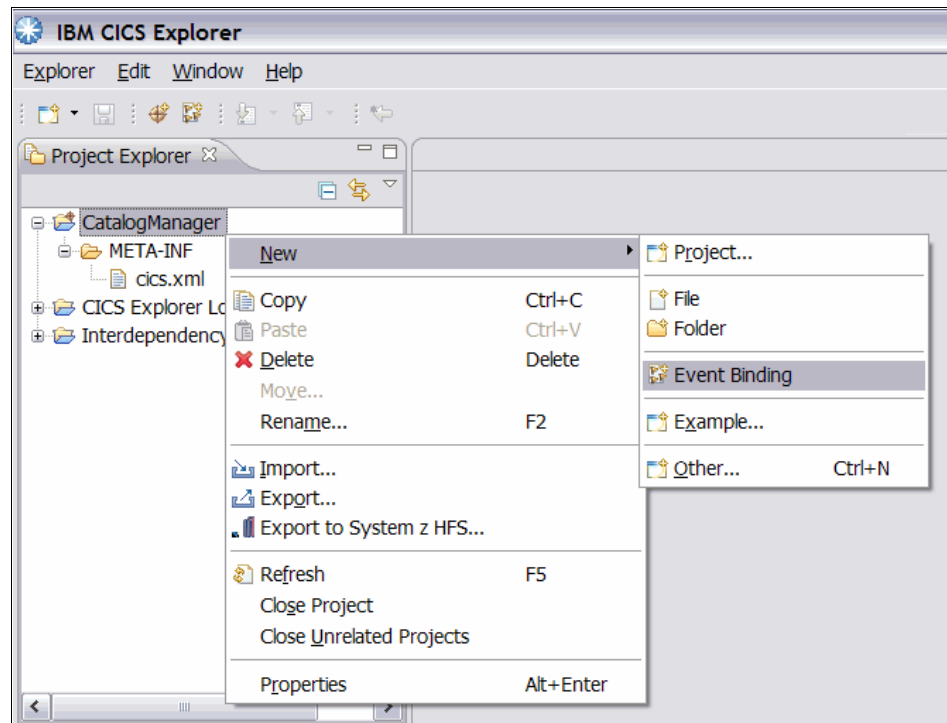


Figure 6-3 Creating event binding

We are prompted to enter a name for the event binding file. In our scenario, we named our first event binding file `SuccessfulOrder` and the second event binding file `InsufficientStock`. We chose these names, because we are capturing two events in CICS and want the bind file name to represent each event.

Creating the event specification

After the event bind is created, the Event Binding tab editor is displayed. On this tab, we can add the event specifications. An *event specification* describes an

event and its processing. On the Event Binding tab, we click Add to create an event specification.

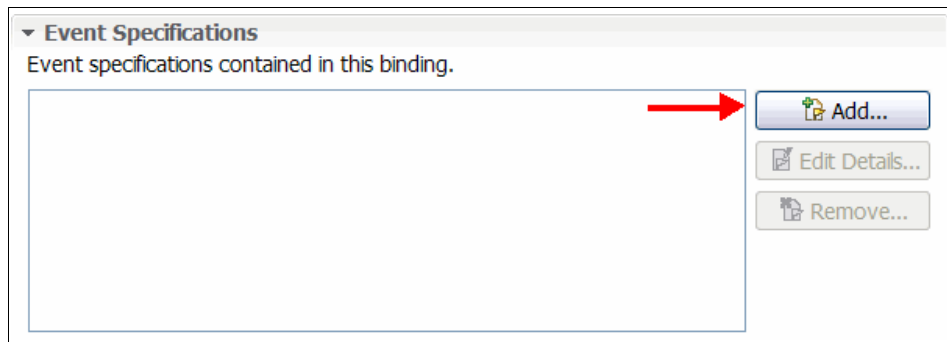


Figure 6-4 Creating an event specification

For our successfulorder event, we create a specification called ItemOrder.

After the specification has been created, we select it in the table, and we click Edit Details. This option causes the Specification tab to open.

Data to be emitted

We start by defining the information that we want CICS to emit when the event is triggered. For our scenario, we want to emit four pieces of data when the event is triggered. The fields that we want to emit are described in the copybook DFH0XCP1. See Example 6-1 on page 139 for the copybook layout. We want to add the following fields to the emitted event:

- ▶ userid (text field)
- ▶ charge_dept (text field)
- ▶ item_ref_number (numeric field)
- ▶ quantity_req (numeric field)

At this time, we do not need to specify a length or a precision value. The emitted business information section looks like Figure 6-5 on page 136 when completed.

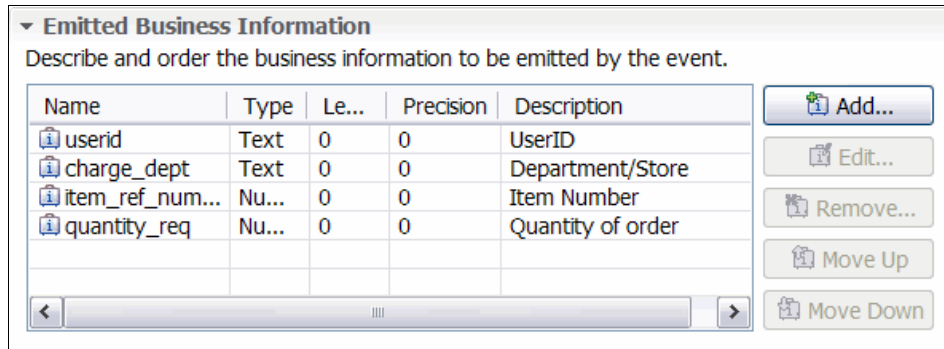


Figure 6-5 *Emitted Business Information*

Adding a capture specification

Now that you have defined what you want to emit, you need to set up when you want the event to be triggered. You can define when you want the event to be triggered by adding a capture specification to the event binding.

For this scenario, we named our capture specification `OrderSuccess`, because we want to trigger an event when an item is ordered successfully.

You can add a capture specification by clicking “Add a Capture Specification” (Figure 6-6).



Figure 6-6 *Creating a capture specification*

When the capture specification is selected, you see three additional tabs at the top of the editor window when you have the Specification editor tab open. Using these three tabs at the top, you can configure the conditions under which you want the event triggered (Figure 6-7 on page 137).

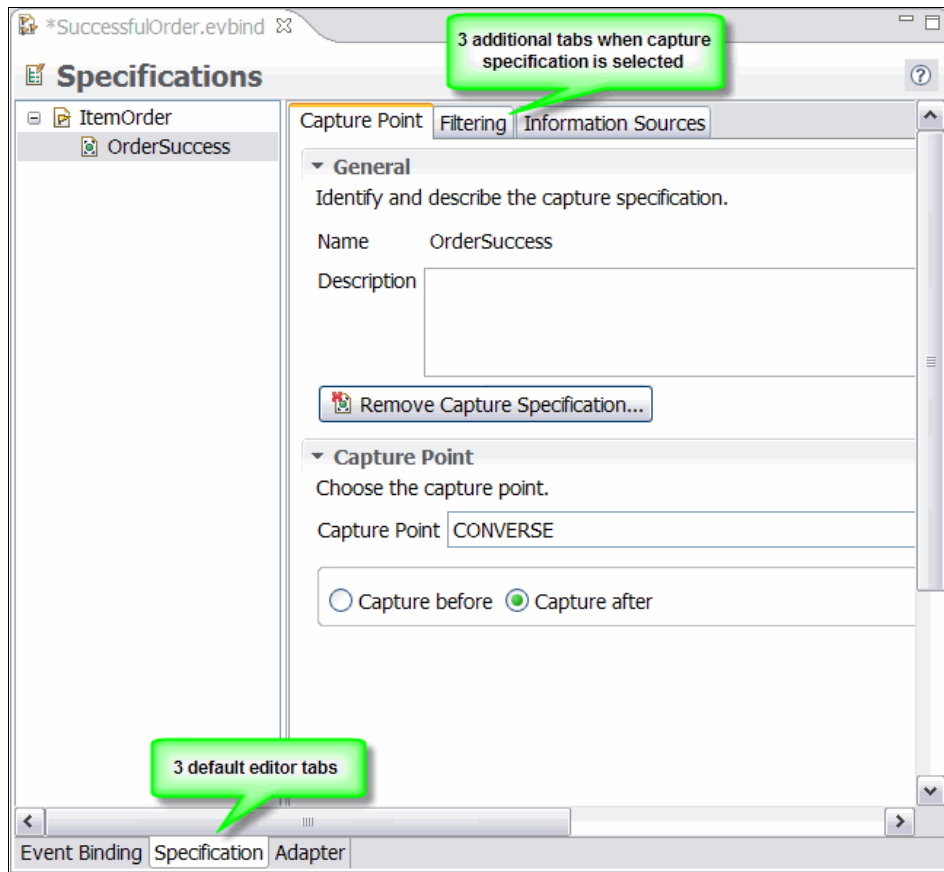


Figure 6-7 Tab layout

When to trigger the event

After creating the capture specification, consider when you want to trigger the event. You can select the EXEC CICS command as the capture point by clicking the drop-down menu next to Capture Point.

In our scenario, we want to trigger the event on the EXEC CICS LINK PROGRAM command.

Tip: We use the Capture after option, which allows us to filter on the return code.

Creating event filters

After selecting a capture point, we select the second tab to set up filters. In our scenario, we set the Operator for Response Code to Equals. We also set an event to be triggered only when the link is to DFH0XVDS. Last, we define predicates for the application data, because we want to capture successful orders only. Breaking down the COMMAREA, we know that the order was successful when the first six bytes are 010RDR and the next two bytes are 00. We use this application data as a filter by clicking Add. A new window titled Application Data Predicate opens. We fill in the operator and value fields, as shown in Figure 6-8, and then, click the “Select from imported language structure”.

Application Data Predicate

Add New Application Data

Define the predicate, choose the location where the data is found then enter the format.

Predicate

Define Predicate

Operator: Equals

Value: 010RDR

Variable location and format

Location: COMMAREA

Container:

Enter the format or choose an item from an imported language structure.

Select from imported language structure...

Type: Character

Offset: 0

Length: 1

Precision: 0

Codepage: Default (LOCALCCSID)

OK Cancel

Figure 6-8 Application Data Predicate

We export a COBOL copybook DFH0XCP1 from the enterprise server to our desktop to use as input into the CICS Explorer Event Binding editor so that we can map the COMMAREA accurately. Example 6-1 shows the copybook that we imported.

Example 6-1 Portion of DFH0XCP1

```

03 CA-REQUEST-ID          PIC X(6) .
03 CA-RETURN-CODE        PIC 9(2) .
03 CA-RESPONSE-MESSAGE   PIC X(79) .
03 CA-ORDER-REQUEST .
    05 CA-USERID          PIC X(8) .
    05 CA-CHARGE-DEPT    PIC X(8) .
    05 CA-ITEM-REF-NUMBER PIC 9(4) .
    05 CA-QUANTITY-REQ   PIC 9(3) .
    05 FILLER            PIC X(888) .

```

After the CICS Explorer parses the copybook, the window that is shown in Figure 6-9 opens. The first field that we are interested in is `ca_request_id`, because this field must have 01ORDD to signify an order. We select the row for `ca_request_id`, and then click OK, which returns us to the Application Data Predicate window. Notice that the type, offset, and length fields have been updated by the CICS Explorer based on the values in the copybook.

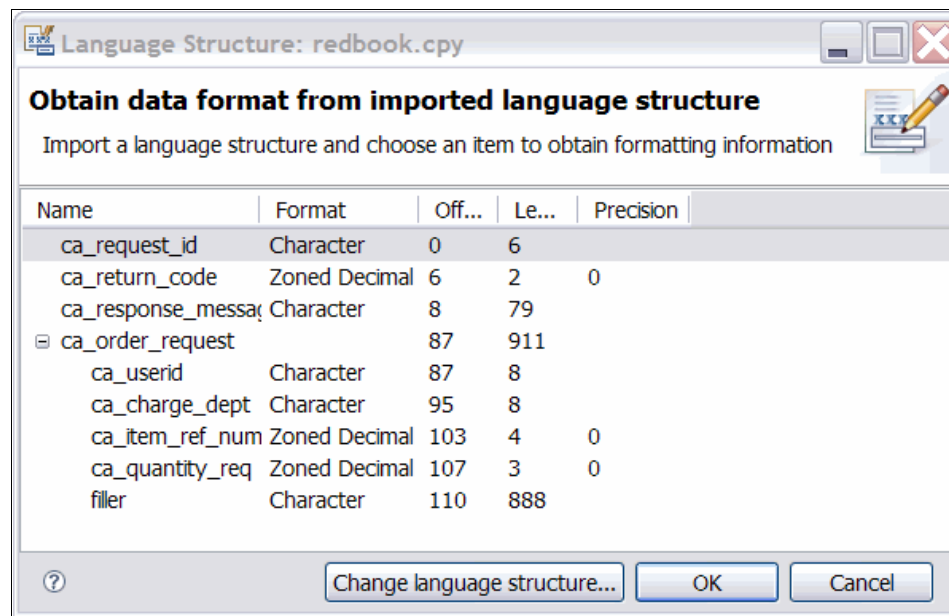


Figure 6-9 Language Structure input

After creating a filter on 01ORDER, we create an additional filter on the ca_return_code field in our copybook. We use the same steps that we used previously and create a filter for 00. When these two conditions are met, we know that a successful order has occurred and an event is emitted. Figure 6-10 shows the Filtering tab.

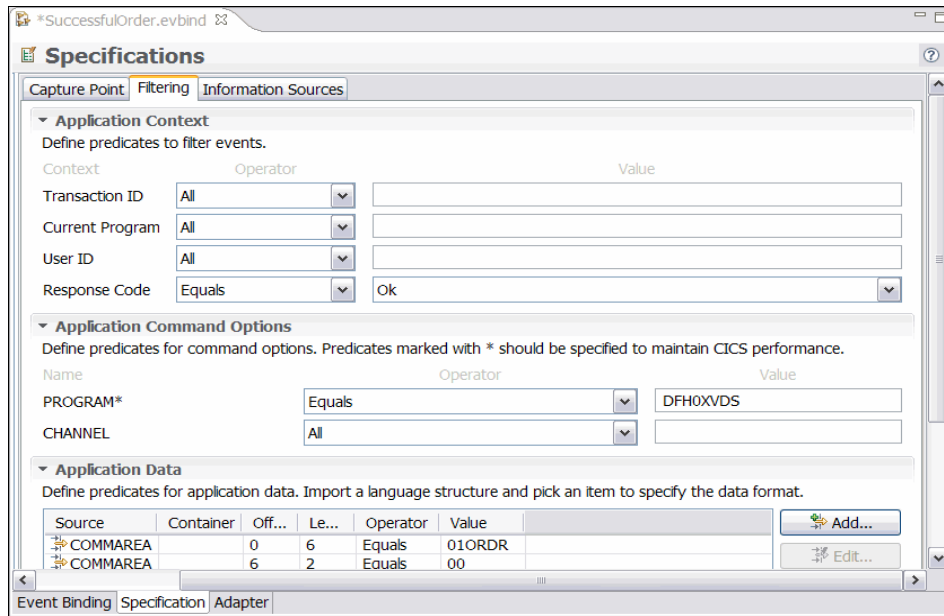


Figure 6-10 Specifications Filtering tab

Now that the filters are set up, the last task in the Specification tab is to define from where the emitted business information is obtained. The Information Sources tab shows the four fields that were defined earlier. We use the copybook that we exported earlier to specify where these fields are located in the COMMAREA. We select userid, and we click Edit, which causes the “Information Source for userid” window to open. Because the emitted information is application data, we select COMMAREA under the application data tree. On the right panel, we now click “Select from imported language structure”. We perform the same steps as before when setting a filter on the ca_request_id; only this time, we select the fields that we want emitted during the event. When completed, the information sources look like Figure 6-11 on page 141.

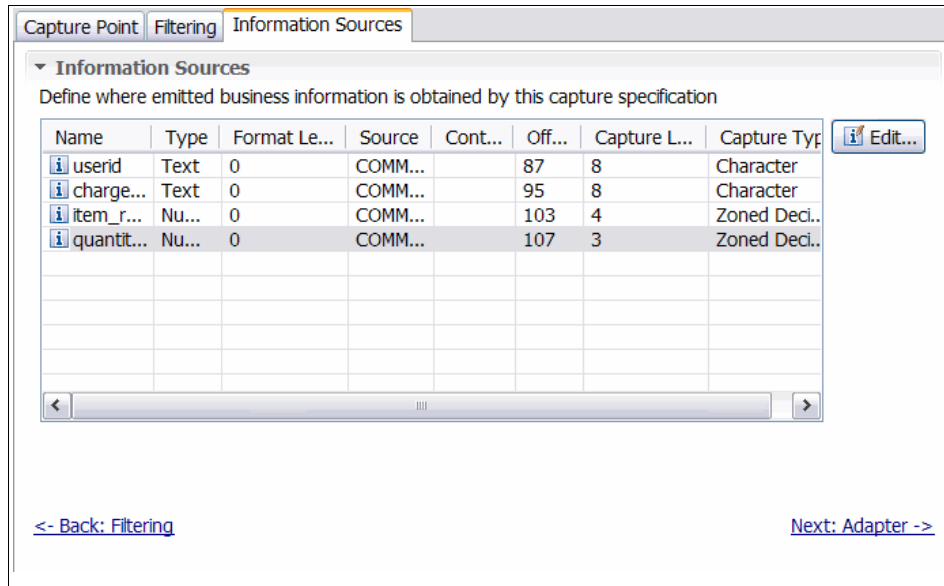


Figure 6-11 Information Sources tab

Event adapter

The last step in creating the evbind file is to choose the EP adapter through which you want the events to be emitted. For our scenario, we use a WebSphere MQ Queue adapter that allows the events to be emitted to a queue, which will be used as input into WebSphere Message Broker. In the editor, we select WebSphere MQ Queue as the adapter. For the queue name, we specify WMB.CICS.CBE.ORDER. For our scenario, we set the data format to Common Base Event (CBE) (XML).

Exporting schemas

Now that the event binding file is configured to capture the successful order event, we click “Export Event Specifications”. This action creates an xsd schema file that describes the format of the payload of the event that will be emitted by CICS. During the WebSphere Message Broker configuration, we import this schema into the tooling. Because we use a CBE-formatted event, we need to describe two additional schemas to describe the entire message. The schema generated by the explorer is considered the *payload* or dynamic portion of the event. The other portion of the event is the *static* portion. The static XML schema ships with CICS at this location:

```
/usr/lpp/cicsts/cicsts41/schemas/eventprocessing/cics_cbe_static.xsd
```

The last schema needed describes the entire CBE envelope within which the event will be included. You can obtain this schema at this website:

http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf

We use these three schema definitions later as input into the WebSphere Message Broker tools.

Insufficient stock event

The only differences between the successful order event and insufficient stock event are the return code and the queue to which the event is written. Everything else remains the same, so you can take the same steps just described to create a second event. There are two changes that you need to make:

- ▶ When setting up the filter predicates, filter on a 97 for `ca_return_code`.
- ▶ Define a separate queue for the event; we called our queue `WMB.CICS.CBE.INSUF.STOCK`.

Deploying the events to CICS

Now that the event binding files have been created, the next step is to deploy these artifacts into CICS. As long as you have connected the CICS Explorer to a host system, you can perform this step within the Explorer. Right-click the project name under the Project Explorer in the left pane. Select Export to system z HFS. Fill in the Destination system detail credentials, as necessary. Determine the location on zFS where you want to export the bundle. You want to organize it in a way that works for your environment, for example, production versus test, or by region. You probably do not want to use your home (`/u/userid`) directory.

For this scenario, we use the `/cicslab/cics/epred3/bundles/events/` directory. CICS Explorer appends the name of the project to the directory. We use the `/cicslab/cics/epred3/bundles/events/CatalogManager` directory. When we browse that directory, we see the META-INF directory and the two event binding files.

Defining the bundle resource

Now that we have exported the event binding files into the hierarchical file system (HFS), we need to create a bundle definition. If you have a CICS management client interface (CMCI) connection, you can create a bundle definition directly through the explorer. We switch to the CICS Systems Management perspective (which is part of CICS Explorer), select the CICS region where we want to install the bundle, and click Administration → Bundle Definitions. When you define your bundle definition, you need to specify the bundle directory. Our bundle directory is `/cicslab/cics/epred3/bundles/events/CatalogManager`. Figure 6-12 on page 143 shows the attributes for the new bundle definition.

Property	Value
Basic	
Basescape	
Bundle Director	/cicslab/cics/epred3/bundles/events/CatalogMana...
CSDGroup	CATMGR
Description	Message Broker Bundle
Name	WMBBUN
Status	✓ ENABLED
Version	0
Definition Signature	
Change Agent	CSDAPI
Change Release	0660
Change Time	Mar 8, 2010 11:16:40 AM
Change User ID	CICRSR1
Create Time	Mar 8, 2010 11:16:40 AM

Figure 6-12 Bundle Attributes

After defining the bundle definition, we install it. The events portion of the CICS configuration is complete.

6.2.2 WebSphere Message Broker configuration

We had to set up our environment to facilitate communications among the various components, which consisted of the following activities:

- ▶ Defining MQ channels between the Qmgr MQCR on z/OS and the broker Qmgr QM70 on Microsoft Windows XP
- ▶ Defining remote queue definitions from MQCR pointing to local queues on QM70
- ▶ Adding the WebSphere Business Events V7 event nodes to the WebSphere Message Broker V7.0 toolkit
- ▶ Making broker component runtime changes to accommodate the new WebSphere Business Events nodes
- ▶ Defining MQ channels between Qmgr QM70 and WebSphere Business Monitor

- Defining a remote queue definition between Qmgr QM70 and WebSphere Business Events

Figure 6-13 shows the MQ resources that need to be defined on the Microsoft Windows XP server that hosts WebSphere Message Broker and WebSphere Business Events.

QUEUE MANAGER - QM70		
Resource Type	Label	Description
Recv Channel	MQCR.TO.QM70	Channel for incoming CICS event messages
Local Queue	WMB.CICS.CBE.ORDER	Input queue for CICSItemOrder message flow
Local Queue	WMB.CICS.CBE.INSUF.STOCK	Input queue for CICSInsufficientStockReorder message flow
Sender Channel	xxxxxxx	Channel for sending event messages to WBM
Remote Queue	WMB.WMB.CBE.INSUF.STOCK	Remote queue pointing to local queue on WBM

Figure 6-13 Queue Manager resources

We add the WebSphere Business Events event nodes.

Installing WebSphere Business Events support

After installation, we restart the toolkit for the two nodes to appear in the WebSphere Business Events folder within the message processing node palette. Before deploying a message flow using these nodes, we must install the runtime support for the Business Events nodes on the WebSphere Message Broker runtime. To install the runtime support for these nodes, we copy the `com.ibm.wbe.broker.runtime_7.0.0.0.jar` file into the broker `<$MQSI_WORKPATH/shared-classes>` directory. We locate this jar file in the `<WMBT700>\features\com.ibm.wbe.broker.feature_7.0.0.0\runtime` directory where `<WMBT700>` is the WebSphere Message Broker V7 toolkit installation directory.

The next step is to configure the node to use the `WebSphere_WAS_Client` Java Message Service (JMS) provider service. This provider service is a configurable service that ships as part of the V7 broker. We must alter this service to point to the WebSphere Business Events runtime libraries. Example 6-2 shows the commands to use to set and verify this change.

Example 6-2 Commands to point to WebSphere Business Events runtime libraries

```
C:\Program Files\IBM\MQSI\7.0>mqsichangeproperties QM70BRK -c
JMSProviders -o WebSphere_WAS_Client -n jarsURL
-v c:\Progra~1\IBM\WBE70\WAS\runtimes
BIP8071I: Successful command completion.
```

```
C:\Program Files\IBM\MQSI\7.0>mqsireportproperties QM70BRK
-c JMSProviders -o WebSphere_WAS_Client -r
```

```
JMSProviders
  WebSphere_WAS_Client
    clientAckBatchSize='0'
    clientAckBatchTime='0'
    jarsURL='C:\Progra~1\IBM\WBE70\WAS\runtimes'
    jndiEnvironmentParms='default_none'
    nativeLibs='default_Path'
    proprietaryAPIAttr1='default_none'
    proprietaryAPIAttr2='default_none'
    proprietaryAPIAttr3='default_none'
    proprietaryAPIAttr4='default_none'
    proprietaryAPIAttr5='default_none'
    proprietaryAPIHandler='default_none'
```

```
BIP8071I: Successful command completion.
```

We must restart the broker to pick up these changes. The tooling and broker runtime environments are now set up for the WebSphere Business Events event nodes. We proceed with setting up the scenarios.

6.2.3 WebSphere Business Events configuration

See Chapter 9, “WebSphere Business Events scenario” on page 249 for details about the WebSphere Business Events configuration.

6.2.4 WebSphere Business Monitor configuration

The WebSphere Business Monitor configuration is identical for all ESB solutions. We describe this configuration in detail in Chapter 10, “WebSphere Business Monitor” on page 271.

6.2.5 WebSphere Process Server configuration

There is no WebSphere Process Server configuration required for any of the ESB products. We defined scenario 4, which involves WebSphere Process Server, so that WebSphere Business Events will make a web service call directly to WebSphere Process Server. As a result, no ESB function is required to complete scenario 4.

6.3 Scenario 1

Now, we describe scenario 1.

6.3.1 WebSphere Message Broker transformation

Scenario 1 shows a WebSphere Message Broker transformation.

Overview of scenario 1

In this scenario, CICS generates an ItemOrder event message and forwards it to WebSphere Message Broker. WebSphere Message Broker forwards the message unchanged to a local queue, which serves as an archive log, and to WebSphere Business Monitor, which consumes this message and other ItemOrder messages. Additionally, WebSphere Message Broker performs a web service call to retrieve pricing information for the ordered item. After this information is returned, WebSphere Message Broker enriches the original event message with the total order cost and forwards it to WebSphere Business Events in its desired format. Figure 6-14 shows the high-level flow design needed to accomplish this scenario.

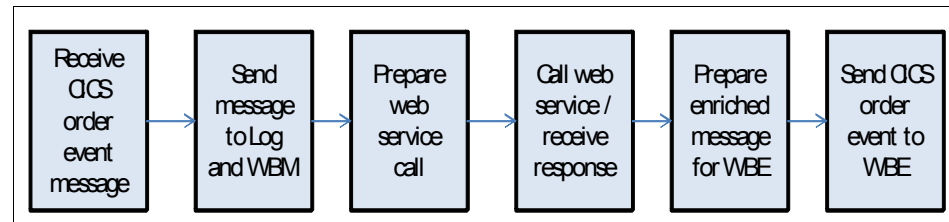


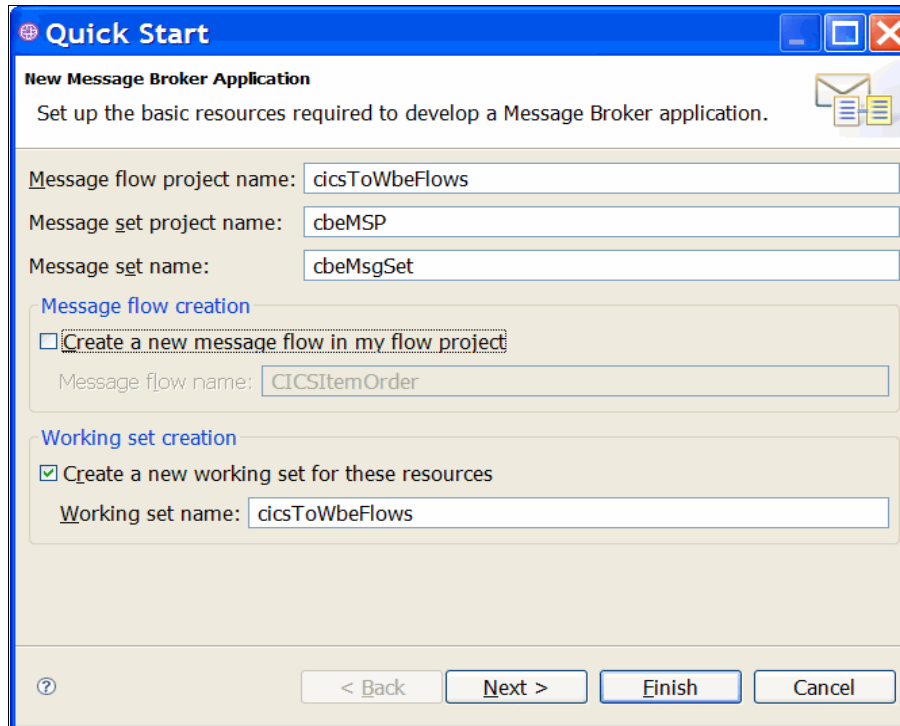
Figure 6-14 Flow design overview

Transformation nodes: WebSphere Message Broker provides a number of transformation nodes for performing message transformation. You can use any of the nodes within the Message Flow Editor Transformation folder to transform the CICS event message. These scenarios merely show one method of WebSphere Message Broker enriching and transforming a CICS event message prior to sending it to downstream consumers.

We now explain the steps that were used to build the message flow and the message sets that were used to process this scenario. We assume that you have a general knowledge of the WebSphere Message Broker Toolkit and the activities that relate to creating and building message flows and message sets.

Refer to the IBM WebSphere Message Broker, Version 7.0, Information Center for additional details related to message broker application development.

We chose the Start from Scratch Quick Start wizard, because it allows us to create both a message flow project and a message set project at the same time (Figure 6-15).



The screenshot shows a 'Quick Start' wizard window titled 'New Message Broker Application'. The window contains the following fields and options:

- Message flow project name:** cicsToWbeFlows
- Message set project name:** cbeMSP
- Message set name:** cbeMsgSet
- Message flow creation:**
 - Create a new message flow in my flow project
 - Message flow name:** CICSItemOrder
- Working set creation:**
 - Create a new working set for these resources
 - Working set name:** cicsToWbeFlows

At the bottom of the window, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. A help icon (?) is also present in the bottom left corner.

Figure 6-15 Start from scratch wizard

Creating the message sets from schemas

WebSphere Message Broker needs to know the structure of an incoming message so that the message can be parsed and a logical message tree structure can be built. Then, the various elements within the message can be referenced and modified by the message flow.

Most applications that generate XML-based messages can also provide an XML schema file, which defines the structure of the message it produced. In certain cases, multiple schemas are used to define a single message for purposes, such as ease of maintenance, reuse, and readability.

In our case, the following three schema descriptions define the CICS event message that is received:

- ▶ Common base event
- ▶ Static event
- ▶ Dynamic payload

WebSphere Message Broker needs to use all three schema definitions to successfully validate the incoming message. To initiate this setup, we first import the schemas into the toolkit using File → Import → File System. Then, we point to the location where the schemas are located on the file system and to the message set project to which we want to import them. Importing these schemas into the message set project helps keep all the resources used to build the solution in a single location.

In our example, we import the schema files into the message set project named cbeMSP (Figure 6-16).

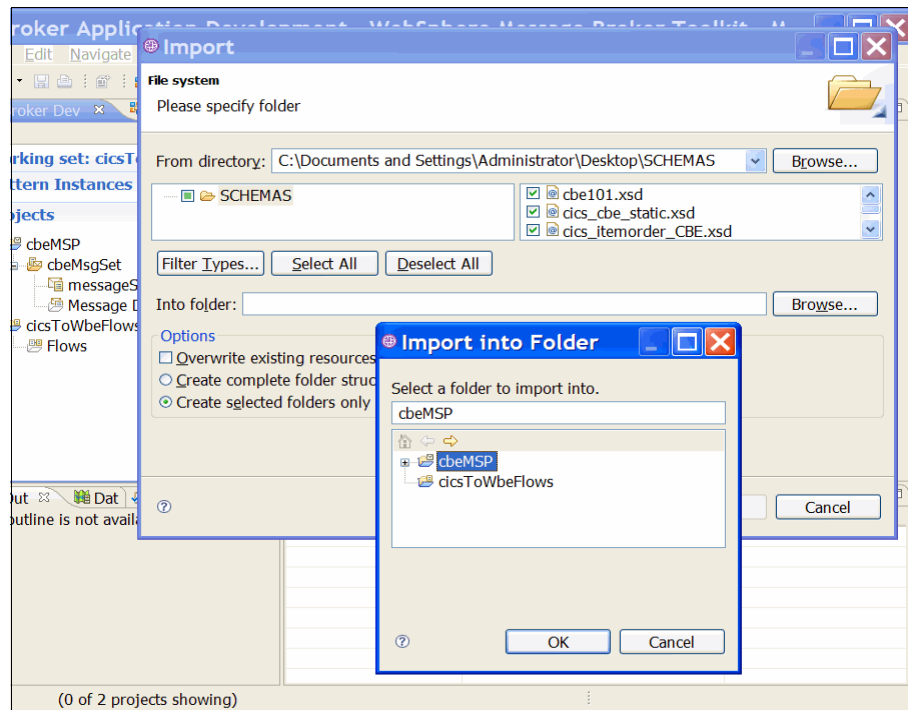


Figure 6-16 Schema import

We have successfully imported the three schema files into the workspace. Now, we define three message definition files. We use *Message definition files* to store the message model objects that are used by message broker. We create

message definition files within the cbeMSP project by using New → Message Definition File using → XML schema. This option launches the New Message Definition File wizard that takes us through the schema selection and message definition process (Figure 6-17 and Figure 6-18 on page 150).

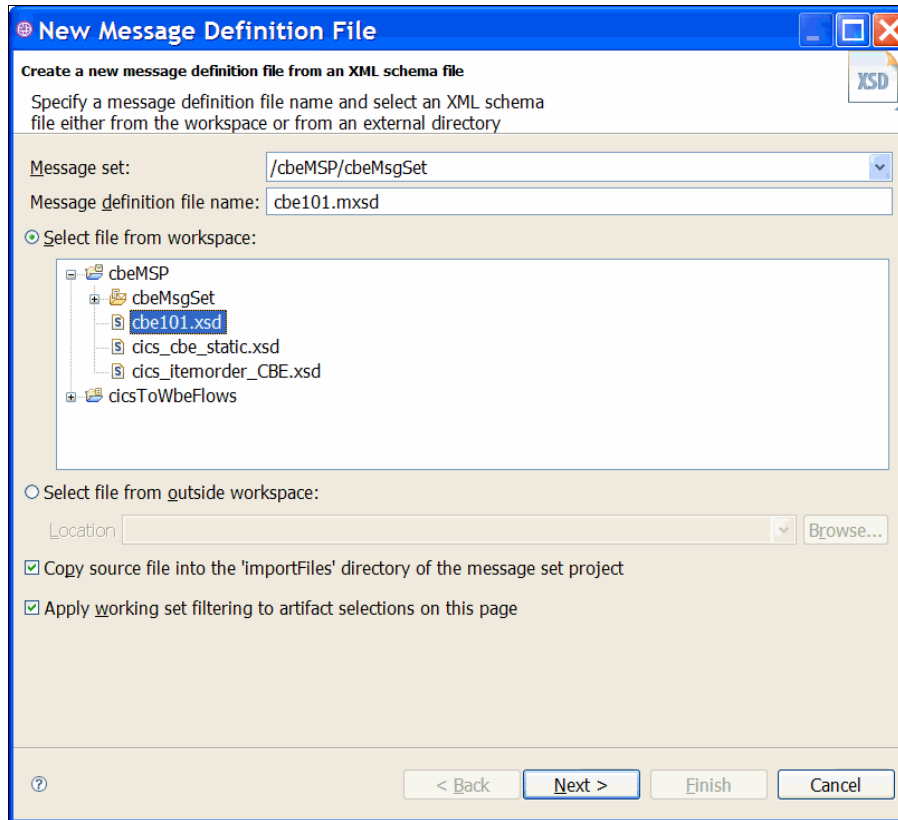


Figure 6-17 New Message Definition File: Schema selection

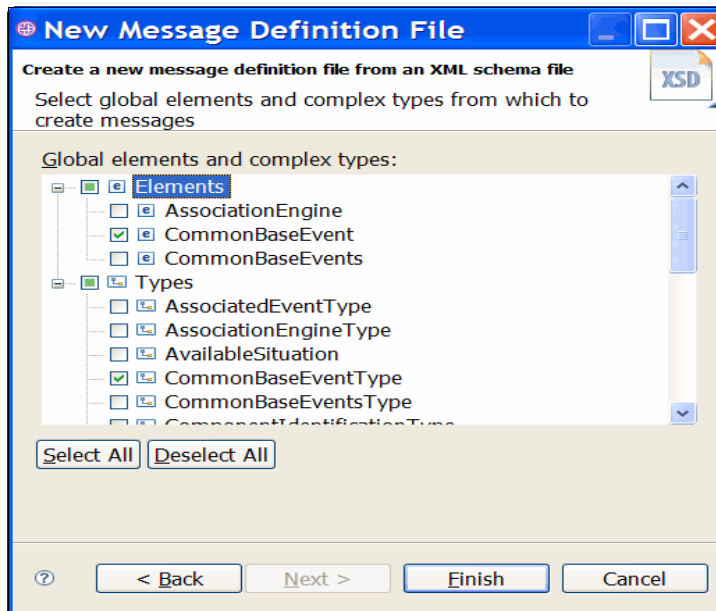


Figure 6-18 New Message Definition File: Element selection

We repeat these tasks two more times to create message definition files for the two remaining schema files. Figure 6-19 on page 151 shows the three message definitions within the cbeMSP project folder.

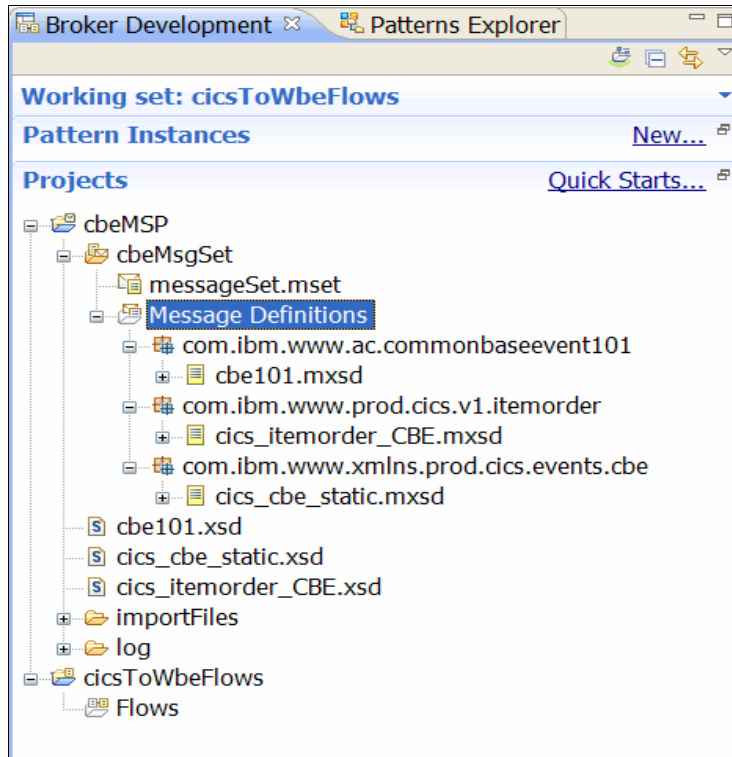


Figure 6-19 Newly added message definition files

Importing the schemas

We use the message set editor Add Import function to link the namespaces used within the three schemas, which allows the WebSphere Message Broker run time to parse the entire message. To be able to process all three portions of the incoming message, we need to import the two schemas into the primary schema. To add these imports, we use the message set editor to open the `cbe101.mxsd` file, select the Properties view, and then, select Imports → Add Import (Figure 6-20 on page 152).

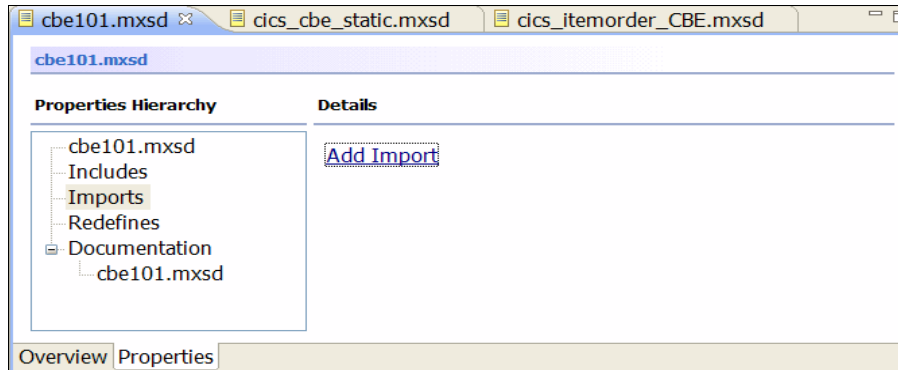


Figure 6-20 Add Import

Figure 6-21 shows how we drill down into the namespaces until we locate one of the two message definition files that we want to import.

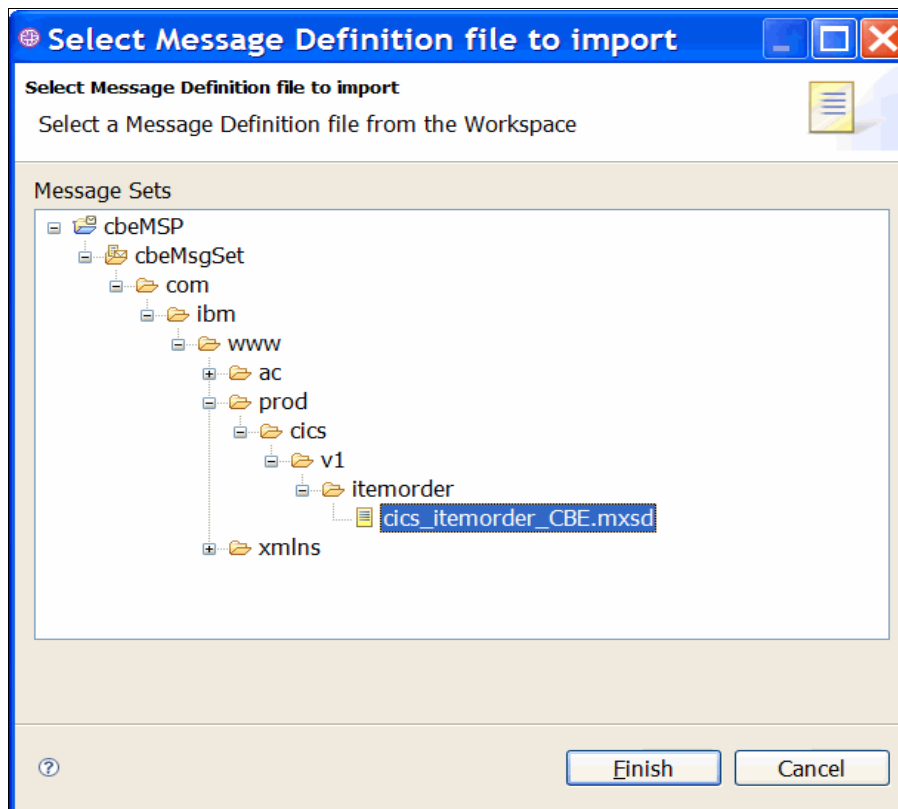


Figure 6-21 Message definition import selection

We then repeat the process to select the remaining message definition. We verify that we successfully imported these message definitions by examining the cbe101.mxsd xsd file.

Example 6-3 Snippet of cbe101.mxsd showing imports

```
<?xml version="1.0" encoding="UTF-8"?><xsd:schema
elementFormDefault="qualified"...
<xsd:import namespace="http://www.ibm.com/prod/cics/v1/ItemOrder"
schemaLocation...
<xsd:import namespace="http://www.ibm.com/xmlns/prod/cics/events/CBE"
schemaLocation...<xsd:complexType name="CommonBaseEventType">
....
```

We have now successfully created and linked the three message definitions that represent the three portions of the incoming event message.

Creating the message set from Web Services Description Language

The message flow is required to call a CICS-based web service to retrieve pricing information for the item that was ordered. The New Message Definition wizard provides an option to build a message definition using a specific Web Services Description Language (WSDL) to set the message definitions needed to interact with the web service. A *WSDL definition* tells a client how to compose a webservice request and describes the interface that is provided by the web service provider. To begin this process, we create a new message set project to accommodate the WSDL message set.

After creating the inquirySingleMSP message set project, we then create a new message definition file using the imported WSDL as its source. This task results in a “deployable WSDL” object appearing within the inquireSingleMSP project. We can then use a deployable WSDL to configure SOAP nodes for the webservice request and response portion of the flow.

Building the message flow

We construct our message flow and configure it to perform the required tasks. To begin, we populate a blank canvas with the nodes that are needed for input, output, and enrichment/transformation. Then, we label the nodes to describe their functionality (Figure 6-22 on page 154).

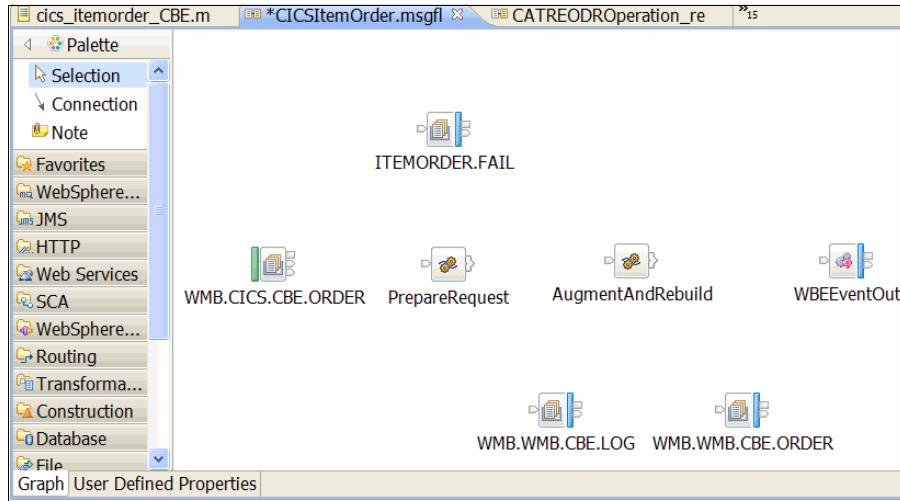


Figure 6-22 Message flow editor

Next, we drag the deployable WSDL from the message set project to the message editor canvas, which results in launching the Configure New Web Service Usage wizard. We select the Invoke web service from message flow option to drive the correct node selections. Figure 6-23 on page 155 shows the web service parameters and binding operations setting, which were automatically set from the WSDL.

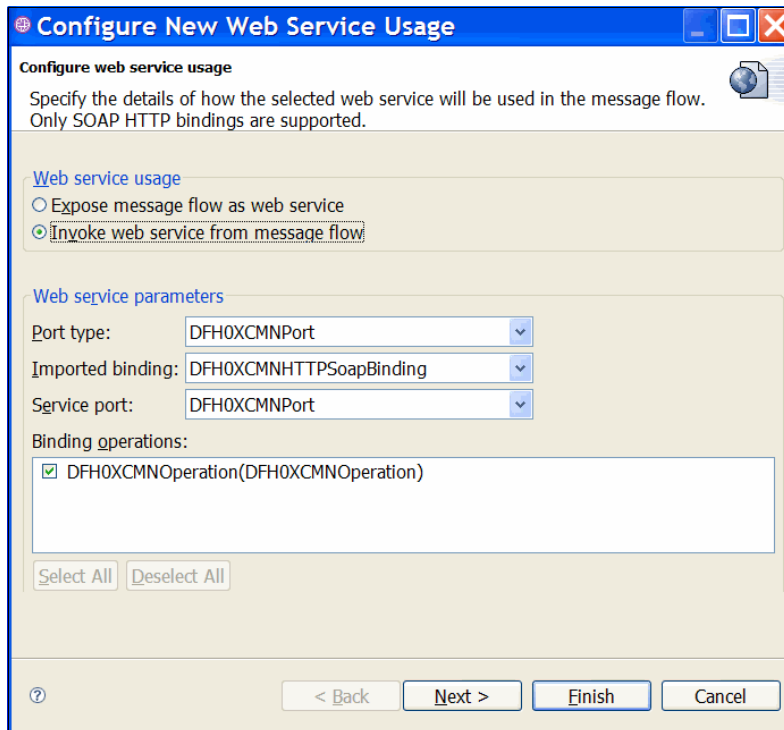


Figure 6-23 Configure new web service

This action results in a subflow labeled `DFH0XCMNOperation_inquireSingle` that is added to the message flow, which is configured with the details derived from the WSDL. Double-clicking the subflow node results in the message flow editor displaying the underlying nodes (Figure 6-24 on page 156).

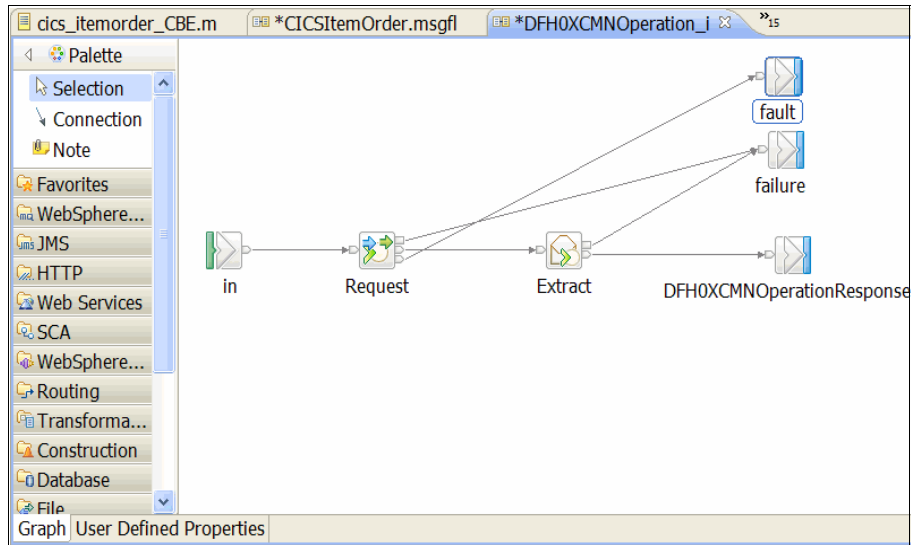


Figure 6-24 Subflow created by WSDL drag and drop

Displaying the properties of the SOAP Request node reveals the web service configuration properties, which were set automatically as a result of the WSDL (Figure 6-25).

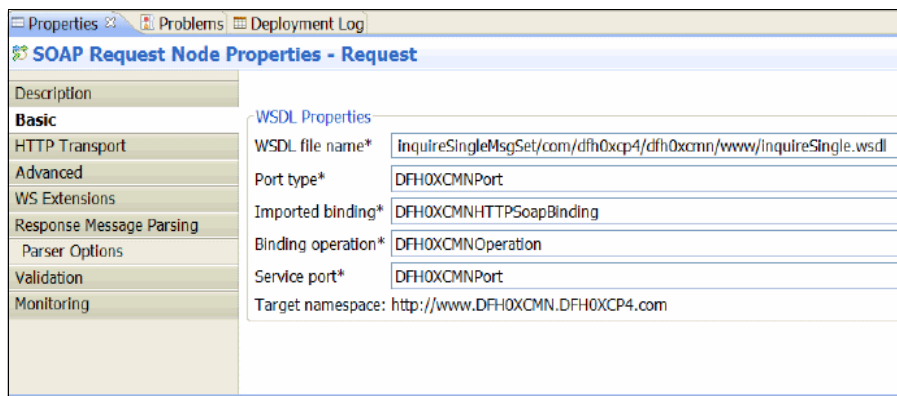


Figure 6-25 SOAP Request node properties

We wire the nodes and subflow and code the compute node's extended SQL (ESQL) to perform the following actions.

We use the PrepareRequest compute node for the following functions:

- ▶ Passing the original CICS event message to MQ queue serving as an archive log

- ▶ Passing the original CICS event message to WebSphere Business Monitor for dashboard display
- ▶ Saving a copy of the original message in an environment variable for later use
- ▶ Preparing the message for a web service call

We code the ESQL that is shown in Example 6-4 into the PrepareRequest compute node.

Example 6-4 PrepareRequest compute node ESQL

```

DECLARE ns37 NAMESPACE 'http://www.DFHOCMN.DFHOCXCP4.Response.com';
  DECLARE ns13 NAMESPACE
'http://wbe.ibm.com/6.2/Event/ItemOrder-wbe';
  DECLARE ns NAMESPACE 'http://www.DFHOCMN.DFHOCXCP4.Request.com';

  DECLARE cbe NAMESPACE 'http://www.ibm.com/AC/commonbaseevent1_0_1';
  DECLARE ns2 NAMESPACE 'http://www.ibm.com/prod/cics/v1/ItemOrder';
  DECLARE cics NAMESPACE
'http://www.ibm.com/xmlns/prod/cics/events/CBE';

CREATE COMPUTE MODULE CICSItemOrder_PrepareRequest

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

  -- Save off a copy of the original msg for later use..
  CREATE LASTCHILD Of Environment.Variables.Input DOMAIN('XMLNSC');
  Set Environment.Variables.Input.XMLNSC=InputRoot.XMLNSC;

  -- Send a copy of original message to the log queue.
  Set OutputRoot=InputRoot;
  PROPAGATE TO TERMINAL 'out1';

  -- Send a copy of original message to WBM.
  Set OutputRoot=InputRoot;
  PROPAGATE TO TERMINAL 'out2';

  -- Prepare the webservice call message..
  Set
OutputRoot.XMLNSC.ns:DFHOCMNOperation.ns:ca_request_id='01INQS';
  Set OutputRoot.XMLNSC.ns:DFHOCMNOperation.ns:ca_return_code=0;
  Set
OutputRoot.XMLNSC.ns:DFHOCMNOperation.ns:ca_response_message='0';

```

```

    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_item_
ref_req=

InputRoot.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:"payload-data".ns2
:payload.ns2:item_ref_number;
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:filler1=
'0';
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:filler2=
'0';
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:ca_sngl_item_ref=0;
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:ca_sngl_description='0';
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:ca_sngl_department=0;
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:ca_sngl_cost='0';
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:in_sngl_stock=0;
    Set
OutputRoot.XMLNSC.ns:DFHOXCMNOperation.ns:ca_inquire_single.ns:ca_singl
e_item.ns:on_sngl_order=0;

    RETURN TRUE;
    END;
END MODULE;

```

We use the AugmentAndRebuild compute node for these functions:

- ▶ Extracting pricing information from the webservice response
- ▶ Computing the total order cost
- ▶ Transforming the original CBE message to WBE format message
- ▶ Adding the total order cost

We code the ESQL that is shown in Example 6-5 on page 159 into the AugmentAndRebuild compute node.

Example 6-5 AugmentAndRebuild compute node ESQL

```
CREATE COMPUTE MODULE CICSItemOrder_AugmentAndRebuild

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

DECLARE item_price DECIMAL;
DECLARE order_quantity DECIMAL;
DECLARE total_cost DECIMAL;

Create LASTCHILD of OutputRoot.XMLNSC NAMESPACE ns13 NAME
'connector' VALUE NULL;

Set OutputRoot.XMLNSC.ns13:connector.(XMLNSC.Attribute)name='CICS
Catalog Orders';
Set
OutputRoot.XMLNSC.ns13:connector.(XMLNSC.Attribute)version='6.2';

Create LASTCHILD of OutputRoot.XMLNSC.ns13:connector NAMESPACE ns13
NAME 'connector-bundle' VALUE NULL;

Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".(XMLNSC.Attrib
ute)name='Event_ItemOrder-esb';
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".(XMLNSC.Attrib
ute)type='Event';
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Binding-user-tag"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"context-info".cics:bindingname;
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Binding-user-tag".(XMLNSC.Attribute)type='String';
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Network-UOWID"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"context-info".cics:UOWid;
```

```

    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Network-UOWID".(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:businessevent=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"context-info".cics:eventname;
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:businessevent.(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Capture-Spec-Name"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"context-info".cics:capturespecname;
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Context.ns13:"Capture-Spec-Name".(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:userid=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"payload-data".ns2:payload.ns2:userid;
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"charge_dept"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"payload-data".ns2:payload.ns2:charge_dept;
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"charge_dept".(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"item_ref_number"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"payload-data".ns2:payload.ns2:item_ref_number;

```

```

Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"item_ref_number".(XMLNSC.Attribute)type='Real';
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"quantity_req"=

Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:
"payload-data".ns2:payload.ns2:quantity_req;
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"quantity_req".(XMLNSC.Attribute)type='Real';

Set order_quantity =
cast(Environment.Variables.Input.XMLNSC.cbe:CommonBaseEvent.cics:event.
cics:"payload-data".ns2:payload.ns2:quantity_req AS DECIMAL);
Set item_price =
cast(InputRoot.XMLNSC.ns37:DFHOXCMNOperationResponse.ns37:ca_inquire_si
ngle.ns37:ca_single_item.ns37:ca_sngl_cost AS DECIMAL);
Set total_cost = order_quantity*item_price;
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"order_cost"=total_cost;
Set
OutputRoot.XMLNSC.ns13:connector.ns13:"connector-bundle".ns13:ItemOrder
_Data.ns13:"order_cost".(XMLNSC.Attribute)type='Real';

RETURN TRUE;
END;

END MODULE;

```

We change the following node properties from their default settings (Figure 6-26 on page 162).

NODE TYPE	NODE NAME	NODE PROPERTY	SETTING
MQInput	WMB.CICS.CBE.ORDER	Queue Name Validate Failure Action	WMB.CICS.CBE.ORDER Content and Value Exception
MQCompute	PrepareRequest	ESQL Module	CICSItemOrder_PrepareRequest
MQCompute	AugmentAndRebuild	ESQL Module	CICSItemOrder_AugmentAndRebuild
MQOutput	WMB.WMB.CBE.LOG	Queue Name	WMB.WMB.CBE.LOG
MQOutput	WMB.WMB.CBE.WBM	Queue Name	WMB.WMB.CBE.WBM
MQOutput	ITEMORDER.FAIL	Queue Name	ITEMORDER.FAIL

Figure 6-26 Additional node property changes

We leave the `WBEEEventOutputNode` setting defaults, because we use WebSphere Business Events on the same Microsoft Windows server as the WebSphere Message Broker installation, and we verify the WebSphere Application Server bootstrap port as the default port of 2809 (Figure 6-27).

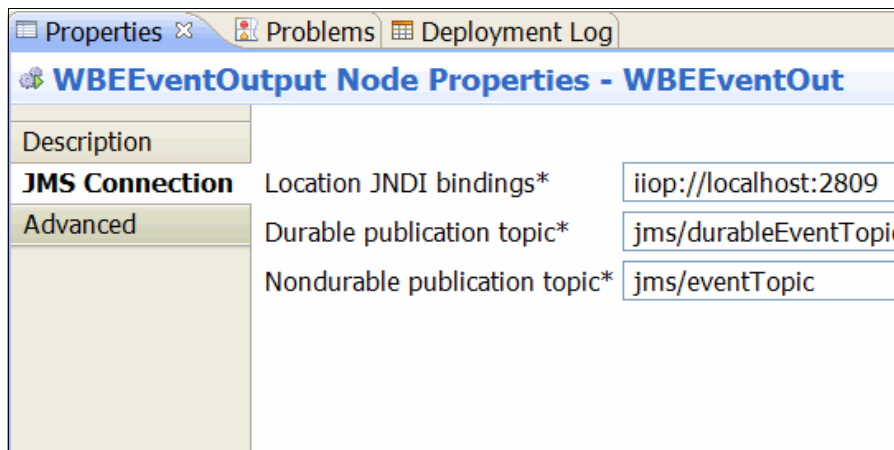


Figure 6-27 WebSphere Business Events Event Output node properties

Deploying the solution

Now that flow has been properly configured, a broker archive (BAR) file containing the newly created artifacts has been created and saved. This file was then deployed to the TESTWBE execution group running under broker QM70BRK. We verify that the deployment was successful by checking for any errors during the time of the deployment and by querying the broker run time using the `mqsilist QM70BRK -e TESTWBE` command to verify that the new flow and message sets are now running (Example 6-6 on page 163).

Example 6-6 MQSILIST command output

BIP8071I: Successful command completion.

C:\Program Files\IBM\MQSI\7.0>mqsilist QM70BRK -e TESTWBE

BIP1288I: Message flow 'CICSItemOrder' on execution group 'TESTWBE' is running.

BIP1290I: File 'cbeMsgSet.xszip' is deployed to execution group 'TESTWBE'.

BIP1290I: File 'inquireSingleMsgSet.xszip' is deployed to execution group 'TESTWBE'.

BIP8071I: Successful command completion.

Verifying the solution with the debugger

Now that the message flow is running, we can test it by putting a CICS ItemOrder event to the message flow input queue and tracking its progress using the message flow debugger. We place breakpoints between all nodes to intercept the message at each step and to examine its contents.

Figure 6-28 shows the message just after it is received.

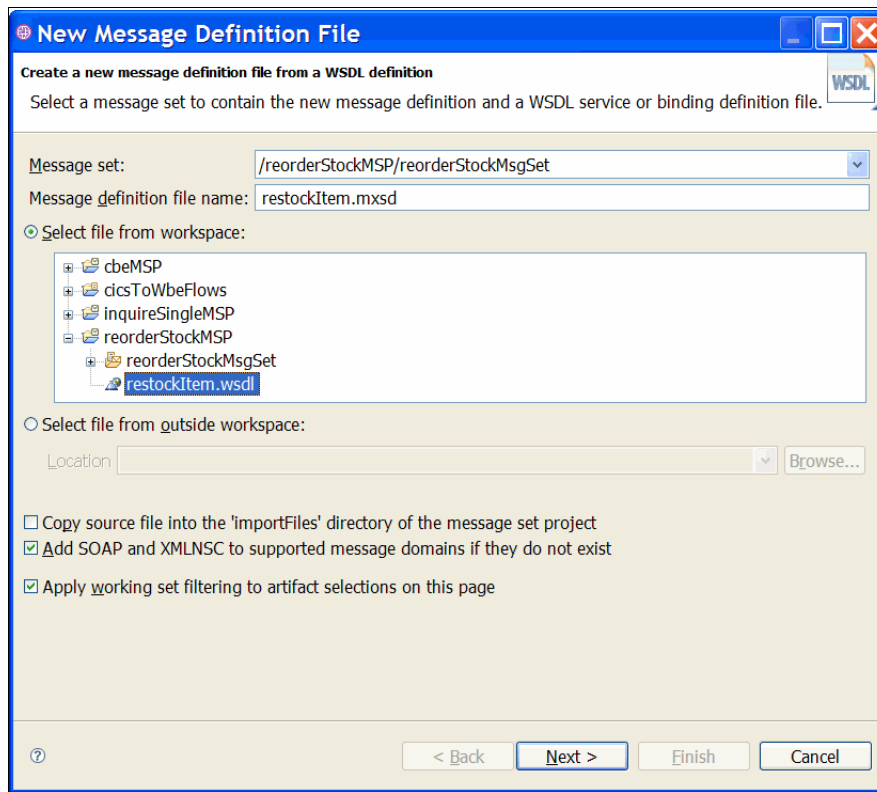


Figure 6-28 CICS ItemOrder input message

Figure 6-29 shows the message after the PrepareRequest compute node and just prior to entering the webservice request subflow.

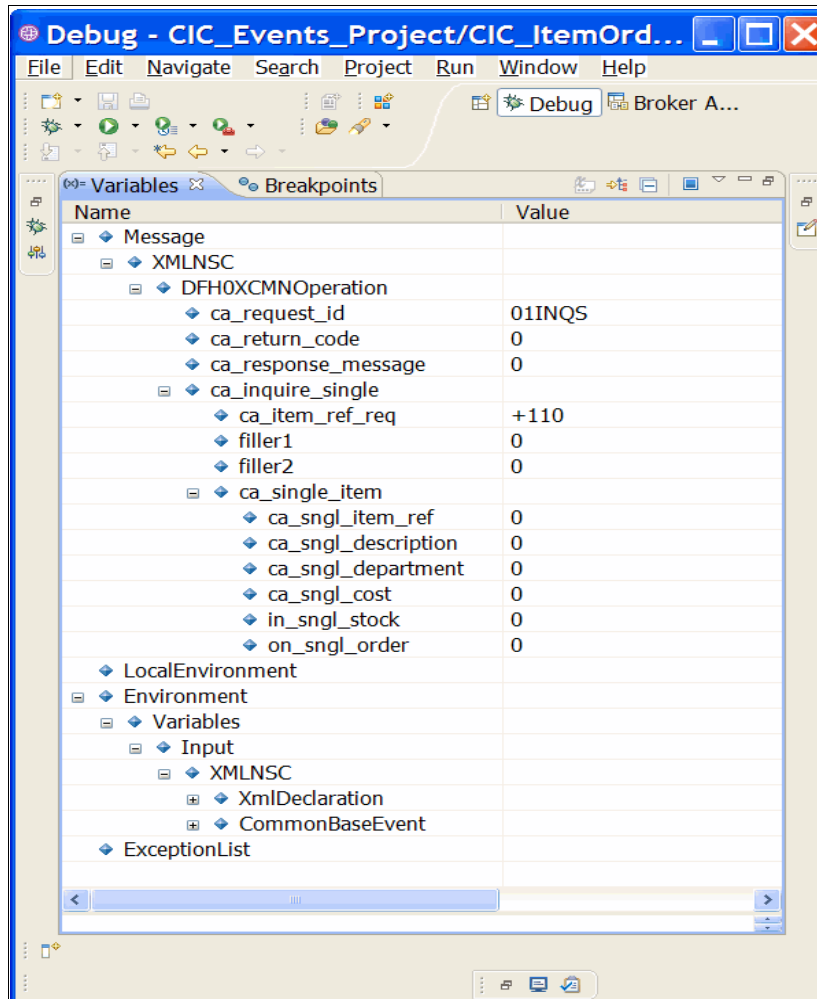


Figure 6-29 Message after PrepareRequest compute node

Figure 6-30 shows the message returned from the webservice, which contains the `ca_sngl_cost` element that allows us to calculate the total order cost.

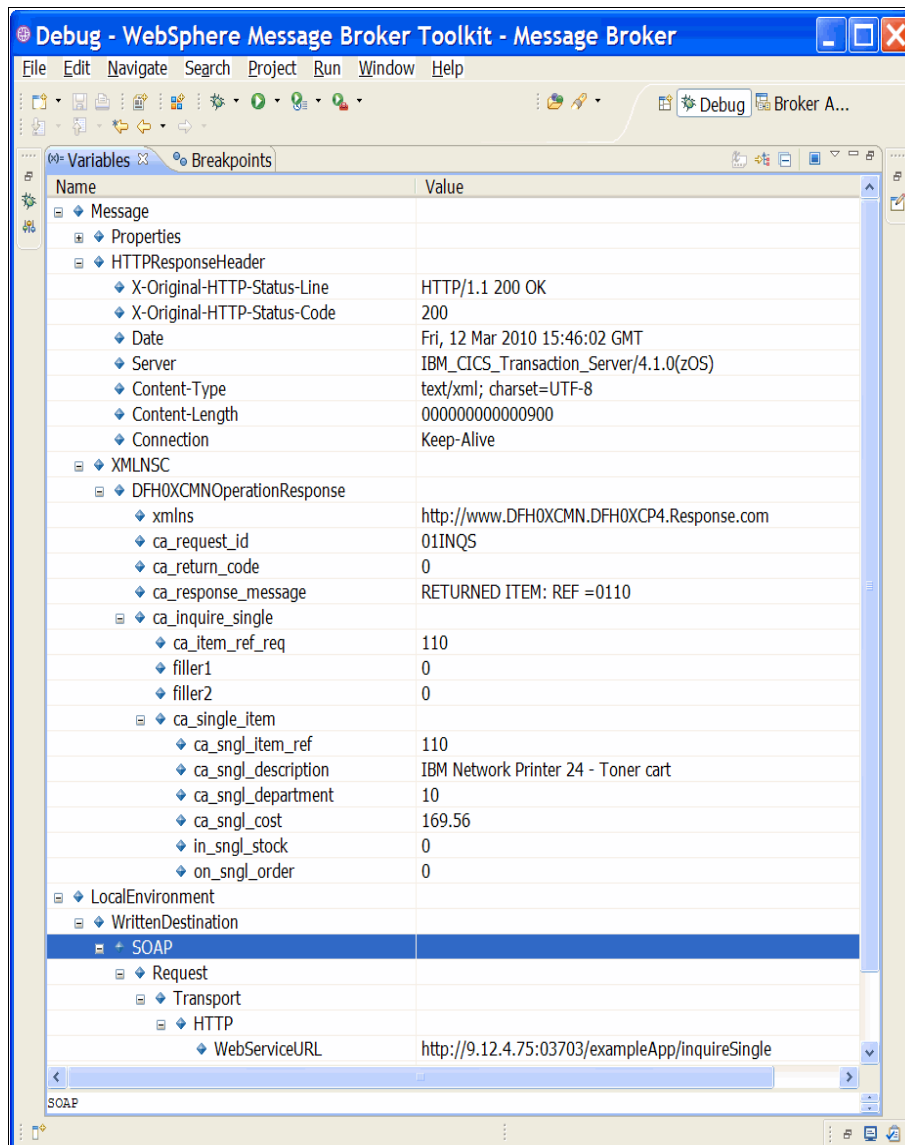


Figure 6-30 Message with webservice response

Figure 6-31 shows the enriched message in WBE format just prior to entering the WebSphere Business Events Event Out node.

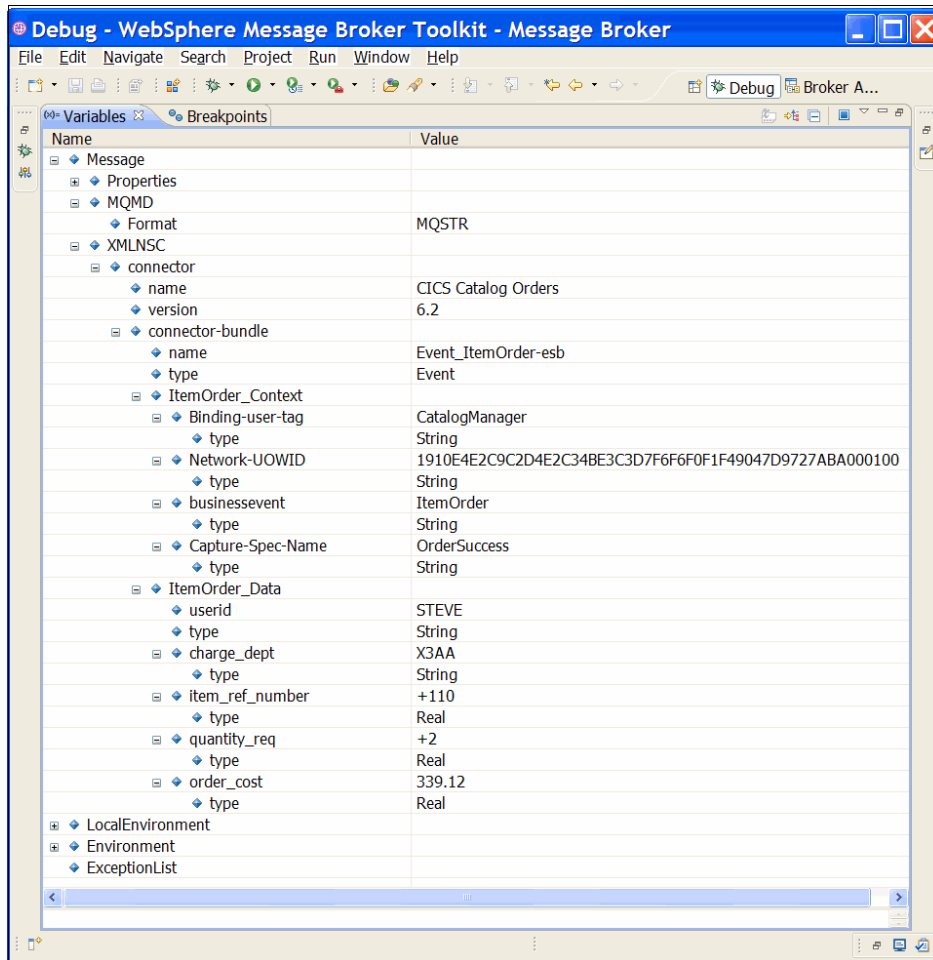


Figure 6-31 Enriched message in WBE format

The final verification is WebSphere Business Events successfully receiving and consuming the CICS ItemOrder events.

6.4 Scenario 2

The implementation of scenario 2 does not require any additional ESB function.

6.5 Scenario 3

Next, we describe scenario 3.

6.5.1 Scenario 3 overview

This scenario is somewhat similar to scenario 1 in that a CICS InsufficientStock event is received by WebSphere Message Broker. WebSphere Message Broker forwards the message unchanged to a local queue, which serves as an archive log, and to WebSphere Business Monitor, which consumes this message and other InsufficientStock messages. Additionally, WebSphere Message Broker performs a webservice call to initiate a reorder action. After this call completes, WebSphere Message Broker converts the original event message to the format required by WebSphere Business Events and sends it using the WBEEventOutput node. Figure 6-32 shows the high-level flow design needed to accomplish this task.

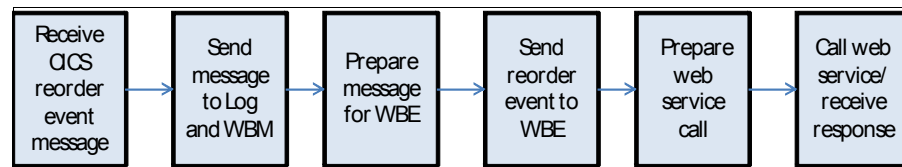


Figure 6-32 Insufficient stock high-level flow design

Differences: Given the similarity of the setup for this scenario compared to the setup for scenario 1, we describe only those details specific to this scenario. Refer to scenario 1 details or the IBM WebSphere Message Broker, Version 7.0, Information Center for additional assistance.

Setting up the message sets

We begin this setup as we did for the first scenario. We define a File → New → Message Set and specify cbe2MSP as the message set project name and cbe2MsgSet as the actual message set name. In this scenario, we work with a FailedOrder CICS event message, which has a similar structure to the ItemOrder event message described previously. However, the FailedOrder CICS event message uses a separate schema to define the dynamic payload portion of the message. The CICS event message that is being received is defined by the following three schema descriptions:

- ▶ Common base event
- ▶ Static CICS event
- ▶ Dynamic FailedOrder event

Next, we launch the New Message Definition File wizard to create three message definition files using the three schemas as input to this process. After the three messages have been added to the message set, we must import the message definition files from `cics_cbe_static.mxsd` and `failedorder.mxsd` into the message schema definition for the `cbe101.mxsd` file. Finally, we save the updated `cbe101.mxsd` file, and we are ready to describe the WSDL. Figure 7-33 shows the results of the Add Imports.

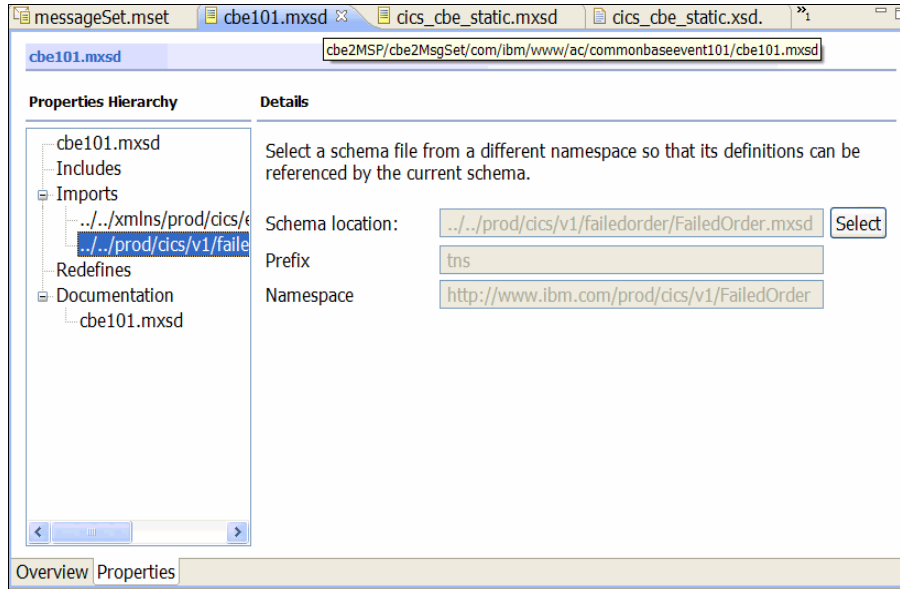


Figure 6-33 Results of Add Imports

Creating a message set from the WSDL

We provide a new WSDL that defines the new web service that is used to drive a Reorder Request. After first creating the new message set project, `reorderStockMSP`, we add the WSDL to the `reorderStockMsgSet` message set by using `File → New → Message Definition File using → WSDL`. Adding the WSDL resulted in a deployable WSDL being created in the message set. Also, adding the WSDL creates several message definitions that are needed to handle the inbound and outbound messages and the soap header that is used by the soap Request node when processing the web service call. Figure 6-34 on page 170 shows the contents of the `reorderStockMSP` project.

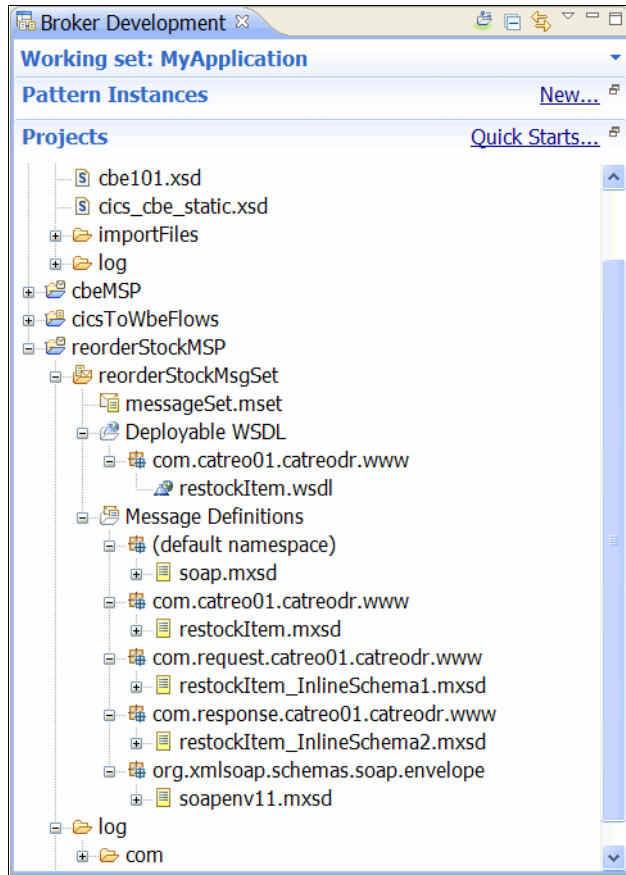


Figure 6-34 reorderStockMsgSet contents

Creating the message flow

We now construct our message flow and configure it to perform the required tasks. To begin, we populate a blank canvas with the nodes needed for input, output, and transformation. Then, we label them to describe their functionality. After we complete the labels, we drop the WSDL onto the canvas and create the new web service subflow, specifying “Invoke web service from message flow” to create the proper subflow nodes. Figure 6-35 on page 171 shows the wired CICSInsufficientStockReOrder flow that is used for this scenario.

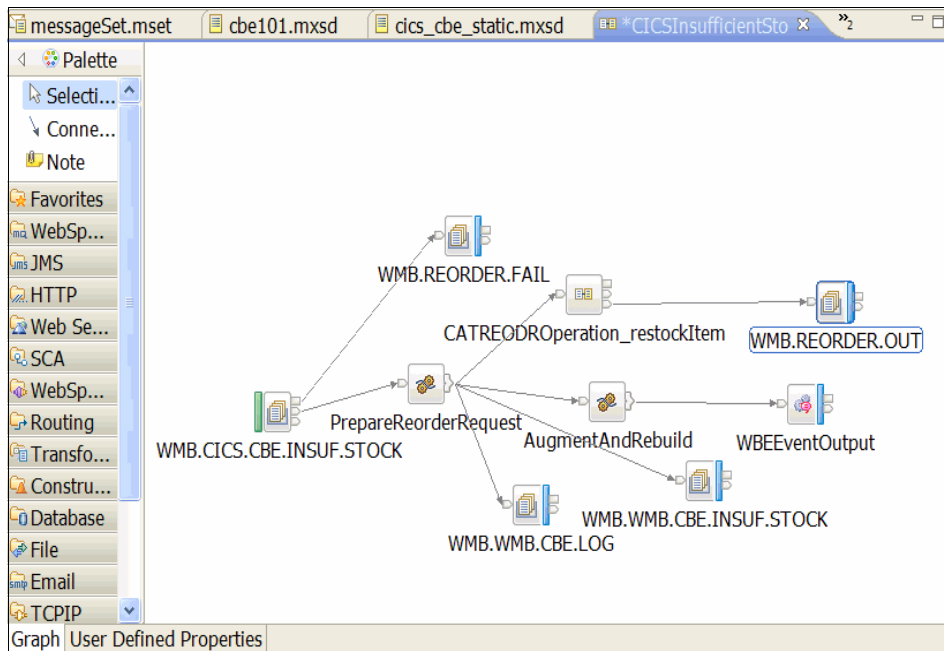


Figure 6-35 Wired CICSInsufficientStock message flow

We code the compute node's ESQL to perform the following actions.

We use the PrepareReorderRequest compute node for these functions:

- ▶ Passing the original CICS event message to MQ queue serving as an archive log
- ▶ Passing the original CICS event message to WebSphere Business Monitor for dashboard display
- ▶ Saving a copy of the original message in an environment variable for later use
- ▶ Preparing the message for a webservice call

We code the ESQL that is shown in Example 6-7 into the compute node named PrepareReorderRequest.

Example 6-7 PrepareReorderRequest compute node ESQL

```

DECLARE ns14 NAMESPACE
'http://we.ibm.com/6.2/Event/Event_FailedOrder-esb';
  DECLARE ns6 NAMESPACE 'http://www.CATREODR.CATRE001.Request.com';
  DECLARE tns NAMESPACE
'http://www.ibm.com/prod/cics/v1/FailedOrder';

```

```

CREATE COMPUTE MODULE
CICSInsufficientStockReOrder_PrepareReorderRequest

    DECLARE cbe NAMESPACE 'http://www.ibm.com/AC/commonbaseevent1_0_1';
    DECLARE cics NAMESPACE
'http://www.ibm.com/xmlns/prod/cics/events/CBE';

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

    -- Save off a copy of the original msg for later use..
    CREATE LASTCHILD Of Environment.Variables.Input DOMAIN('XMLNSC');
    Set Environment.Variables.Input.XMLNSC=InputRoot.XMLNSC;

    -- Send a copy of original message to the log queue.
    Set OutputRoot=InputRoot;
    PROPAGATE TO TERMINAL 'out1';

    -- Send a copy of original message to WBM.
    Set OutputRoot=InputRoot;
    PROPAGATE TO TERMINAL 'out2';

    -- Send a copy of original message to WBE.
    Set OutputRoot=InputRoot;
    PROPAGATE TO TERMINAL 'out3';

    -- Prepare a webservice reorder call message..
    Set
OutputRoot.XMLNSC.ns6:CATREODROperation.ns6:reorderRequest.ns6:reorderI
tem=

InputRoot.XMLNSC.cbe:CommonBaseEvent.cics:event.cics:"payload-data".tns
:payload.tns:item_ref_number;

    RETURN TRUE;
    END;
END MODULE;

```

We use the AugmentAndRebuild compute node to transform the original CBE format message to the format that WebSphere Business Events requires.

We code the ESQL that is shown in Example 6-8 into the AugmentAndRebuild compute node.

Example 6-8 ESQL for the AugmentAndRebuild compute node

```
CREATE COMPUTE MODULE CICSInsufficientStockReOrder_AugmentAndRebuild

    DECLARE cbe NAMESPACE 'http://www.ibm.com/AC/commonbaseevent1_0_1';
    DECLARE cics NAMESPACE
'http://www.ibm.com/xmlns/prod/cics/events/CBE';

    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

        Set OutputRoot.MQMD.Format=MQFMT_STRING;
        Create LASTCHILD of OutputRoot.XMLNSC NAMESPACE ns14 NAME
'connector' VALUE NULL;

        Set
OutputRoot.XMLNSC.ns14:connector.(XMLNSC.Attribute)xmlns='http://wbe.ib
m.com/6.2/Event/Event_FailedOrder-esb';
        Set OutputRoot.XMLNSC.ns14:connector.(XMLNSC.Attribute)name='CICS
Catalog ReOrder';
        Set
OutputRoot.XMLNSC.ns14:connector.(XMLNSC.Attribute)version='6.2';

        Create LASTCHILD of OutputRoot.XMLNSC.ns14:connector NAMESPACE ns14
NAME 'connector-bundle' VALUE NULL;

        Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".(XMLNSC.Attrib
ute)type='Event';
        Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".(XMLNSC.Attrib
ute)name='Event_FailedOrder-esb';
        Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:"Binding-user-tag"=Environment.Variables.Input.XMLNSC.c
be:CommonBaseEvent.cics:event.cics:"context-info".cics:bindingname;
        Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:"Binding-user-tag".(XMLNSC.Attribute)type='String';
        Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
```

```

er_Context.ns14:"Network-UOWID"]=Environment.Variables.Input.XMLNSC.cbe:
CommonBaseEvent.cics:event.cics:"context-info".cics:UOWid;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:"Network-UOWID".(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:businessevent=Environment.Variables.Input.XMLNSC.cbe:Co
mmonBaseEvent.cics:event.cics:"context-info".cics:eventname;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:businessevent.(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:"Capture-Spec-Name"]=Environment.Variables.Input.XMLNSC.
cbe:CommonBaseEvent.cics:event.cics:"context-info".cics:capturespecname
;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Context.ns14:"Capture-Spec-Name".(XMLNSC.Attribute)type='String';

    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:userid=Environment.Variables.Input.XMLNSC.cbe:CommonBaseEv
ent.cics:event.cics:"payload-data".tns:payload.tns:userid;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:userid.(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:"charge_dept"]=Environment.Variables.Input.XMLNSC.cbe:Commo
nBaseEvent.cics:event.cics:"payload-data".tns:payload.tns:charge_dept;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:"charge_dept".(XMLNSC.Attribute)type='String';
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:"item_ref_number"]=Environment.Variables.Input.XMLNSC.cbe:C
ommonBaseEvent.cics:event.cics:"payload-data".tns:payload.tns:item_ref_
number;
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:"item_ref_number".(XMLNSC.Attribute)type='Real';
    Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd

```



```

er_Data.ns14:"quantity_req"=Environment.Variables.Input.XMLNSC.cbe:Comm
onBaseEvent.cics:event.cics:"payload-data".tns:payload.tns:quantity_req
;
  Set
OutputRoot.XMLNSC.ns14:connector.ns14:"connector-bundle".ns14:FailedOrd
er_Data.ns14:"quantity_req".(XMLNSC.Attribute)type='Real';

  RETURN TRUE;
  END;

END MODULE;

```

We change the node properties that are shown in Figure 6-36 from their default property settings.

NODE TYPE	NODE NAME	NODE PROPERTY	SETTING
MQInput	WMB.CICS.CBE.INSUF.STOCK	Queue Name Validate Failure Action	WMB.CICS.CBE.INSUF.STOCK Content and Value Exception
MQCompute	PrepareReorderRequest	ESQL Module	CICSInsufficientStockReOrder_PrepareReorderRequest
MQCompute	AugmentAndRebuild	ESQL Module	CICSItemOrder_AugmentAndRebuild
MQOutput	WMB.WMB.CBE.LOG	Queue Name	WMB.WMB.CBE.LOG
MQOutput	WMB.WMB.CBE.WBM	Queue Name	WMB.WMB.CBE.WBM
MQOutput	WMB.REORDER.FAIL	Queue Name	WMB.REORDER.FAIL

Figure 6-36 Node property changes

Figure 6-37 shows the completed msgflow.

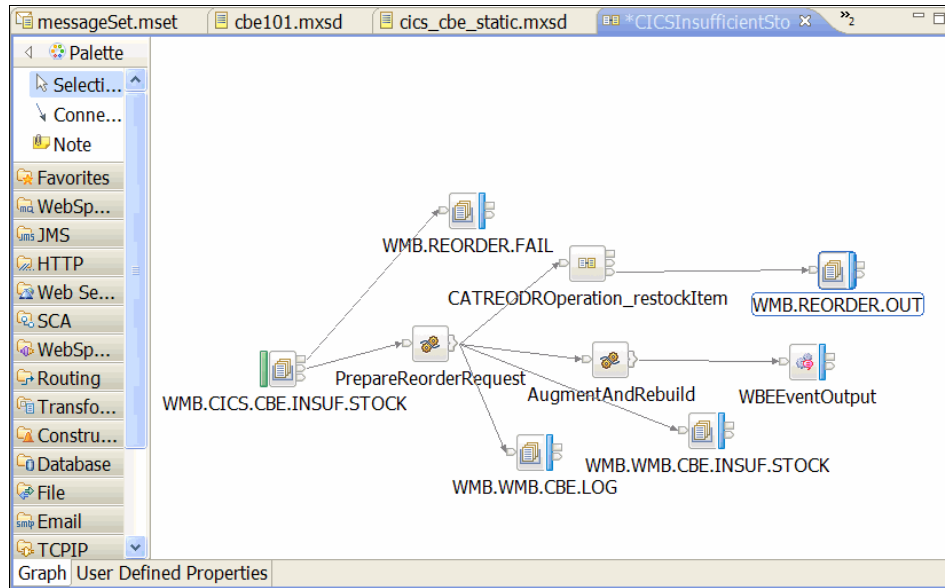


Figure 6-37 Completed msgflow

Deploying the solution

Configuring the flow created a bar file containing the newly created artifacts, which was then deployed to the TESTWBE execution group running under the QM70BRK broker. We verify the success of the deployment by checking for any deployment time errors and by querying the broker run time using the **mqsilist QM70BRK -e TESTWBE** command to verify that the new flow and message sets are running (Example 6-9).

Example 6-9 MQSILIST command output

```
C:\Program Files\IBM\MQSI\7.0>mqsilist QM70BRK -e TESTWBE
BIP1288I: Message flow 'CICSItemOrder' on execution group 'TESTWBE' is
running.
BIP1288I: Message flow 'CICSInsufficientStockReOrder' on execution
group 'TESTWBE' is running.
BIP1290I: File 'cbe2MsgSet.xsdzip' is deployed to execution group
'TESTWBE'.
BIP1290I: File 'cbeMsgSet.xsdzip' is deployed to execution group
'TESTWBE'.
BIP1290I: File 'inquireSingleMsgSet.xsdzip' is deployed to execution
group 'TESTWBE'.
```

BIP1290I: File 'reorderStockMsgSet.xsdzip' is deployed to execution group 'TESTWBE'.
BIP8071I: Successful command completion.

Verifying the solution with the debugger

Now that the message flow is running, we test it by sending a CICSFailedOrder event to the message flow input queue and tracking its progress using the message flow debugger. We placed breakpoints among all nodes to intercept the message at each step to examine its contents.

Figure 6-38 shows the message after it was received.

Name	Value
MQMD	
XMLNSC	
XmlDeclaration	
Version	1.0
CommonBaseEvent	
cbe	http://www.ibm.com/AC/commonbaseevent1_0_1
xsi	http://www.w3.org/2001/XMLSchema-instance
version	1.0.1
creationTime	2010-02-23T20:57:44.843+00:00
sourceComponentId	
component	IBM CICS TS#4.1.0
componentIdType	ProductName
executionEnvironment	IBM z/OS
instanceId	USIBMSC.EPRED3
location	SC66
locationType	Hostname
subComponent	CICS EP
componentType	http://www.ibm.com/xmlns/prod/cics/eventprocessing
situation	
categoryName	OtherSituation
situationType	
type	cbe:OtherSituation
reasoningScope	EXTERNAL
CICSApplicationEvent	
event	
cics	http://www.ibm.com/xmlns/prod/cics/events/CBE
context-info	
eventname	FailedOrder
usertag	v1
networkapplid	USIBMSC.EPRED3
timestamp	2010-02-23T20:57:44.843+00:00
bindingname	InsufficientStock
capturespecname	InsufficientStock
UOWid	1910E4E2C9C2D4E2C34BE3C3D7F6F0F1F495B15...
payload-data	
payload	
data	http://www.ibm.com/prod/cics/v1/FailedOrder
userid	Person01
charge_dept	Dept0005
item_ref_number	+110
quantity_req	+15

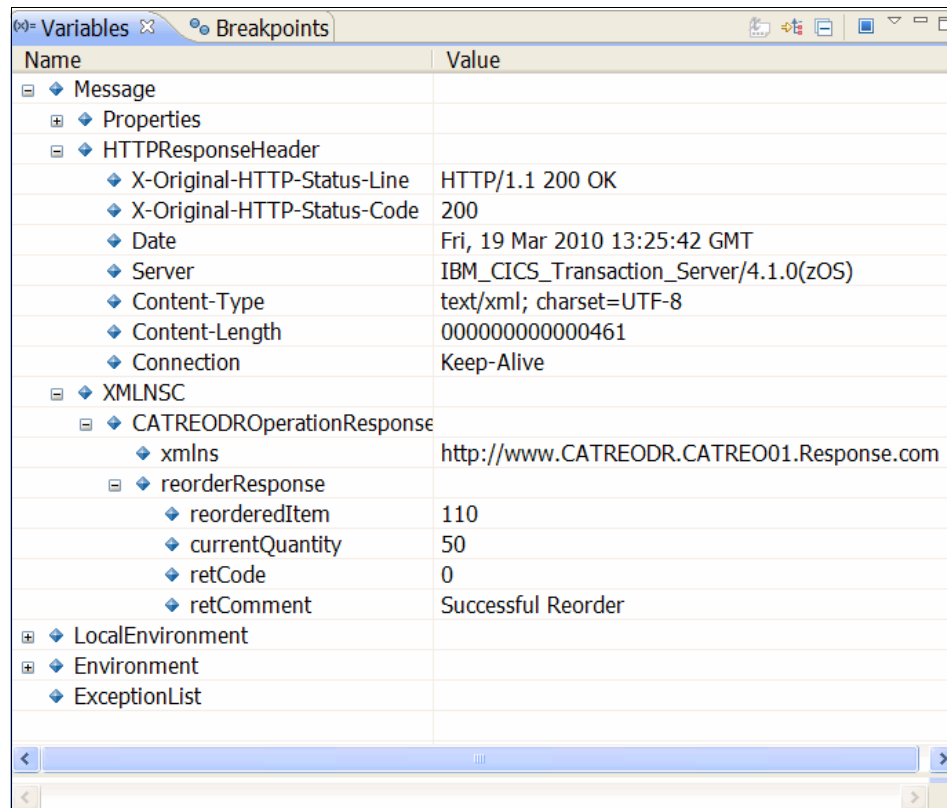
Figure 6-38 Failed order message

Figure 6-39 shows the message after the AugmentAndPrepare compute node and prior to entering the WebSphere Business Events Event Output node.

Name	Value
Message	
Properties	
MQMD	
Format	MQSTR
XMLNSC	
connector	
xmlns	http://wbe.ibm.com/6.2/Event/Event_FailedOrder-esb
name	CICS Catalog ReOrder
version	6.2
connector-bundle	
type	Event
name	Event_FailedOrder-esb
FailedOrder_Context	
Binding-user-tag	InsufficientStock
type	String
Network-UOWID	1910E4E2C9C2D4E2C34BE3C3D7F6F6F0F1F495B15...
type	String
businessevent	FailedOrder
type	String
Capture-Spec-Name	InsufficientStock
type	String
FailedOrder_Data	
userid	Person01
type	String
charge_dept	Dept0005
type	String
item_ref_number	+110
type	Real
quantity_req	+15
type	Real
LocalEnvironment	
Environment	

Figure 6-39 Message in WBE format

Figure 7-40 shows the webservice successful web service response.



Name	Value
Message	
Properties	
HTTPResponseHeader	
X-Original-HTTP-Status-Line	HTTP/1.1 200 OK
X-Original-HTTP-Status-Code	200
Date	Fri, 19 Mar 2010 13:25:42 GMT
Server	IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type	text/xml; charset=UTF-8
Content-Length	000000000000461
Connection	Keep-Alive
XMLNSC	
CATREODROperationResponse	
xmlns	http://www.CATREODR.CATREO01.Response.com
reorderResponse	
reorderedItem	110
currentQuantity	50
retCode	0
retComment	Successful Reorder
LocalEnvironment	
Environment	
ExceptionList	

Figure 6-41 Web service successful reorder response

The final verification is WebSphere Business Events successfully receiving and consuming the CICS FailedOrder events.

6.6 Scenario 4 test

The implementation of scenario 4 does not require any additional ESB function.

6.7 Problems encountered, hints, and tips

Overview

During the course of constructing and testing any message flow, problems occur that require you to perform problem determination, corrective action, and retest, especially in cases where complex messages are processed by the message flow. WebSphere Message Broker provides useful debugging tools to help isolate the issue.

Debugger

The WebSphere Message Broker Toolkit provides a Java-based debugger that enables developers to quickly and easily monitor the activity within the message flow to determine if flow logic is correct. You can set breakpoints in a flow and then step through the flow. While you are stepping through the flow, you can examine and change the message variables and the variables that are used by ESQL code, Java code, and mappings. You can debug a wide variety of error conditions in flows, such as these error conditions:

- ▶ Nodes that are wired incorrectly
- ▶ Incorrect conditional branching in transition conditions
- ▶ Unintended infinite loops in the flow

We used the debugger successfully during both scenarios to locate ESQL coding issues. Using the debugger resulted in significant time savings compared to code inspections alone or capturing and reviewing user traces for each discrepancy that was encountered.

Tracing

In certain cases, collecting a user trace provides helpful information that can be used to isolate environment setup issues, for example, if you experience issues when attempting to deploy a message flow containing a user-defined node.

For detailed information about using the debugger and collecting message flow user traces, refer to the WebSphere Message Broker V7.0 Information Center at this website:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

6.8 Summary

In this chapter, we demonstrated two simple scenarios using WebSphere Message Broker to route, transform, and enrich CICS event messages prior to delivering the messages to downstream consumers. We showed how to

configure WebSphere Message Broker to work with event messages originating in CICS that used multiple schemas and how to use WebSphere Message Broker to transform those event messages into a WebSphere Business Events specified format.

The security setup and implementation considerations were outside the scope of this book and were not included in either test scenario.

We provide specific details of the scenario implementations in the form of a project interchange file, which is available with the source materials for this IBM Redbooks publication in Appendix A, “Additional material” on page 401.



DataPower business scenario

In this chapter, we describe the following topics:

- ▶ Overview of the environment
- ▶ Setup of the environment:
 - Customer Information Control System (CICS) setup
 - DataPower setup
 - WebSphere Business Events setup
 - WebSphere Business Monitor setup
 - WebSphere Process Server setup

We show how each scenario works and how each product in our environment handles the event.

7.1 Environment overview

The DataPower SOA Appliance is a purpose-built IT device that is used to simplify the development and deployment of applications in a networked environment. The DataPower device® family (several devices in the family are shown in Figure 7-1) has a number of members that provide a broad collection of processing services and data handling capabilities. This book does not concentrate on describing all the various models and features available, but instead, it focuses on the integration capabilities of the model XI50. In particular, this book highlights the use of these capabilities to implement the enterprise service bus (ESB) architectural pattern.

For complete information about IBM WebSphere DataPower SOA Appliances, go to this website:

<http://www.ibm.com/software/integration/datapower>

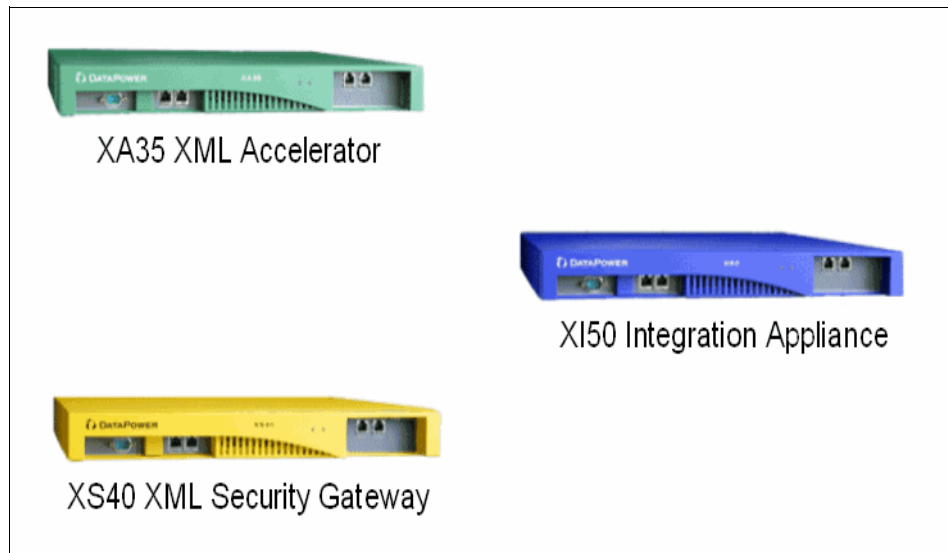


Figure 7-1 DataPower Appliances

The XI50 provides many of the primitives that are needed by an ESB, including transformation, mediation, dynamic routing, and protocol conversion. To best illustrate the use of DataPower as an ESB in our scenario, we chose the Multi-Protocol Gateway service, because it offers the largest range of capabilities that do not involve the presentation of Web services (note that the Web Service Proxy service is provided for virtualizing Web services). We describe the details of the device configuration in the following sections.

7.2 Environment configuration

Figure 7-2 shows the topology of our environment.

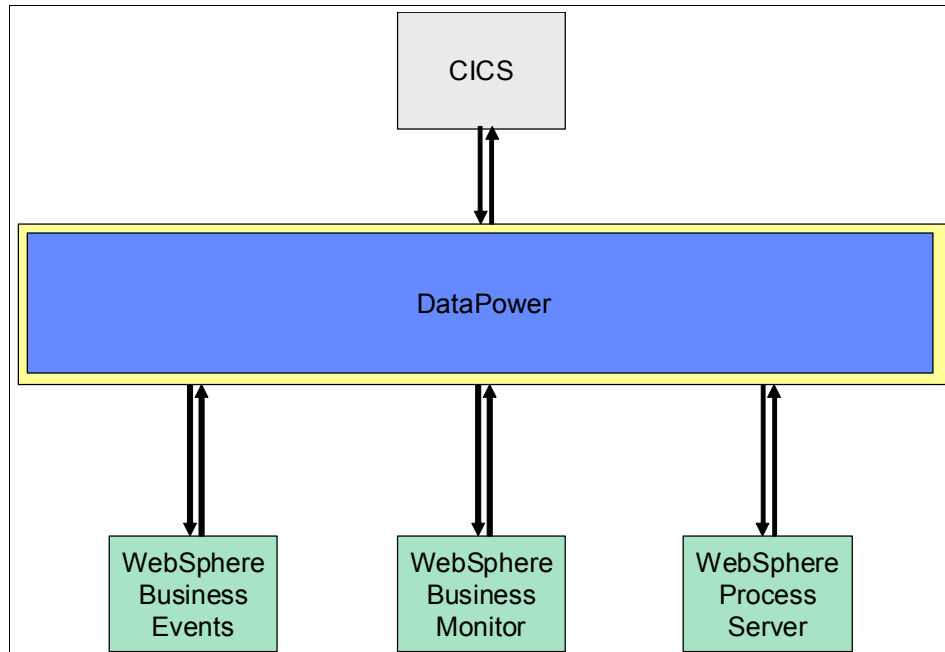


Figure 7-2 Environment topology

7.2.1 CICS configuration

Next, we describe the CICS configuration.

Creating the bundle project

To configure CICS to emit events, we first create a new bundle project in CICS Explorer. The bundle project contains the evbind files and other metadata that will be deployed into CICS.

We perform these steps to create a new bundle project in the CICS Explorer (Figure 7-3 on page 188):

1. Click Explorer on the menu bar.
2. Mouse over New Wizards.
3. Click CICS Bundle Project.

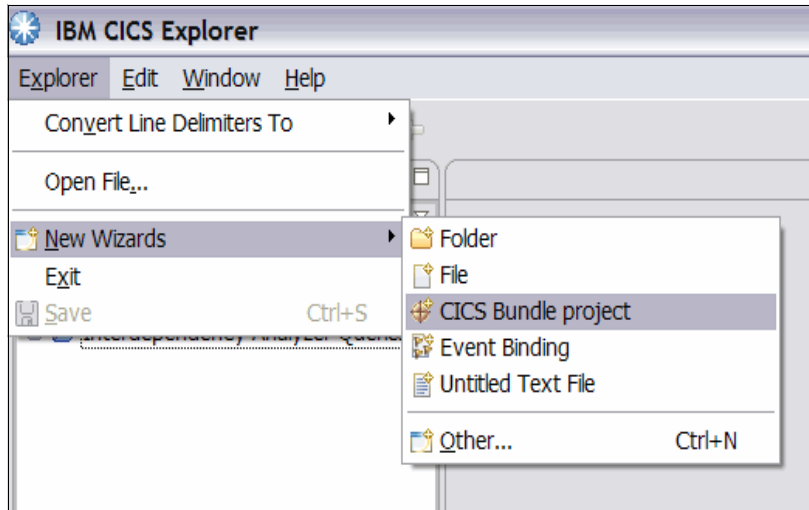


Figure 7-3 Creating the CICS bundle project

After clicking CICS Bundle Project, we are prompted to name the project. We name our project CatalogManager. After the project is named, it shows under the Project Explorer.

Creating the event binding file

After the project has been created, you can create the event binding file. The event binding is an XML definition that defines one or more business events to CICS. It consists of the event specifications, capture specifications, and event processing (EP) adapter and dispatcher information. Follow these steps to create the event binding file (Figure 7-4 on page 189):

1. Right-click the bundle project name.
2. Mouse over New.
3. Click Event Binding.

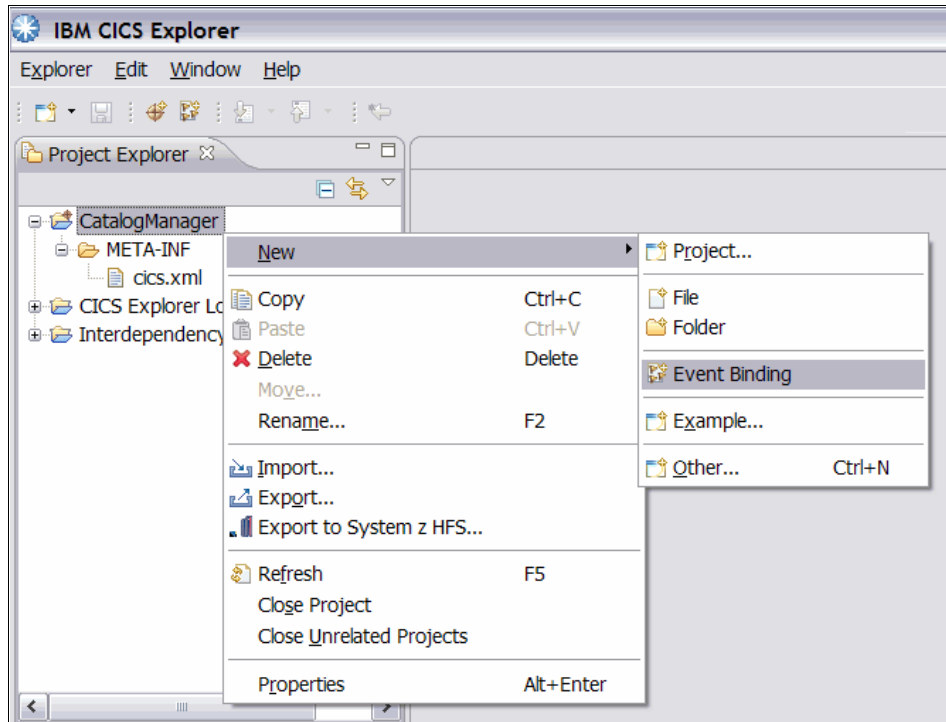


Figure 7-4 Create Event Binding

You are prompted to enter a name for the event binding file.

In our scenario, we named our first event binding file `SuccessfulOrder` and the second event binding file `InsufficientStock`. We chose these names because we capture two events in CICS, and we want the event binding file names to represent the events.

Creating the event specification

After creating the event binding file, the event binding tab editor is displayed. On this window, you can add the event specifications. An *event specification* describes an event and its processing. In the Event Binding tab, click Add to create an event specification (Figure 7-5 on page 190).

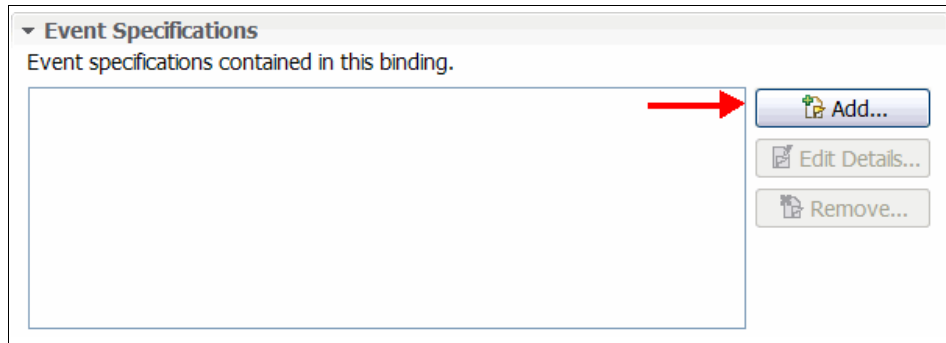


Figure 7-5 Creating an event specification

For our successfulOrder event, we create a specification called ItemOrder.

After creating the specification, we select it in the table and click Edit Details. This option opens the specification tab.

Data to be emitted

We start by defining the information that we want CICS to emit when the event is triggered. For our scenario, we emit four pieces of data when the event is triggered. The fields that we emit are described in the copybook DFH0XCP1. See Example 7-1 on page 194 for the copybook layout. We add the following fields to the emitted event:

- ▶ userid (text field)
- ▶ charge_dept (text field)
- ▶ item_ref_number (numeric field)
- ▶ quantity_req (numeric field)

At this time, we do not need to specify a length or a precision value. Figure 7-6 shows the emitted business information section.

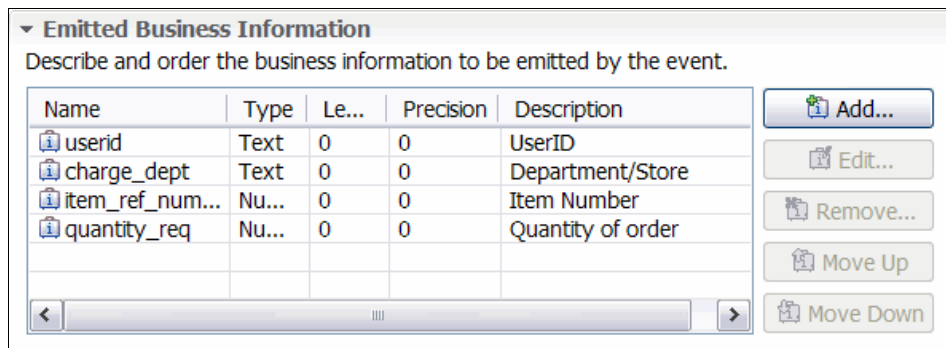


Figure 7-6 Emitted Business Information

Adding a capture specification

Now that you have defined what to emit, you need to set up when to trigger the event. You can define when to trigger the event by adding a capture specification to the event binding.

For this scenario, we name our capture specification `OrderSuccess`, because when an item is ordered successfully, we want to trigger an event.

You can add a capture specification by clicking **Add a Capture Specification** (Figure 7-7).



Figure 7-7 Creating a capture specification

When the capture specification is selected, you see three additional tabs at the top of the editor window, when you have the Specification editor tab open. Using these three additional tabs at the top, you can configure the conditions under which to trigger the event (Figure 7-8 on page 192).

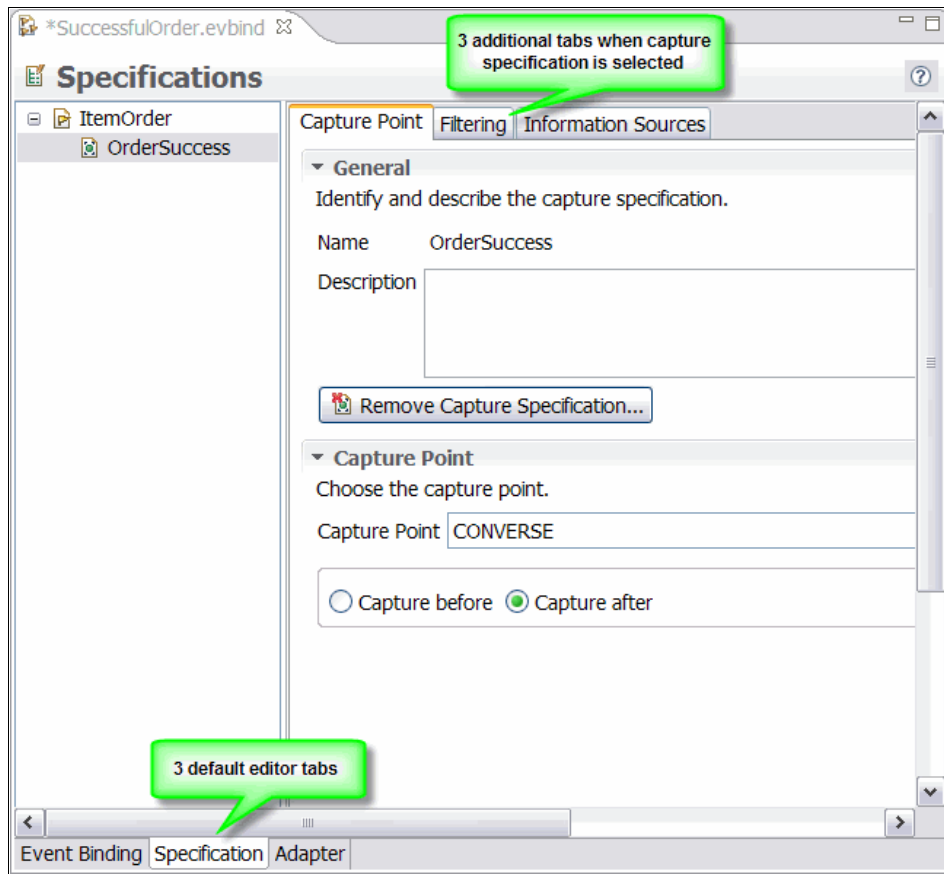


Figure 7-8 Tab layout

When to trigger the event

After creating the capture specification, consider when to trigger the event. You can use the EXEC CICS command as the capture point, which can be selected by clicking the drop-down menu next to Capture Point. In our scenario, we trigger the event on the EXEC CICS LINK PROGRAM command.

Capture after: We use the “Capture after” option, which allows us to filter on the return code.

Creating event filters

After selecting a capture point, go to the second tab to set up filters.

In our scenario, we set the Operator for the response code to Equals. We also set up an event to be triggered only when the link is to DFH0XVDS. We need to define predicates for application data, because we want to capture only successful orders. Breaking down the COMMAREA, we know that the order is successful when the first six bytes are 01ORDR and the next two bytes are 00. We use this application data as a filter. We click Add. A new window titled Application Data Predicate opens. We fill in the operator and value fields, as shown in Figure 7-9, and we click “Select from imported language structure”.

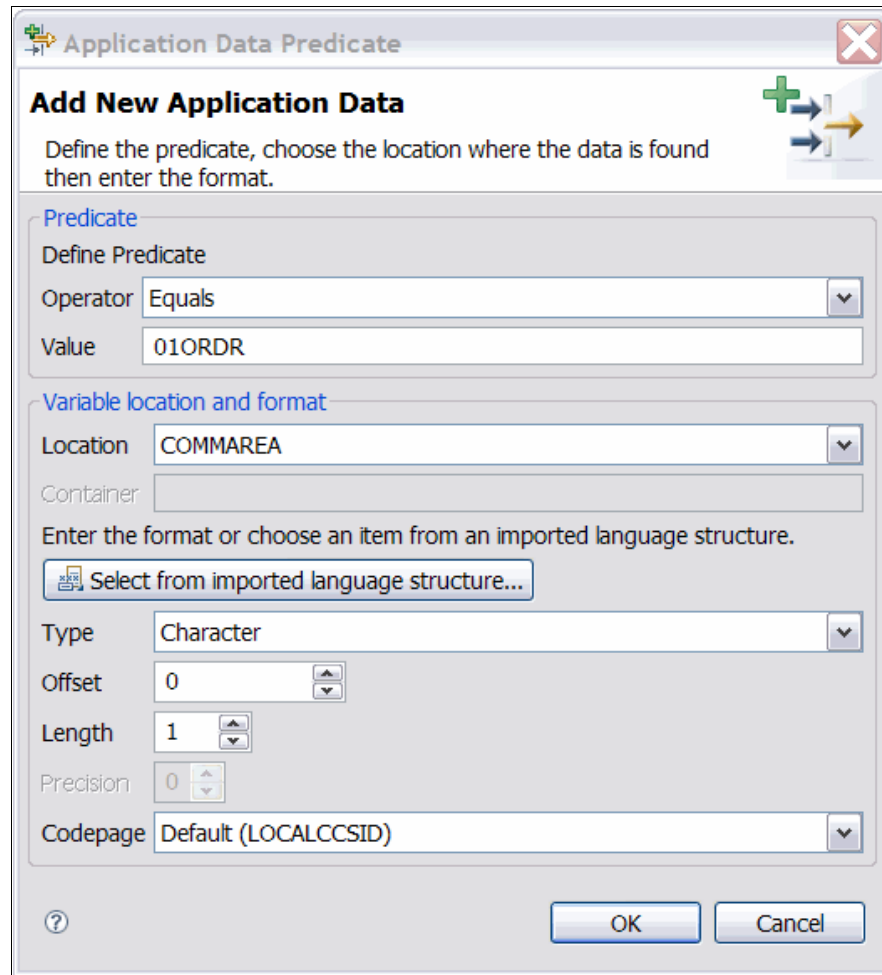


Figure 7-9 Application Data Predicate

We export a COBOL copybook named DFH0XCP1 from the enterprise server to our desktop. We use the copybook as input to the CICS Explorer Event Binding

editor so that we can map the COMMAREA accurately. Example 7-1 shows the copybook that we imported.

Example 7-1 Portion of DFH0XCP1

```

03 CA-REQUEST-ID          PIC X(6) .
03 CA-RETURN-CODE         PIC 9(2) .
03 CA-RESPONSE-MESSAGE   PIC X(79) .
03 CA-ORDER-REQUEST .
    05 CA-USERID          PIC X(8) .
    05 CA-CHARGE-DEPT    PIC X(8) .
    05 CA-ITEM-REF-NUMBER PIC 9(4) .
    05 CA-QUANTITY-REQ   PIC 9(3) .
    05 FILLER             PIC X(888) .

```

After the CICS Explorer parses the copybook, we are presented with a window (Figure 7-10). The `ca_request_id` field must have 010RDR to signify an order. We select the row for `ca_request_id`, and we click OK, which returns us to the Application Data Predicate window. Notice that the type, offset, and length fields are updated by the CICS Explorer based on the values in the copybook.

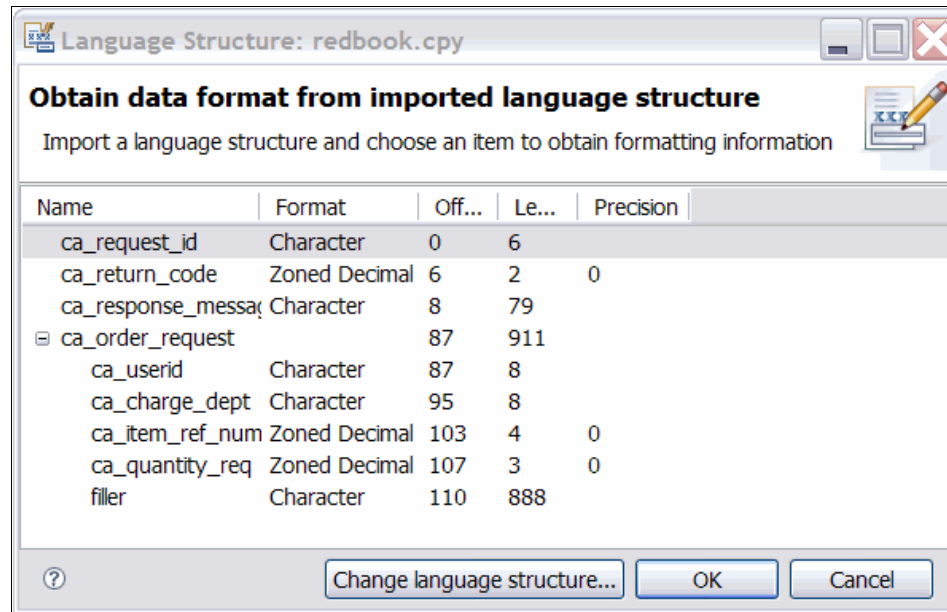


Figure 7-10 Language Structure input

After creating a filter on 01ORDER, we create an additional filter on the ca_return_code field in our copybook. We take the same steps as described previously and create a filter for 00. When these two conditions are met, we know that we have a successful order and that an event is emitted. Figure 7-11 shows the Filtering tab.

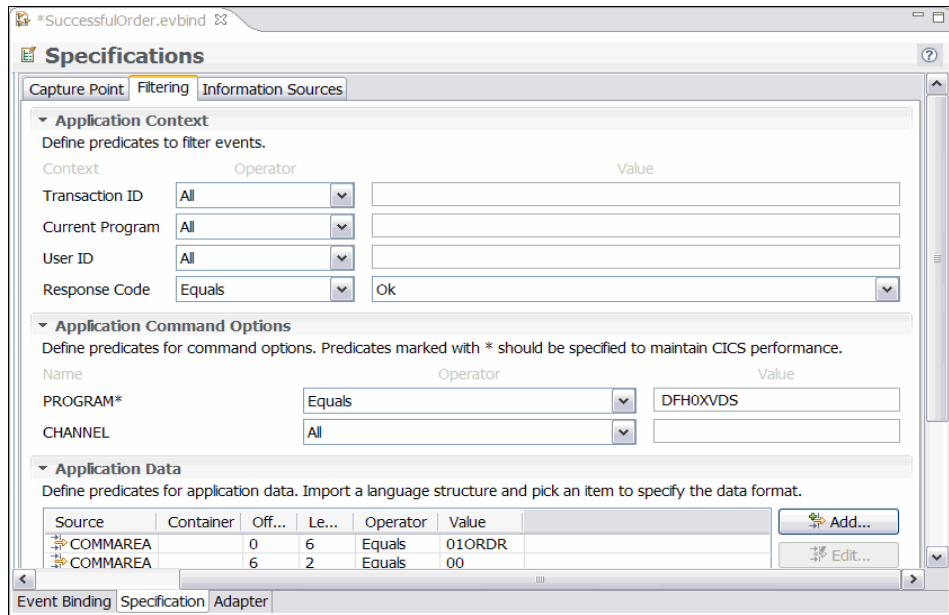


Figure 7-11 Completed Filtering view

Now that the filters are set up, in the Specification tab, we define from where the emitted business information is obtained. Under the Information Sources tab, we see the four fields that we defined earlier. We can use the copybook that we exported earlier to specify where these fields are located in the COMMAREA. We select userid, and we click Edit, which opens a window titled “Information Source for userid”. Because the emitted information is application data, we select COMMAREA under the application data tree.

On the right panel, we click “Select from imported language structure”. We perform the same steps as before when setting a filter on ca_request_id, only this time, selecting the fields that we want emitted during the event. Figure 7-12 on page 196 shows the completed information sources.

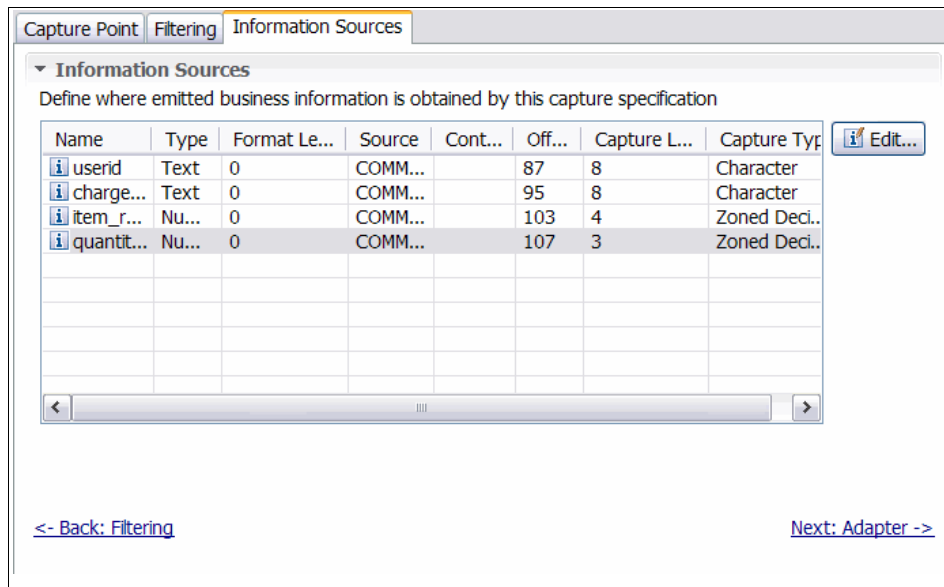


Figure 7-12 Information Sources tab

Event adapter

The last step in creating the evbind file is to choose the EP adapter through which to emit the events. For our scenario, we use a WebSphere MQ Queue adapter that allows the events to be emitted to a queue that is used as input into DataPower. In the editor, we select WMQ Queue as the adapter. For the queue name, we specify DP.CICS.CBE.ORDER. For our scenario, we set the data format to Common Base Event (CBE) (XML).

Exporting schemas

Now that the event binding file is configured to capture the successful order event, we click Export Event Specifications. This option creates an xsd schema file that describes the format of the payload of the event that will be emitted by CICS. During the DataPower configuration, you import this schema into the tooling. Because we are using a CBE-formatted event, there are two additional schemas that are needed to describe the entire message. The schema generated by the explorer is considered the payload or the dynamic portion of the event. The other portion of the event is the static portion. The static XML schema ships with CICS at

`/usr/lpp/cicsts/cicsts41/schemas/eventprocessing/cics_cbe_static.xsd.`

The last schema needed describes the entire CBE envelope within which the event will be included.

You can obtain this schema at this website:

http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/CommonBaseEvent_SituationData_V1.0.1.pdf

We use these three schema definitions later as input to the DataPower tools.

Insufficient stock event

The only differences between the successful order event and the insufficient stock event are the return code and the queue to which the event is written. Everything else remains the same, so we take the same steps that were described previously to create a second event. We need to make two changes:

- ▶ When setting up the filter predicates, filter on a 97 for `ca_return_code`.
- ▶ Define a separate queue for the event. We named our queue `DP.CICS.CBE.INSUF.STOCK`.

Deploying the events to CICS

Now that the event binding files have been created, the next step is to deploy these artifacts to CICS. As long as you have connected the CICS Explorer to a host system, you can perform this step in the CICS Explorer.

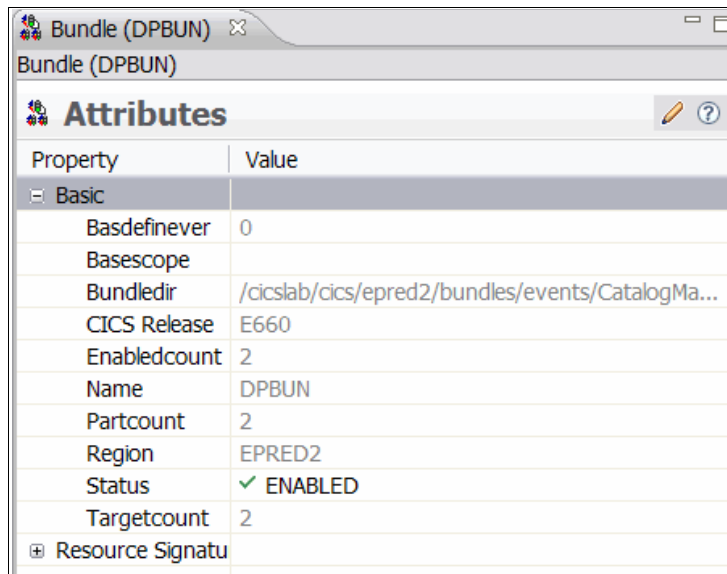
In the left pane, under the Project Explorer, we right-click the project name. We select “Export to system z HFS” and fill in the destination system detail credentials, as necessary. We determine the location on zFS to which to export the bundle. We organize zFS in a way that works for our environment, for example, production versus test, or by region. We do not use our home `/u/userid` directory. For this scenario, we use the `/cicslab/cics/epred2/bundles/events/` directory. The CICS Explorer appends the name of the project to the directory. In our case, the directory is `/cicslab/cics/epred2/bundles/events/CatalogManager`.

When you browse that directory, you see the META-INF directory and the two event binding files.

Defining the bundle resource

Now that we have exported the event binding files into the hierarchical file system (HFS), we need to create a bundle definition. If you have a CICS management client interface (CMCI) connection, you can create a bundle definition directly through the explorer. We switch to the CICS Systems Management perspective (which is part of CICS Explorer), select the CICS region in which to install the bundle, and click Administration → Bundle Definitions. When we define our bundle definition, we need to specify the bundle directory. In our scenario, the bundle directory is the `/cicslab/cics/epred2/bundles/events/CatalogManager`

directory. Figure 7-13 shows the attributes for the newly created bundle definition.



The screenshot shows a window titled 'Bundle (DPBUN)' with a sub-header 'Attributes'. Below the header is a table with two columns: 'Property' and 'Value'. The table is organized into sections: 'Basic' (expanded) and 'Resource Signatu' (collapsed). The 'Basic' section contains the following properties and values:

Property	Value
Basdefinever	0
Basescope	
Bundledir	/cicslab/cics/epred2/bundles/events/CatalogMa...
CICS Release	E660
Enabledcount	2
Name	DPBUN
Partcount	2
Region	EPRED2
Status	✓ ENABLED
Targetcount	2

Figure 7-13 Bundle Attributes

After defining the bundle definition, we install it. The events portion of the CICS configuration is complete.

7.2.2 DataPower configuration

Next, we configure DataPower.

Multi-Protocol Gateway service configuration

We use the Multi-Protocol Gateway service to provide our ESB functionality. The implementation actually employs two gateways: one gateway for general orders and another gateway to handle the special processing of failed orders (Figure 7-14 on page 199). There is a lot of flexibility allowed. Decisions regarding the number of device services to use and the details of their configurations are based on many requirements.

Multi-Protocol Gateway Name	Op-State	Logs	Type	Req-Type	Back Side URL	Resp-Type	Front Protocol
CICS-F-MPGW	up		static-backend	xml	dpmq://CICS-QM/?RequestQueue=DP.DP.WBE.ORDER	xml	CICS-F-FSH
CICS-MPGW	up		static-backend	xml	dpmq://CICS-QM/?RequestQueue=DP.DP.WBE.ORDER	xml	CICS-FSH

Figure 7-14 Multi-Protocol Gateways used in the example scenario

We configure the Multi-Protocol Gateways and related support objects that are used in our scenario by using the WebGUI browser interface. Figure 7-15 shows the major Multi-Protocol Gateway configuration page for the order service.

Configure Multi-Protocol Gateway
[Help](#)

General Advanced Stylesheet Params Headers Monitors WS-Addressing WS-ReliableMessag

Apply Cancel Delete
Export | View Log | View Status | Show Probe | Validate Confor

Multi-Protocol Gateway status: [up]

General Configuration

Multi-Protocol Gateway Name
CICS-MPGW *

Summary
|

Type
 dynamic-backend
 static-backend
 *

XML Manager
default + ... *

Multi-Protocol Gateway Policy
CICS-MPGW-POLICY + ... *

URL Rewrite Policy
(none) + ...

Back side settings

Backend URL
dpmq://CICS-QM/?RequestQueue *

Front side settings

Front Side Protocol
CICS-FSH (MQ Front Side Handler) ✕
 + ... *

Figure 7-15 Configuration page for the order service

Before we proceed with the service definitions for catalog orders and failed orders, we define several configuration objects that will be used by both of the services.

Multi-Protocol Gateway front side handler configuration

The Multi-Protocol Gateway service accepts incoming requests in a number of communication protocols. Our example uses the WebSphere MQ protocol exclusively and thus requires the definition of an MQ Front Side Handler (FSH) and an MQ Queue Manager (QM). Combined, these objects allow the service to communicate with both client-side and back-end resources.

Defining the MQ Queue Manager

An MQ FSH requires an MQ Queue Manager object to provide the location of the Queue Manager host machine. To define the MQ Queue Manager using the WebGUI, we navigate to the Objects → Network Settings → MQ Queue Manager page. We see a list of the objects that are presently defined, if any. We click Add to display a page of options for the new MQ Queue Manager. Although there are many simple and advanced configuration settings presented, the only required field is the Host Name value. Because our scenario employs the use of a Queue Manager other than the default Queue Manager, we provide a value in the Queue Manager Name field, as well. Figure 7-16 shows the relevant fields.

Configure MQ Queue Manager

Main | Advanced

MQ Queue Manager: CICS-QM [up]

Apply Cancel Delete Undo

Admin State enabled disabled

Comments

Host Name *

Queue Manager Name

Channel Name

Figure 7-16 MQ Queue Manager settings

Defining the MQ Front Side Handler

The MQ FSH contains a reference to the MQ Queue Manager and provides the names of the various queues that are used to pass messages to the service. To create a new MQ FSH, we select Objects → Protocol Handlers → MQ Front Side Handler. We click Add to create a new definition. We can configure a number of options and settings. Because our scenario does not provide response messages, we only define a value for the Get Queue field. We must also specify the MQ Queue Manager that was defined in the previous step, because it provides the network connectivity to the host computer. Figure 7-17 shows the relevant field values.

Configure MQ Front Side Handler

Main

MQ Front Side Handler: CICS-FSH [up]

Apply Cancel Delete Undo

General

Admin State enabled disabled

Comments

Queue Manager CICS-QM + ... *

Get Queue DP.CICS.CBE.ORDER *

Figure 7-17 MQ Front Side Handler settings

WebSphere Java Message Service configuration

We use the network and protocol configuration objects that have been discussed up to this point for getting event messages into the Multi-Protocol Gateway service through the MQ FSH.

There are two options for communicating with the WebSphere Business Events runtime at the back end. We can use the WebSphere Business Events Connectors, which provide a predefined communication capability for the

WebSphere Business Events runtime. However, having the DataPower device functioning as an ESB gives us another option for communicating with WebSphere Business Events. We can create a WebSphere Java Message Service (JMS) messaging object, which allows us to publish events directly to the WebSphere Business Events runtime event topic. This approach simplifies both the configuration of the WebSphere Business Events runtime environment and the deployed topology of the overall system.

Defining the WebSphere Java Message Service

We select Objects → Network Settings → WebSphere JMS and click Add to create a new definition. On the Main tab, the only non-default required field is the value for WebSphere JMS Messaging Bus. The default value for a typical WebSphere Business Events installation is WbeBus, as shown in Figure 7-18.



Figure 7-18 WebSphere JMS settings

The WebSphere JMS Endpoint tab is where we enter the configuration information pertaining to the JMS host. The values of the three fields shown, host, port, and protocol, are largely self-explanatory, but note that you can identify the target port by inspecting the symbolic name SIB_ENDPOINT_ADDRESS or SIB_ENDPOINT_SECURE_ADDRESS in the WebSphere Application Server Administration Console. Figure 7-19 shows the advanced settings.

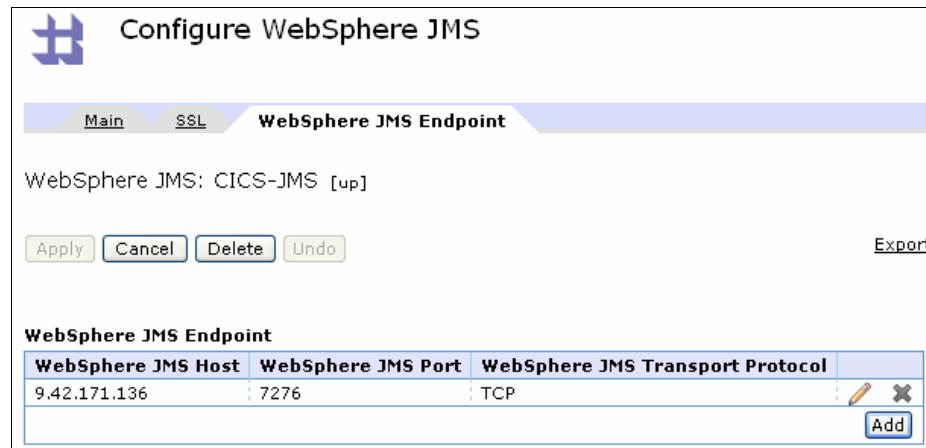


Figure 7-19 WebSphere JMS settings

Copying file artifacts to the device

The final step before we start the configuration of the Multi-Protocol Gateway service and begin constructing the document processing policy is to copy several support files to the device. We use these files by the various actions in the document processing policy to perform the validation of incoming events, the transformation from the source event format to the target event format, and for the invocation of supporting Web services. The files were created off of the device using a text editor or were emitted by the CICS tooling, as described in 7.2.1, “CICS configuration” on page 187. Table 7-1 lists the files.

Table 7-1 Files used by the Multi-Protocol Gateway services

File name	Device location	Purpose
cbe101.xsd	local://CICS/XSD	Common Base Event schema used to validate incoming Common Event Infrastructure (CEI) events
cics_cbe_static.xsd	local://CICS/XSD	CICS static event schema used to validate an extracted CICS event
FailedOrder.xsd	local://CICS/XSD	Failed order event schema used to validate extracted event payload
ItemOrder.xsd	local://CICS/XSD	Order event schema used to validate extracted event payload
ItemRestock.xsl	local://CICS/XSL	Stylesheet used to invoke the Restock Item web service
queryItem.xsl	local://CICS/XSL	Stylesheet used to invoke the Query Item web service
wbeEvent.xsl	local://CICS/XSL	Stylesheet used to create the WebSphere Business Events order event
wbeFailedOrderEvent.xsl	local://CICS/XSL	Stylesheet used to created the WebSphere Business Events failed order event

We can copy the files to the device using the WebGUI File Management interface that is shown in Figure 7-20. The source files generally reside on the host computer where the WebGUI is running. To upload the files, we select Control Panel → File Management and expand the local : folder (Figure 7-20).

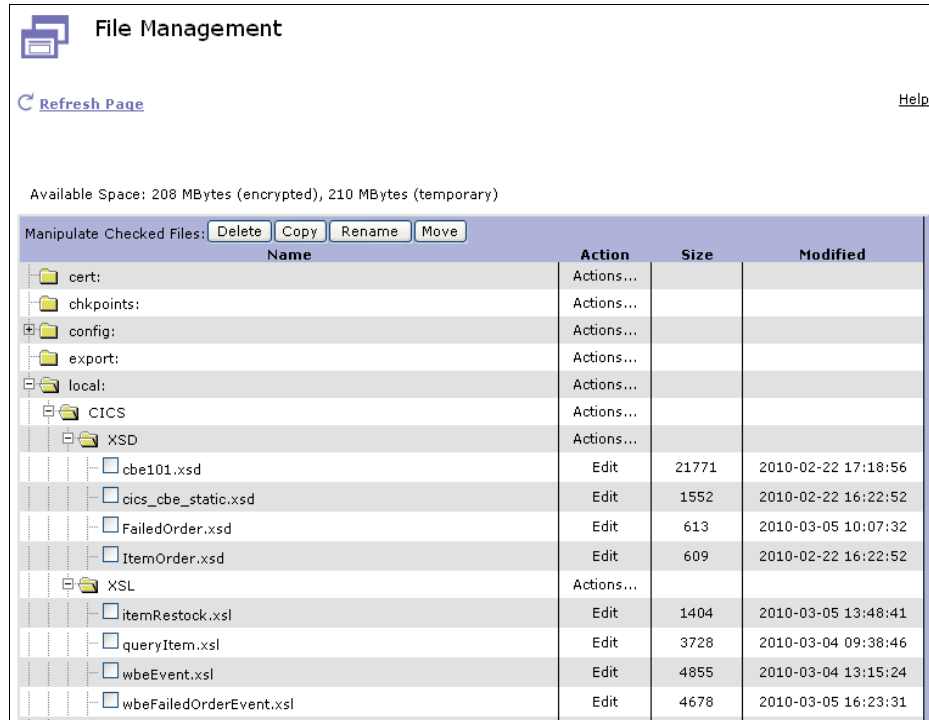


Figure 7-20 The WebGUI File Management interface

Clicking Actions for the local : folder opens the Directory Actions menu, where we can create a new subfolder and copy the necessary files.

7.2.3 Scenario 1

Next, we describe scenario 1.

Configuring the Multi-Protocol Gateway order service

With the supporting network and protocol objects defined and the file artifacts instanced on the device, we can now proceed to the creation of the actual order service gateway. To create a new Multi-Protocol Gateway service, we select Control Panel → Multi-Protocol Gateway and click Add.

General settings

We access all the configuration options that we will change from the General Settings tab of the Configure Multi-Protocol Gateway page (shown in Figure 7-15 on page 199). The input to the Multi-Protocol Gateway is a function of the Front Side Handlers that are configured for the service. Although a gateway service can have a number of Front Side Handlers spanning all the available communication protocols, for our scenario, we only need to take event messages from WebSphere MQ. We add the MQ FSH that we defined earlier. To establish a Front Side Handler, we select it from the Front Side Protocol drop-down list in the Front side settings section and click Add. The FSH is added to the list of configured handlers, as shown in Figure 7-21.



Figure 7-21 Completed Front Side Handler configuration

There are many options available in the Multi-Protocol Gateway for communicating with back-end destinations. We ultimately will send messages to a number of destinations, but the primary consumer of event messages produced by the ESB is WebSphere Business Events. To establish that connection, we designate the back-end type of this gateway as *static-back end* and establish a Backend URL to specify both the protocol and network location of the WebSphere Business Events runtime. Figure 7-22 on page 206 shows the designation of Type and Backend URL.

Type

dynamic-backend

static-backend

*

Back side settings

Backend URL

*

Figure 7-22 Multi-Protocol Gateway back-end Type and URL designations

Although the back-end value employs a URL syntax, the format of many URLs that are used in device service configurations is specific to DataPower. The Extension Elements and Functions Catalog describes all of the available formats in great detail, but we discuss the specifics of our chosen setting. Recall that we plan to publish our event messages directly to the JMS event topic of the WebSphere Business Events runtime rather than use the WebSphere Business Events JMS Connector. The following form is the general form of the DataPower URL that is used to publish our event messages directly to the JMS event topic of the WebSphere Business Events runtime:

```
dpwasjms://server-object/?RequestQueue=queue;RequestTopicSpace=topic-space
```

The protocol scheme `dpwasjms` identifies this type of URL as a JMS URL. The value of the `server-object` is the name of the WebSphere JMS network object that we defined earlier. The parameter values for `RequestQueue` and `RequestTopicSpace` must match the values that are configured on the messaging bus of the WebSphere Application Server, which is running WebSphere Business Events. By default, the WebSphere Business Events event topic name is `eventTopic` and the topic space is `WbeTopicSpace`. The value of `RequestQueue` must be qualified with the identifier `topic:` when the value represents a JMS topic. The following example is the exact URL setting:

```
dpwasjms://CICS-JMS/?RequestQueue=topic:eventTopic;RequestTopicSpace=WbeTopicSpace
```


Document processing policy definition

With the basic Multi-Protocol Gateway service level details, including input and output, established, we now focus on the heart of the service, the document processing policy. This part of message handling is also known as *multistep processing*. The general pattern is for the device to perform any required front-side processing (flow control, XML threat protection, XML parsing, and so on) and, then, for the device to begin multistep processing. After all the multistep processing actions have completed, the result is routed to the back end. We describe this message flow through the device in a general manner. We omit many details and options, especially in the area of error handling, because those details are not specifically relevant to our scenario. To create a new document processing policy, we click the plus symbol (+) next to the Multi-Protocol Gateway Policy drop-down list, as shown in Figure 7-23.

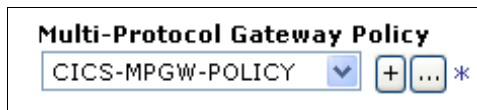


Figure 7-23 Document processing policy

Document processing policy rule editor

A DataPower document processing policy consists of a number of discrete rules that apply to message traffic flowing through a service. Rules can apply in a unidirectional fashion, pertaining either to client-to-server or server-to-client traffic, or bidirectional, applying to both client requests and server responses. The rules themselves must at a minimum have a match action, which defines the criteria that are used to select the rule for execution. Beyond the match action, rules have a variable number of processing actions, which execute against the message.

The DataPower Multi-Protocol Gateway Developers Guide, which ships with the product, contains extensive information about document processing policies, including the use of the rule editor. It does not offer information about what is possible in multistep rules. Figure 7-24 on page 208 shows our document processing policy for order events.

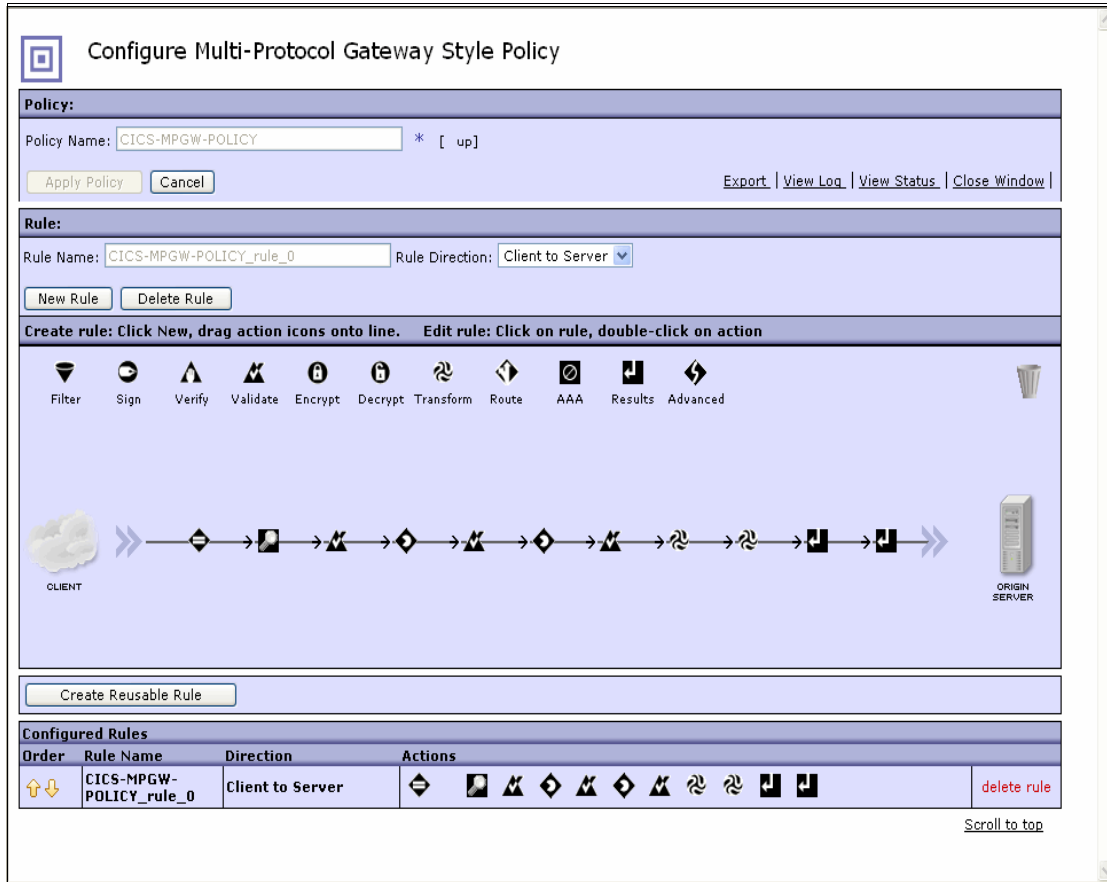


Figure 7-24 Document processing policy rule editor

Document processing policy actions

Central to the flow of data through the actions in a rule is the concept of a context. A *context* is a short-lived region of storage, which is identified by a name that contains transient data. Processing actions typically act on the data presented to them in a context and can, if necessary, create a new output context for subsequent processing actions to consume. Although this superficial overview of contexts omits many important aspects, we hope that it suffices to help you understand how the actions in this policy interact with each other. Table 7-2 on page 209 describes the actions that are used in this processing policy and details the contexts that they use.

Table 7-2 Order Multi-Protocol Gateway document processing policy actions

Action	Contexts	Purpose
Match	N/A	Select this rule for execution
Log	Input: INPUT Output: NULL	Log the incoming event
Validate	Input: INPUT Output: NULL	Validate the incoming CEI event from CICS against the CBE schema
Extract	Input: INPUT Output: event-context	Using XPATH, extract the CICS event from the CBE
Validate	Input: event-context Output: NULL	Validate the CICS event against the CICS static schema
Extract	Input: event-context Output: payload-context	Using XPATH, extract the event payload from the CICS event
Validate	Input: payload-context Output: NULL	Validate the event payload against the CICS payload schema
Transform	Input: payload-context Output: NULL	Invoke the Query Item web service to enrich the event
Transform	Input: event-context Output: wbe-event	Convert the CICS event to the WebSphere Business Events event format
Results	Input: wbe-event Output: NULL	Send the WebSphere Business Events event to the statically defined service back end
Results	Input: INPUT Output: NULL	Send the original input event to MQ for WebSphere Business Monitor

Next, we describe the configuration settings of these actions in detail.

Match action

The match action, which is also known as a *match rule*, determines whether the document processing policy rule, in which it appears, actually executes. Match rules have a number of types and options for each of those types. You can use a single match action object in multiple processing policy rules, even rules spanning services within the same application domain. We use the URL match rule type of matching rule in our scenario. This type of match checks the value of the incoming URL against a simple pattern. If a match is made, the rule executes. Figure 7-25 shows the match rule.

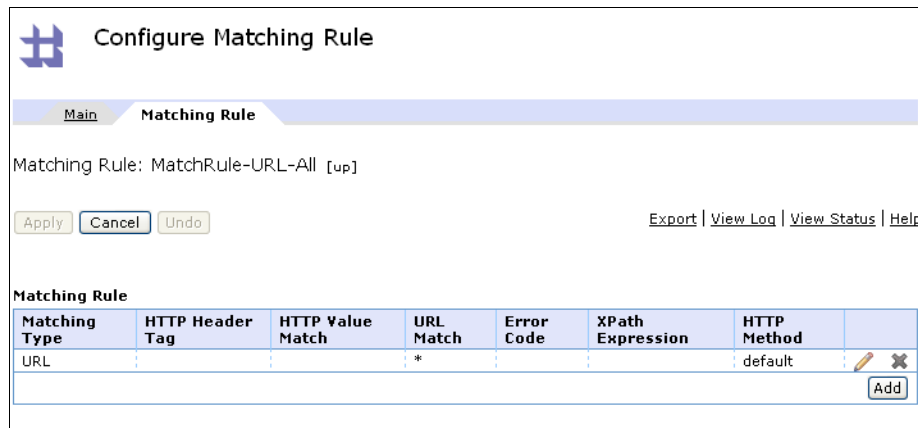


Figure 7-25 Order processing policy match rule

The URL Match pattern of an asterisk character (*) selects all traffic for processing. If we want to be more restrictive, we can use a match pattern of `dpmq://*`, which only matches traffic coming over WebSphere MQ. Still another option is to define an XPATH match rule to examine the incoming message itself. If the XPATH expression evaluated to true, the rule executes.

Log action

The first policy action in our rule is to log the incoming event message. There are several reasons to require this policy action, such as audit and debugging. Because our logging destination is a WebSphere MQ queue, we select the URL protocol scheme from the list of available logging protocols. We enter an appropriate URL value and set the Destination settings. The other options allow for flexibility in the additional metadata values that are associated with the log message. The Log Level and a user-defined Log Type can have semantic meanings at the log destination.

Figure 7-26 shows the settings for our log action. Note that the log action uses the INPUT context, which means that it sends the content of the original source event to the log destination. Because the log action does not modify the source message in any way, for efficiency, you can set the output context explicitly to NULL, or leave it blank.

Figure 7-26 Log action settings

Validate action on the incoming event

Because we characterized our incoming traffic as XML, one of the first actions taken by the Multi-Protocol Gateway service is to parse the message before invoking multistep processing. If the message fails to parse, it is rejected without any further action taken. Thus, within our document processing policy, we know then that we have a well-formed XML message.

But, before we take any further action, we want to know that the message containing the base event is valid in accordance with the Common Base Event (CBE) schema. Therefore, we use Validate Action. The schema, which will be used to perform the validation, was copied to the device during our initial setup phase. There are many Schema Validation Method options from which to

choose, but having the schema housed on the device suffices for our scenario. Figure 7-27 shows the settings for this validate action.

The screenshot shows the 'Configure Validate Action' configuration window. The 'Basic' tab is selected. The 'Input' field is set to 'INPUT'. Under the 'Options' section, the 'Validate' sub-section has five radio button options: 'Validate Document via Attribute Rewrite Rule', 'Validate Document via Schema Attribute', 'Validate Document via Schema URL' (selected), 'Validate Document via WSDL URL', and 'Validate Document with Encrypted Sections'. The 'Schema URL' field is set to 'local:///CICS/XSD'. The 'SOAP Validation' field is set to 'Body'. The 'Asynchronous' section has two radio buttons, 'on' and 'off', with 'off' selected. The 'Output' field is set to 'cbe-context'.

Figure 7-27 CBE Validate Action settings

Note that we have chosen to create a new *Output* context at this point called `cbe-context`. Other than the name, it is identical to the `INPUT` context, because the validate action acts primarily as a filter. We use this new context in subsequent steps. Having a meaningful name helps to clarify exactly what data is being processed.

Extract action to obtain the CICS static event

After passing the first validation action, we know we have a valid CBE. The CICS tooling produces a pair of schemas, which can be used to validate the static portion of an event and the dynamic payload of the event. We intend for the next few actions in the document processing policy to isolate and validate these entities. The Extract Using XPath Action allows us to select the portion of the incoming CBE that represents the CICS event. Figure 7-28 on page 213 shows the settings.

The screenshot shows the configuration interface for an 'Extract Using XPath' action. The 'Input' field is set to 'cbe-context'. Under 'Options', the 'Extract Using XPath' checkbox is checked. The XPath expression is partially entered as '/*[namespace-uri()='http://www.'. The 'XPath Tool' button is visible. The 'Variable Name' is set to 'var://'. The 'Asynchronous' option is set to 'off'. The 'Output' field is set to 'event-context'.

Figure 7-28 Extract CICS event from the CBE

The *Input* context to this action is the *Output* context of the last validate action, the *cbe-context*. The current action's *Output* will be another new context with the descriptive name *event-context*. Note that the data in this new context is a subset of the original *Input* context. The data itself is extracted using an XPATH expression intended to obtain the CICS event from the original CBE. We construct the following XPATH expression using the XPath tool:

```
/*[namespace-uri()='http://www.ibm.com/AC/commonbaseevent1_0_1' and
local-name()='CommonBaseEvent']/*[namespace-uri()='http://www.ibm.com/x
mlns/prod/cics/events/CBE' and local-name()='event']
```

The tool can build an XPATH expression from any valid sample XML document that represents the structure of an expected message.

Validating the CICS static event body

We want to validate the CICS static event against the schema emitted from the CICS tooling. We use the Validate Action option. The schema is contained in the *cics_cbe_static.xsd* file, which was uploaded to the device in a previous step.

The Input context is event-context from the previous extract action. Because there is no Output context, we specify it as NULL or leave it blank.

Extracting the event payload from the CICS static event body

This action utilizes an XPATH expression to select the portion of the CICS static event containing the payload. The Input context is event-context. The XPATH expression, which was built using the XPath tool, identifies the part of the CICS static event containing the payload:

```
/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and  
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/  
cics/events/CBE' and  
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod/  
cics/v1/ItemOrder' and local-name()='payload']
```

The Output context, which is named payload-context, holds the result.

Validating the event payload

The second schema that is produced by the CICS tooling is specific to the event payload that is carried in the CICS static event. For the Order Processing service, this schema is ItemOrder.xsd. We apply a validate action to insure that the final event payload conforms to the schema. The Input context is payload-context from the most recent extract action, and the Output is NULL.

Transform action: Enriching the input event

Up to this point, our processing policy has validated and extracted various parts of the input message. At the completion of these processing actions, we have isolated the incoming event payload in its own context named payload-context. Before transforming the CICS payload to a format that is suitable for WebSphere Business Events, we perform an enrichment step. The incoming event contains an item number and a quantity ordered. From this information, we compute the total cost of the order. However, we need to find the price for the item and then calculate the total based on the quantity of items requested. We use the transform action to execute an Extensible Stylesheet Language (XSL) stylesheet, which performs the necessary steps to enrich the event. Figure 7-29 on page 215 shows the details of the transform action configuration.

WebSphere DataPower XIS0

Configure Transform Action [Help](#)

Basic **Advanced**

Input

Input | payload-context | payload-context *

Options

Transform

Use Document Processing Instructions

- Use XSLT specified in this action on a non-XML message
- Use XSLT specified in this action
- Use XSLT specified in XML document processing instructions, if available

Processing Control File

local:///CICS/XSL

queryItem.xsl | Upload... | Fetch... | Edit... | View... | Var Builder *

URL Rewrite Policy

(none) | + | ...

Asynchronous

on off

Output

Output | NULL | NULL

Delete Done Cancel

Figure 7-29 Transform Action for query item

Example 7-2 shows the stylesheet.

Example 7-2 Enrichment stylesheet for queryItem.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:dpquery="http://www.datapower.com/param/query"
  extension-element-prefixes="dp"
  exclude-result-prefixes="dp">

  <xsl:template match="/">

    <xsl:variable name="itemNumber"
      select="/*[namespace-uri()='http://www.ibm.com/prod/cics/v1/ItemOrder'
      and
      local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
      /v1/ItemOrder' and local-name()='item_ref_number']"/>
```

```

    <dp:set-variable name="'var://context/enrichment/itemNumber'"
value="$itemNumber"/>

    <xsl:variable name="soap-request">
    <soapenv:Envelope
      xmlns:q0="http://www.DFH0XCMN.DFH0XCP4.Request.com"
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <q0:DFH0XCMNOperation>
        <q0:ca_request_id>01INQS</q0:ca_request_id>
        <q0:ca_return_code>0</q0:ca_return_code>
        <q0:ca_response_message/>
        <q0:ca_inquire_single>
          <q0:ca_item_ref_req>
            <xsl:value-of select="$itemNumber"/>
          </q0:ca_item_ref_req>
        <q0:filler1/>
        <q0:filler2/>
        <q0:ca_single_item>
          <q0:ca_sngl_item_ref>0</q0:ca_sngl_item_ref>
          <q0:ca_sngl_description/>
          <q0:ca_sngl_department>0</q0:ca_sngl_department>
          <q0:ca_sngl_cost/>
          <q0:in_sngl_stock>0</q0:in_sngl_stock>
          <q0:on_sngl_order>0</q0:on_sngl_order>
        </q0:ca_single_item>
      </q0:ca_inquire_single>
    </q0:DFH0XCMNOperation>
    </soapenv:Body>
    </soapenv:Envelope>
    </xsl:variable>

    <xsl:variable name="soap-response"

select="dp:soap-call('http://9.12.4.75:03702/exampleApp/inquireSingle',
$soap-request)"/>
    <xsl:copy-of select="$soap-response"/>

    <xsl:variable name="itemCost"
select="$soap-response/*[namespace-uri()='http://schemas.xmlsoap.org/so
ap/envelope/' and
local-name()='Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/
soap/envelope/' and

```

```

local-name()='Body']/*[namespace-uri()='http://www.DFH0XCMN.DFH0XCP4.Re
sponse.com' and
local-name()='DFH0XCMNOperationResponse']/*[namespace-uri()='http://www
.DFH0XCMN.DFH0XCP4.Response.com' and
local-name()='ca_inquire_single']/*[namespace-uri()='http://www.DFH0XCM
N.DFH0XCP4.Response.com' and
local-name()='ca_single_item']/*[namespace-uri()='http://www.DFH0XCMN.D
FH0XCP4.Response.com' and local-name()='ca_sngl_cost']"/>
  <dp:set-variable name="'var://context/enrichment/itemCost'"
value="$itemCost"/>
  <xsl:variable name="itemCostVal">
    <xsl:value-of select="$itemCost"/>
  </xsl:variable>
  <xsl:message dp:priority="debug">
    **** itemCostVal: <xsl:value-of select="$itemCostVal"/>
  </xsl:message>

  <xsl:variable name="itemQuantity"
select="/*[namespace-uri()='http://www.ibm.com/prod/cics/v1/ItemOrder'
and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/ItemOrder' and local-name()='quantity_req']"/>
  <dp:set-variable name="'var://context/enrichment/itemQuantity'"
value="$itemQuantity"/>
  <xsl:variable name="itemQuantityVal">
    <xsl:value-of select="substring-after($itemQuantity, '+')"/>
  </xsl:variable>
  <xsl:message dp:priority="debug">
    **** itemQuantityVal: <xsl:value-of select="$itemQuantityVal"/>
  </xsl:message>

  <xsl:variable name="orderTotal" select="($itemQuantityVal *
$itemCostVal)"/>
  <dp:set-variable name="'var://context/enrichment/orderTotal'"
value="$orderTotal"/>
  <xsl:message dp:priority="debug">
    <xsl:value-of select="concat('**** orderTotal: ',
$orderTotal)"/>
  </xsl:message>

</xsl:template>
</xsl:stylesheet>

```

The code that is shown in Example 7-2 on page 215 performs these steps:

- ▶ Extracts the item number from the incoming event context using XPATH
- ▶ Constructs a suitable SOAP request for invoking the Query Item web service
- ▶ Calls the Query Item web service using the `dp:soap-call()` extension function
- ▶ Retrieves the item cost from the SOAP response
- ▶ Extracts the quantity from the incoming event context
- ▶ Computes the order total
- ▶ Saves the total in the DataPower context variable that is named `var://context/enrichment/orderTotal`

The primary output of this action is the context variable, which contains the value of the order total.

Transform action: Constructing the WebSphere Business Events event

The final step in data handling is to transform the incoming CICS event to the WebSphere Business Events event format and to enrich it with the value that was computed in the prior action. All the required inputs to construct the target WebSphere Business Events event are available in the event-context and the DataPower context variables. The schema that is produced by the WebSphere Business Events Design Data tool defines the exact format of the WebSphere Business Events event. Because this step is performed using a stylesheet, we use the transform action. Figure 7-30 on page 219 shows the configuration of this transform action.

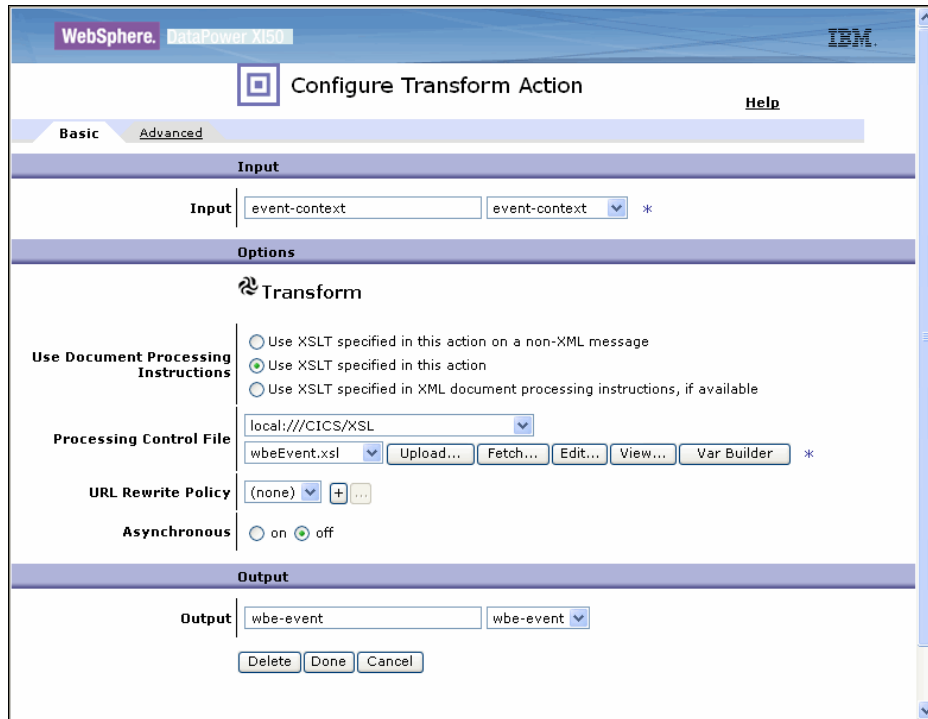


Figure 7-30 Transform action to create the WebSphere Business Events event

Example 7-3 shows the `wbeEvent.xsl` stylesheet that is configured in the action.

Example 7-3 Event generation stylesheet `wbeEvent.xsl`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:dpquery="http://www.datapower.com/param/query"
  extension-element-prefixes="dp"
  exclude-result-prefixes="dp">

  <xsl:template match="/">

    <xsl:variable name="binding-user-tag"
      select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and
      local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and
```

```

local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='bindingname']"/>
    <xsl:variable name="network-uowid"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='UOWid']"/>
    <xsl:variable name="businessevent"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='eventname']"/>
    <xsl:variable name="capture-spec-name"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='capturespecname']"/>

    <xsl:variable name="userid"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/ItemOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/ItemOrder' and local-name()='userid']"/>
    <xsl:variable name="charge_dept"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/ItemOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/ItemOrder' and local-name()='charge_dept']"/>
    <xsl:variable name="item_ref_number"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and

```

```

local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/ItemOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/ItemOrder' and local-name()='item_ref_number']"/>
    <xsl:variable name="quantity_req"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/ItemOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/ItemOrder' and local-name()='quantity_req']"/>
    <xsl:variable name="order_cost"
select="dp:variable('var://context/enrichment/orderTotal')"/>

```

```

<connector name="CICS Catalog Orders" version="6.2">
  <connector-bundle name="Event_ItemOrder-esb" type="Event">
    <ItemOrder_Context>
      <Binding-user-tag type="String">
        <xsl:value-of select="$binding-user-tag"/>
      </Binding-user-tag>
      <Network-UOWID type="String">
        <xsl:value-of select="$network-uowid"/>
      </Network-UOWID>
      <businessevent type="String">
        <xsl:value-of select="$businessevent"/>
      </businessevent>
      <Capture-Spec-Name type="String">
        <xsl:value-of select="$capture-spec-name"/>
      </Capture-Spec-Name>
    </ItemOrder_Context>
    <ItemOrder_Data>
      <userid type="String">
        <xsl:value-of select="normalize-space($userid)"/>
      </userid>
      <charge_dept type="String">
        <xsl:value-of select="normalize-space($charge_dept)"/>
      </charge_dept>
      <item_ref_number type="Real">
        <xsl:value-of select="substring-after($item_ref_number, '+')"/>
      </item_ref_number>
      <quantity_req type="Real">

```

```
        <xsl:value-of select="substring-after($quantity_req, '+')"/>
        </quantity_req>
        <order_cost type="Real">
        <xsl:value-of select="$order_cost"/>
        </order_cost>
    </ItemOrder_Data>
</connector-bundle>
</connector>

</xsl:template>

</xsl:stylesheet>
```

The WebSphere Business Events event generation stylesheet performs the following tasks:

- ▶ Extracts the payload item values that are needed from the Input context
- ▶ Generates the WebSphere Business Events event using the proper values from the Input context and the DataPower context variables

The output of this action is a new context named `wbe-event`. This context represents the event packet that will be sent to WebSphere Business Events.

Results action: Sending the WebSphere Business Events event to the configured back end

Now that all validation, enrichment, and transformation steps are complete, we send the new event packet onto WebSphere Business Events for additional complex event processing. Figure 7-31 on page 223 shows the results action that is used.

Figure 7-31 Results Action used to send an event to WebSphere Business Events

Recall that the Order Multi-Protocol Gateway service is configured to send output to the WebSphere Business Events runtime by publishing directly to its event topic. Therefore, this results action only needs to take the Input context of wbe-event. The implicit target of this action is the statically configured back end of the service. Example 7-4 shows the event that is sent to WebSphere Business Events.

Example 7-4 WebSphere Business Events event packet from DataPower

```
<connector name="CICS Catalog Orders" version="6.2">
<connector-bundle name="Event_ItemOrder-esb" type="Event">
<ItemOrder_Context>
<Binding-user-tag type="String">SuccessfulOrder</Binding-user-tag>
<Network-UOWID
type="String">170EE4E2C9C2D4E2C34BC5D7D9C5C4F2AE9FE09887EC0001000000</N
etwork-UOWID>
<businessevent type="String">ItemOrder</businessevent>
<Capture-Spec-Name type="String">OrderSuccess</Capture-Spec-Name>
</ItemOrder_Context>
```

```

<ItemOrder_Data>
<userid type="String">Person12</userid>
<charge_dept type="String">Dept0002</charge_dept>
<item_ref_number type="Real">30</item_ref_number>
<quantity_req type="Real">2</quantity_req>
<order_cost type="Real">5.8</order_cost>
</ItemOrder_Data>
</connector-bundle>
</connector>

```

Results action: Sending the CICS event to WebSphere Business Monitor

The final action in our document processing policy sends the original CBE CICS event to WebSphere Business Monitor. Because this message is the original message that was sent to the service, the Input context is INPUT. Figure 7-32 shows the settings for this action.

Figure 7-32 Results action: Send to WebSphere Business Monitor

The destination in this case is a queue on the MQ server, which WebSphere Business Monitor scans for new messages. Setting the Destination type of URL to `dpmq://` and value to `CICS-QM/?RequestQueue=DP.DP.CBE.ORDER` insures that the message is sent to the target queue that is hosted by the MQ Queue Manager.

7.2.4 Scenario 3

Next, we describe scenario 3.

Configuring the Multi-Protocol Gateway failed order service

In addition to the Multi-Protocol Gateway service for order processing, we implement a separate service to handle failed orders. The configuration details of this service are similar to the order processing gateway. An alternative to the creation of separate services is to create separate rules within a single service. The match action criteria determine the execution of the various rules. The ultimate decision about how to implement a given solution depends on many factors. Figure 7-33 on page 226 shows the document processing policy that is used for the failed order Multi-Protocol Gateway.

Configure Multi-Protocol Gateway Style Policy

Policy:
 Policy Name: * [up]
 [Export](#) | [View Log](#) | [View Status](#) | [Close Window](#)

Rule:
 Rule Name: Rule Direction:

Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Filter Sign Verify Validate Encrypt Decrypt Transform Route AAA Results Advanced

CLIENT → [Filter] → [Sign] → [Verify] → [Validate] → [Encrypt] → [Decrypt] → [Transform] → [Route] → [AAA] → [Results] → [Advanced] → ORIGIN SERVER

Configured Rules			
Order	Rule Name	Direction	Actions
↑ ↓	CICS-F-MPGW-POLICY_rule_0	Client to Server	[Filter] [Sign] [Verify] [Validate] [Encrypt] [Decrypt] [Transform] [Route] [AAA] [Results] [Advanced] <input type="button" value="delete rule"/>

[Scroll to top](#)

Figure 7-33 Document processing policy for the failed order Multi-Protocol Gateway

Table 7-3 on page 227 lists the descriptions of the various actions.

Table 7-3 Failed order Multi-Protocol Gateway document processing policy action

Action	Contexts	Purpose
Match	N/A	Select this rule for execution
Log	Input: INPUT Output: NULL	Log the incoming event
Validate	Input: INPUT Output: NULL	Validate the incoming CEI event from CICS against the CBE schema
Extract	Input: INPUT Output: event-context	Using XPATH, extract the CICS event from the CBE
Validate	Input: event-context Output: NULL	Validate the CICS event against the CICS static schema
Extract	Input: event-context Output: payload-context	Using XPATH, extract the event payload from the CICS event
Validate	Input: payload-context Output: NULL	Validate the event payload against the CICS payload schema
Transform	Input: payload-context Output: NULL	Invoke the Restock Item web service
Transform	Input: event-context Output: wbe-event	Convert the CICS event to the WebSphere Business Events event format
Results	Input: wbe-event Output: NULL	Send the WebSphere Business Events event to the statically defined service back end
Results	Input: INPUT Output: NULL	Send the original input event to MQ for WebSphere Business Monitor

The primary difference between the order gateway and the failed order service are the two transform actions, which we describe next.

Transform action: Invoke the Restock Item web service

For a failed order, we call a web service to restock the item that was unavailable. We use the extension function `dp:soap-call()`. Example 7-5 shows the stylesheet.

Example 7-5 Restock item stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:dpquery="http://www.datapower.com/param/query"
  extension-element-prefixes="dp"
  exclude-result-prefixes="dp">

  <xsl:template match="/">

    <xsl:variable name="itemNumber"
      select="/*[namespace-uri()='http://www.ibm.com/prod/cics/v1/FailedOrder'
        and
        local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
        /v1/FailedOrder' and local-name()='item_ref_number']"/>
    <dp:set-variable name="'var://context/enrichment/itemNumber'"
      value="$itemNumber"/>

    <xsl:variable name="soap-request">
      <soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:q0="http://www.CATREODR.CATRE001.Request.com"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
          <q0:CATREODROperation>
            <q0:reorderRequest>
              <q0:reorderItem>
                <xsl:value-of select="$itemNumber"/>
              </q0:reorderItem>
            </q0:reorderRequest>
          </q0:CATREODROperation>
        </soapenv:Body>
      </soapenv:Envelope>
    </xsl:variable>

    <xsl:variable name="soap-response">
```

```

select="dp:soap-call('http://9.12.4.75:3702/exampleApp/restockItem', $soap-request)"/>
  <xsl:copy-of select="$soap-response"/>

  </xsl:template>

</xsl:stylesheet>

```

This stylesheet obtains the item number from the Input context and builds a SOAP message to call the Restock Item web service. The call is then executed by invoking `dp:soap-call()`.

Transform action: Constructing the WebSphere Business Events failed order event

The failed order service sends an event to WebSphere Business Events that differs from the event that the order gateway sends. The event schema produced by WebSphere Business Events dictates the format of this event.

We use a stylesheet that is called `wbeFailedOrderEvent.xsl`, which is shown in Example 7-6, to create the WebSphere Business Events event from the CICS payload.

Example 7-6 Event generation stylesheet wbeFailedOrder.Event.xsl

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:dpquery="http://www.datapower.com/param/query"
  extension-element-prefixes="dp"
  exclude-result-prefixes="dp">

  <xsl:template match="/">

    <xsl:variable name="binding-user-tag"
      select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and
      local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and
      local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and local-name()='bindingname']"/>
    <xsl:variable name="network-uowid"
      select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/CBE' and

```

```

local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='U0wid']"/>
    <xsl:variable name="businessevent"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='eventname']"/>
    <xsl:variable name="capture-spec-name"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='context-info']/*[namespace-uri()='http://www.ibm.com/xmln
s/prod/cics/events/CBE' and local-name()='capturespecname']"/>

    <xsl:variable name="userid"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/FailedOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/FailedOrder' and local-name()='userid']"/>
    <xsl:variable name="charge_dept"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/FailedOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/FailedOrder' and local-name()='charge_dept']"/>
    <xsl:variable name="item_ref_number"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/FailedOrder' and

```



```

local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/FailedOrder' and local-name()='item_ref_number']"/>
  <xsl:variable name="quantity_req"
select="/*[namespace-uri()='http://www.ibm.com/xmlns/prod/cics/events/C
BE' and
local-name()='event']/*[namespace-uri()='http://www.ibm.com/xmlns/prod/
cics/events/CBE' and
local-name()='payload-data']/*[namespace-uri()='http://www.ibm.com/prod
/cics/v1/FailedOrder' and
local-name()='payload']/*[namespace-uri()='http://www.ibm.com/prod/cics
/v1/FailedOrder' and local-name()='quantity_req']"/>

```

```

<connector name="CICS Catalog Orders" version="6.2">
  <connector-bundle name="Event_FailedOrder-esb" type="Event">
    <FailedOrder_Context>
      <Binding-user-tag type="String">
        <xsl:value-of select="$binding-user-tag"/>
      </Binding-user-tag>
      <Network-UOWID type="String">
        <xsl:value-of select="$network-uowid"/>
      </Network-UOWID>
      <businessevent type="String">
        <xsl:value-of select="$businessevent"/>
      </businessevent>
      <Capture-Spec-Name type="String">
        <xsl:value-of select="$capture-spec-name"/>
      </Capture-Spec-Name>
    </FailedOrder_Context>
    <FailedOrder_Data>
      <userid type="String">
        <xsl:value-of select="normalize-space($userid)"/>
      </userid>
      <charge_dept type="String">
        <xsl:value-of select="normalize-space($charge_dept)"/>
      </charge_dept>
      <item_ref_number type="Real">
        <xsl:value-of select="substring-after($item_ref_number, '+')"/>
      </item_ref_number>
      <quantity_req type="Real">
        <xsl:value-of select="substring-after($quantity_req, '+')"/>
      </quantity_req>
    </FailedOrder_Data>
  </connector-bundle>
</connector>

```

```
</xsl:template>  
</xsl:stylesheet>
```

Results action: Send to WebSphere Business Events and WebSphere Business Monitor

Similar to the order service, the failed order Multi-Protocol Gateway sends the data in the `wbe-context` to WebSphere Business Events by executing a results action. Because the service specifies a static back end, and that location is the JMS event topic of the WebSphere Business Events runtime, no explicit Destination value is required in the first results action. The second results action contains a `dpmq://` URL identifying the Queue Manager and target queue for the event.

7.3 Hints and tips

Useful tools exist to help you.

7.3.1 Probe for debug

Besides examining the device processing logs, another powerful tool used to debug the flow of data through the device is the multistep Probe. Excellent documentation is available in the *DataPower Problem Determination Guide*, but one feature of the Probe that is often overlooked is the ability to configure triggers rather than capture all transactions. This feature can have a dramatic effect on the temporary space that is required by the Probe data, as well as allowing for a more selective view of the messages that are generated by the application.

7.3.2 External tools to help create stylesheets

Many tools are available to create stylesheets, ranging from a simple text editor to more sophisticated Integrated Development Environments (IDEs). We used IBM Rational Application Developer (RAD) for the creation of our example stylesheets. This tool contains an XPATH expression builder, as well as an intelligent editor that understands the syntax of XSL Transformation (XSLT). The free Eclipse platform upon which RAD is built also provides these same capabilities.

7.4 Summary

In this chapter, we have illustrated the use of the DataPower Multi-Protocol Gateway service to implement the ESB architectural pattern. The two scenarios that handle order processing and failed order processing are separate Multi-Protocol Gateways. Each Multi-Protocol Gateway contains connectivity configuration and document processing policies for its use case.



Scenario flow

In this chapter, we show sample output for each of our four scenarios as the message event flows from Customer Information Control System (CICS), through the enterprise service buses (ESBs), to WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server. We describe four scenarios:

- ▶ Successful order
- ▶ Multiple high value orders
- ▶ Failed order: Insufficient stock
- ▶ Multiple insufficient stock failures

8.1 Scenario 1

Our first scenario is the successful order event. Whenever a product order is placed successfully, CICS emits an event on the WebSphere MQ event processing (EP) adapter in Common Base Event (CBE) format. The ESB then performs enrichment, transformation, and routing. See Figure 8-1 for the successful order event diagram.

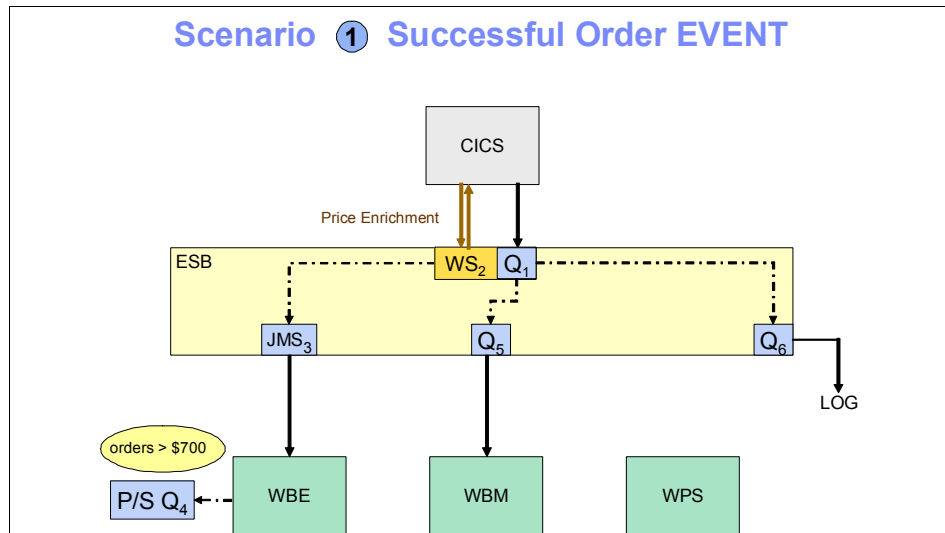


Figure 8-1 Scenario 1: Successful order event diagram

Scenario 1 consists of these steps:

1. CICS emits an event in CBE format (Example 8-1), using the WebSphere MQ (WMQ) EP adapter, to a queue that is received by the ESB, which is identified as Q_1 in the diagram (Figure 8-1).

Example 8-1 CBE format emitted from CICS for ItemOrder event

```
<?xml version="1.0"?>
<cbe:CommonBaseEvent xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_0_1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0.1"
  creationTime="2010-02-19T20:02:21.389+00:00">
  <cbe:sourceComponentId component="IBM CICS TS#4.1.0" componentIdType="ProductName"
    executionEnvironment="IBM z/OS"
    instanceId="USIBMSC.EPRED3" location="SC66" locationType="Hostname"
    subComponent="CICS EP"
    componentType="http://www.ibm.com/xmlns/prod/cics/eventprocessing"/>
  <cbe:situation categoryName="OtherSituation">
```

```

    <cbe:situationType xsi:type="cbe:OtherSituation" reasoningScope="EXTERNAL">
      <CICSApplicationEvent/>
    </cbe:situationType>
  </cbe:situation>
<cics:event xmlns:cics="http://www.ibm.com/xmlns/prod/cics/events/CBE">
  <cics:context-info>
    <cics:eventname>ItemOrder</cics:eventname>
    <cics:usertag>v1</cics:usertag>
    <cics:networkapplid>USIBMSC.EPRED3</cics:networkapplid>
    <cics:timestamp>2010-02-19T20:02:21.389+00:00</cics:timestamp>
    <cics:bindingname>SuccessfulOrder</cics:bindingname>
    <cics:capturespecname>OrderSuccess</cics:capturespecname>
    <cics:UOWid>1910E4E2C9C2D4E2C34BE3C3D7F6F6F0F1F4909D870F8CC5000100</cics:UOWid>
  </cics:context-info>
  <cics:payload-data>
    <data:payload xmlns:data="http://www.ibm.com/prod/cics/v1/ItemOrder">
      <data:userid>Person03</data:userid>
      <data:charge_dept>Dept0001</data:charge_dept>
      <data:item_ref_number>+20</data:item_ref_number>
      <data:quantity_req>+66</data:quantity_req>
    </data:payload>
  </cics:payload-data>
</cics:event>
</cbe:CommonBaseEvent>

```

Authorized program analysis report (APAR) PM03045 program temporary fix (PTF) UK54723:

In both Example 8-1 and Example 8-4, the CBE format event has plus signs (+) in front of the data for `item_ref_number` and `quantity_req`. With this fix, these + signs no longer appear, so the length of `item_ref_number` appears as 20 (because the format length that is specified in the event spec is zero, there is no padding of the numbers; otherwise, the numbers have leading zeros added to the format length). This change occurred, because certain consumers that tried to parse the XML were unable to handle the + signs.

2. The ESB receives the CBE event and then makes a web service call, `inquireSingle`, which is located in box `WS2`, to enrich the payload with the product item price. The price is in the VSAM inventory file that is available to CICS.
3. The ESB transforms the input event, changing the format from a CBE format to a WebSphere Business Events (WBE) format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, `JMS3` (Example 8-2 on page 238).

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xsi:type="wbe:connector"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wbe="http://wbe.ibm.com/6.2/Event/ItemOrder-wbe" name="CICS Catalog Orders"
  version="6.2">
  <wbe:connector-bundle name="Event_ItemOrder-esb" type="Event">
    <wbe:ItemOrder_Context>
      <wbe:Binding-user-tag type="String">SuccessfulOrder</wbe:Binding-user-tag>
      <wbe:Network-UOWID
type="String">1910E4E2C9C2D4E2C34BE3C3D7F6F6F0F1F4909D870F8CC5000100</wbe:Network-UOW
ID>
      <wbe:businessevent type="String">ItemOrder</wbe:businessevent>
      <wbe:Capture-Spec-Name type="String">OrderSuccess</wbe:Capture-Spec-Name>
    </wbe:ItemOrder_Context>
    <wbe:ItemOrder_Data>
      <wbe:userid type="String">Person03</wbe:userid>
      <wbe:charge_dept type="String">Dept0001</wbe:charge_dept>
      <wbe:item_ref_number type="Real">20.0</wbe:item_ref_number>
      <wbe:quantity_req type="Real">66.0</wbe:quantity_req>
      <wbe:order_cost type="Real">191.4</wbe:order_cost>
    </wbe:ItemOrder_Data>
  </wbe:connector-bundle>
</connector>
```

4. WebSphere Business Events receives the event, evaluates it, and if it represents an order over USD700, the information is placed on a Pub/Sub queue for interested parties through Q₄.
5. The ESB transfers the original CBE-formatted event to Q₅ for consumption by WebSphere Business Monitor. Figure 8-2 on page 239 shows the number of successful orders grouped by department.

The screenshot shows a web application interface titled "CICS Catalog Order". At the top, there is a navigation bar with links for "Home", "Go to Spaces", "Manage Spaces", and "Actions". Below the navigation bar, there is a "Page 1" dropdown menu. The main content area is titled "Instances" and contains an "Export ..." button, a search field with a magnifying glass icon, and a "Reset" button. Below the search bar is a table with two columns: "Order Count" and "Department Number". The table contains five rows of data. At the bottom right of the table, there is a pagination control showing "1 - 5" and "5".

Order Count	Department Number
78	Dept0003
84	Dept0001
99	Dept0004
76	Dept0005
88	Dept0002

Figure 8-2 WebSphere Business Monitor successful orders by department

- Finally, the ESB logs the event by placing the original CBE-formatted event on Q₆.

8.2 Scenario 2

Our second scenario is the multiple high value order event. WebSphere Business Events reviews each CICS event from scenario 1 and matches orders that are placed by the same customer over a predefined period of time. If a customer's total value exceeds a trigger value, this action indicates a high value customer. See Figure 8-3 on page 240 for the multiple high value orders event diagram.

Scenario ② Multi High Value Orders in 3 days EVENT

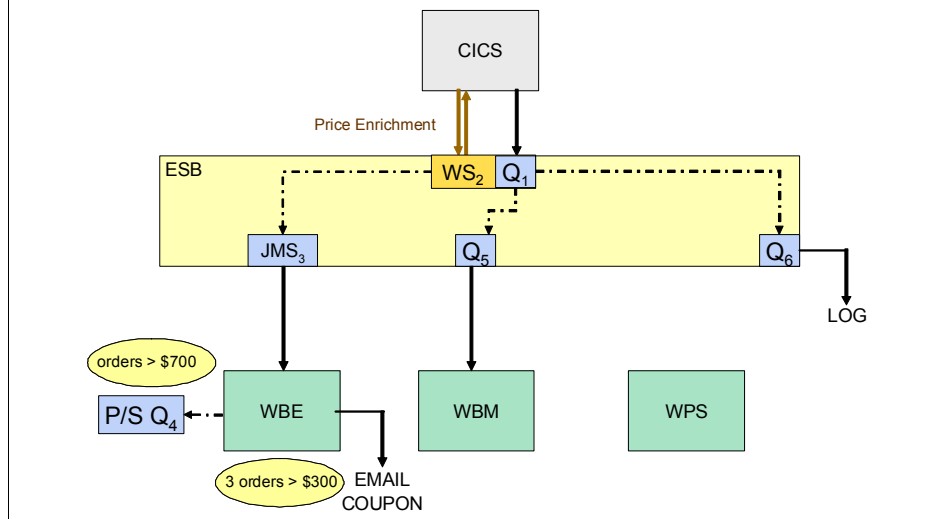


Figure 8-3 Scenario 2: Multiple high value orders event diagram

Most of the processing is the same as 8.1, “Scenario 1” on page 236, so we only show in detail the steps that differ:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter to a queue that is received by the ESB, identified as Q₁ in the diagram (Figure 8-3).
2. The ESB receives the CBE event and then makes a web service call, `inquireSingle`, which is located in box WS₂, to enrich the payload with the product item price. The price is in the VSAM inventory file that is available to CICS.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, JMS₃.
4. WebSphere Business Events receives the event, evaluates it, and if it represents an order over USD700, the information is placed on a publish/subscribe (Pub/Sub) queue for interested parties through Q₄. Example 8-3 on page 241 shows the `HighValueNotify` action event from the WebSphere Business Events application log when the action event fires.

Example 8-3 WebSphere Business Events HighValueNotify action event firing from the WebSphere Business Events application log

```
<item time='2010-03-19T12:59:16.125-0400' level='DEBUG'  
  thread='java.lang.ThreadGroup[name=DefaultWorkManager: wberuntimeear,maxpri=10] '>  
  <source>event.com.ibm.wbe.server.action.jmsactionwriter</source>  
  <event id='50E8D5C04BA800337811DF186D5B0805' name='Event_ItemOrder-esb'></event>  
  <message>  
    <![CDATA[  
      Sending Action packet:  
      <?xml version="1.0" encoding="UTF-8"?>  
      <connector name="WBE" version="6.2"  
        xmlns="http://wbe.ibm.com/6.2/Action/HighValueNotify">  
        <connector-bundle id="5468D5C0639CD0337811DF186D5B0805"  
          name="HighValueNotify" stream="DEPT0001PERSON03" type="Action"  
          workflow="FACE">  
          <HighValueNotify>  
            <item_ref_number type="Real">110</item_ref_number>  
            <Dept_UID type="String">DEPT0001PERSON03</Dept_UID>  
            <quantity_req type="Real">5</quantity_req>  
            <order_cost type="Real">847.8</order_cost>  
          </HighValueNotify>  
        </connector-bundle>  
        <system>kcgl6hk</system>  
        <timestamp>2010-03-19T12:59:16.125-04:00</timestamp>  
        <loginfo>This is an event from IBM WebSphere Business Events</loginfo>  
      </connector>  
    ]]>  
  </message>  
</item>
```

5. WebSphere Business Events reviews the order, comparing it with previous orders by the same customer, and checking to see if the total value exceeds a trigger value during a specified time window. If true, a complex scenario has occurred, and WebSphere Business Events generates an event that results in sending the customer a coupon through email. Example 8-4 on page 242 shows the OfferDiscount action event from the WebSphere Business Events application log when the action event fires.

Example 8-4 WebSphere Business Events OfferDiscount action event firing from the WebSphere Business Events application log

```
<item time='2010-03-19T13:13:38.125-0400' level='DEBUG'  
  thread='java.lang.ThreadGroup[name=DefaultWorkManager: wberuntimeear,maxpri=10] '>  
  <source>event.com.ibm.wbe.server.action.jmsactionwriter</source>  
  <event id='8278E8C22007F0337A11DF186D5B0805' name='Event_ItemOrder-esb'></event>  
  <message>  
    <![CDATA[  
      Sending Action packet:  
      <?xml version="1.0" encoding="UTF-8"?>  
      <connector name="WBE" version="6.2"  
        xmlns="http://wbe.ibm.com/6.2/Action/Offer-Discount">  
        <connector-bundle id="8468E8C22BEED0337A11DF186D5B0805"  
          name="Offer Discount" stream="DEPT0001PERSON03" type="Action"  
          workflow="FACE">  
        <Offer-Discount>  
          <Dept_UID type="String">DEPT0001PERSON03</Dept_UID>  
          <CumTotal type="Real">3052.08</CumTotal>  
          <Count type="Integer">3</Count>  
        </Offer-Discount>  
        </connector-bundle>  
        <system>kcgl6hk</system>  
        <timestamp>2010-03-19T13:13:38.109-04:00</timestamp>  
        <loginfo>This is an event from IBM WebSphere Business Events</loginfo>  
      </connector>  
    ]]>  
  </message>  
</item>
```

6. The ESB transfers the original CBE-formatted event to Q₅ for consumption by WebSphere Business Monitor.
7. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₆.

8.3 Scenario 3

Our third scenario is the failed order, or insufficient stock, event. If an order is placed and there is insufficient stock to fill the order, the Catalog Manager Example Application causes the order to fail and CICS emits an event on the WebSphere MQ (WMQ) EP adapter in CBE format. The ESB then performs enrichment, transformation, and routing. See Figure 8-4 on page 243 for the failed order - insufficient stock event diagram.

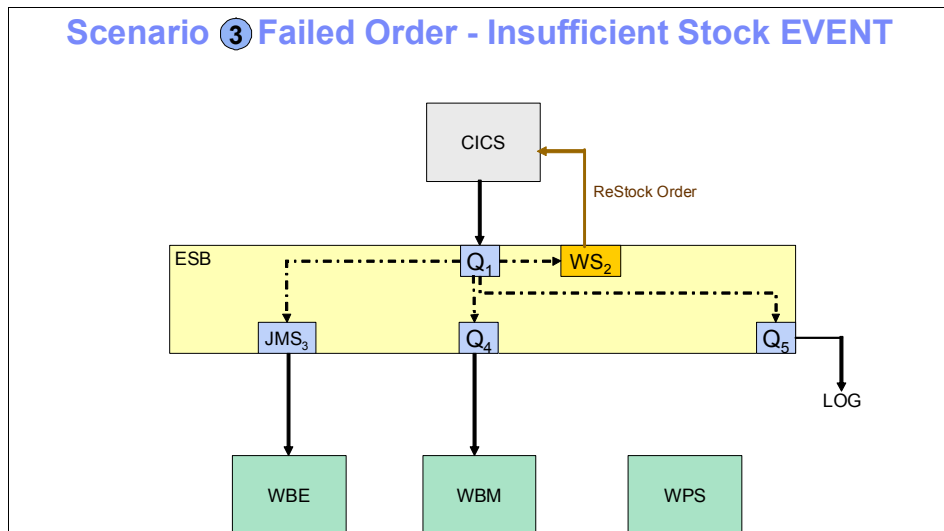


Figure 8-4 Scenario 3: Failed order - insufficient stock event diagram

This scenario consists of these steps:

1. CICS emits an event in CBE format (Example 8-5) using the WebSphere MQ (WMQ) EP adapter to a queue that is received by the ESB, identified as Q₁ in the diagram (Figure 8-4).

Example 8-5 CBE format emitted from CICS for FailedOrder event

```

<?xml version="1.0"?>
<cbe:CommonBaseEvent xmlns:cbe="http://www.ibm.com/AC/commonbaseevent1_0_1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0.1"
  creationTime="2010-02-23T20:57:44.843+00:00">
  <cbe:sourceComponentId component="IBM CICS TS#4.1.0" componentIdType="ProductName"
    executionEnvironment="IBM z/OS"
    instanceId="USIBMSC.EPRED3" location="SC66" locationType="Hostname"
    subComponent="CICS EP"
    componentType="http://www.ibm.com/xmlns/prod/cics/eventprocessing"/>
  <cbe:situation categoryName="OtherSituation">
    <cbe:situationType xsi:type="cbe:OtherSituation" reasoningScope="EXTERNAL">
      <CICSApplicationEvent/>
    </cbe:situationType>
  </cbe:situation>
  <cics:event xmlns:cics="http://www.ibm.com/xmlns/prod/cics/events/CBE">
    <cics:context-info>
      <cics:eventname>FailedOrder</cics:eventname>
      <cics:usertag>v1</cics:usertag>
    </cics:context-info>
  </cics:event>
</cbe:CommonBaseEvent>
  
```

```

<cics:networkapplid>USIBMSC.EPRED3</cics:networkapplid>
<cics:timestamp>2010-02-23T20:57:44.843+00:00</cics:timestamp>
<cics:bindingname>InsufficientStock</cics:bindingname>
<cics:capturespecname>InsufficientStock</cics:capturespecname>
<cics:UOWid>1910E4E2C9C2D4E2C34BE3C3D7F6F6F0F1F495B15E65E7D3000100</cics:UOWid>
</cics:context-info>
<cics:payload-data>
  <data:payload xmlns:data="http://www.ibm.com/prod/cics/v1/FailedOrder">
    <data:userId>Person01</data:userId>
    <data:charge_dept>Dept0005</data:charge_dept>
    <data:item_ref_number>+110</data:item_ref_number>
    <data:quantity_req>+15</data:quantity_req>
  </data:payload>
</cics:payload-data>
</cics:event>
</cbe:CommonBaseEvent>

```

2. The ESB receives the CBE event and then makes a web service call, `restockItem`, which is located in box `WS2`, to generate a reorder of the stock to meet future orders. Our simple application performs a stock order. There is no intelligence built into the application to calculate how much to order. We always order a predefined amount, such as 50 units.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, `JMS3`, as shown in Example 8-6.

Example 8-6 WBE format sent from ESB to WebSphere Business Events

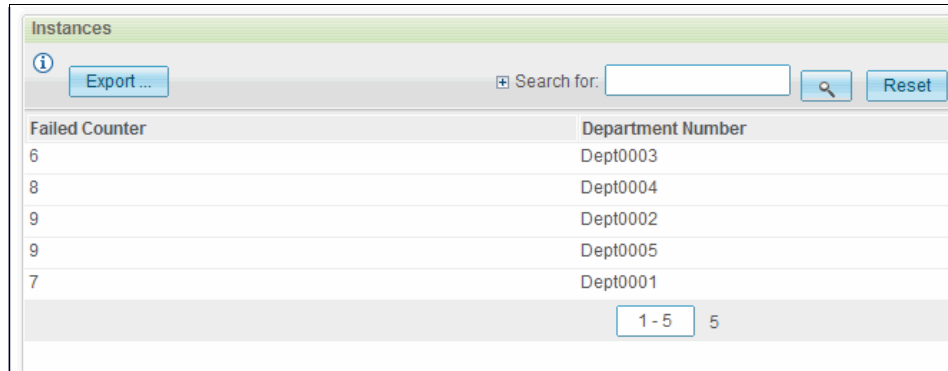
```

<?xml version="1.0" encoding="UTF-8"?>
<connector xsi:type="wbe:connector"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wbe="http://wbe.ibm.com/6.2/Event/FailedOrder-wbe" name="CICS Catalog Orders"
  version="6.2">
  <wbe:connector-bundle name="Event_FailedOrder-esb" type="Event">
    <wbe:FailedOrder_Context>
      <wbe:Binding-user-tag type="String">InsufficientStock</wbe:Binding-user-tag>
      <wbe:Network-UOWID
type="String">1910E4E2C9C2D4E2C34BE3C3D7F6F6F0F1F495B15E65E7D3000100</wbe:Network-UOW
ID>
      <wbe:businessevent type="String">FailedOrder</wbe:businessevent>
      <wbe:Capture-Spec-Name type="String">InsufficientStock</wbe:Capture-Spec-Name>
    </wbe:FailedOrder_Context>
  <wbe:FailedOrder_Data>
    <wbe:userId type="String">Person01</wbe:userId>
    <wbe:charge_dept type="String">Dept0005</wbe:charge_dept>

```

```
<wbe:item_ref_number type="Real">110.0</wbe:item_ref_number>
<wbe:quantity_req type="Real">15.0</wbe:quantity_req>
</wbe:FailedOrder_Data>
</wbe:connector-bundle>
</connector>
```

- The ESB transfers the original CBE-formatted event to Q₄ for consumption by WebSphere Business Monitor. Figure 8-5 shows the number of failed orders grouped by department.



The screenshot shows a table titled 'Instances' with two columns: 'Failed Counter' and 'Department Number'. The table contains five rows of data. At the bottom right of the table, there is a summary row showing a range of '1 - 5' and a total count of '5'. The interface includes an 'Export...' button, a search field with a magnifying glass icon, and a 'Reset' button.

Failed Counter	Department Number
6	Dept0003
8	Dept0004
9	Dept0002
9	Dept0005
7	Dept0001
1 - 5 5	

Figure 8-5 WebSphere Business Monitor failed orders by department

- Finally, the ESB logs the event by placing the original CBE-formatted event on Q₅.

8.4 Scenario 4

Our fourth scenario is the multiple insufficient stock failures event. WebSphere Business Events reviews each CICS event from scenario 3 and matches orders for the same product item number over a predefined period of time. This process indicates that there is a problem with the restock process and that we need human intervention to determine why orders continue to fail. In our example, the fix might be to increase the restock amount to avoid running out of inventory. See Figure 8-6 on page 246 for the multiple insufficient stock failures event diagram.

Scenario 4 Multi Insufficient Stock Failures EVENT

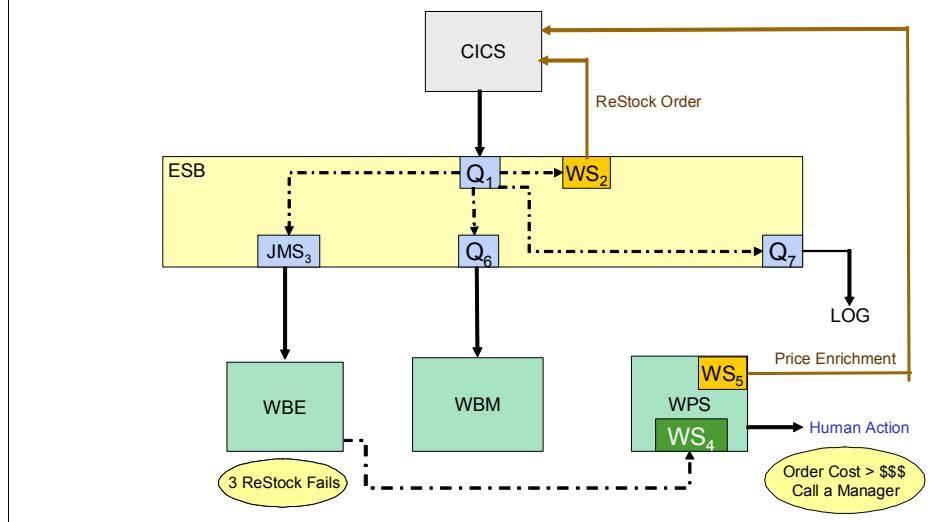


Figure 8-6 Scenario 4: Multiple insufficient stock failures event diagram

Most of the processing is the same as 8.3, “Scenario 3” on page 242, so we only show in detail the steps that differ:

1. CICS emits an event in CBE format using the WebSphere MQ (WMQ) EP adapter to a queue that is received by the ESB, which is identified as Q₁ in the diagram (Figure 8-6).
2. The ESB receives the CBE event and then makes a web service call, `restockItem`, which is located in box WS₂, to generate an order and replenish the on-hand inventory for the item that received a failed order.
3. The ESB transforms the input event, changing the format from a CBE format to a WBE format, and sends the enriched event to WebSphere Business Events runtime through a JMS topic, JMS₃.
4. WebSphere Business Events reviews the failed order, comparing it with previous failed orders by the same product item number, checking to see if the number of failures exceeds a trigger value during a specified time window. If true, a complex scenario has occurred and WebSphere Business Events sends a request to WebSphere Process Server through a web service call, which is located in box WS₄. Example 8-7 on page 247 shows the `FailedOrdersAlertNotify` action event from the WebSphere Business Events application log when the action event fires.

Example 8-7 WebSphere Business Events FailedOrdersAlertNotify action event firing from the WebSphere Business Events application log

```
<item time='2010-03-19T13:25:00.750-0400' level='DEBUG'  
  thread='java.lang.ThreadGroup[name=DefaultWorkManager: wberuntimeear,maxpri=10] '>  
  <source>event.com.ibm.wbe.server.action.jmsactionwriter</source>  
  <event id='44591458FB9C10337C11DF186D5B0805' name='Event_FailedOrder-esb'></event>  
  <message>  
    <![CDATA[  
      Sending Action packet:  
      <?xml version="1.0" encoding="UTF-8"?>  
      <connector name="WBE" version="6.2"  
        xmlns="http://wbe.ibm.com/6.2/Action/Failed-Orders-Alert-Notify">  
        <connector-bundle id="476914590EAE0337C11DF186D5B0805"  
          name="Failed Orders Alert Notify" stream="110" type="Action"  
          workflow="FACE">  
          <Failed-Orders-Alert-Notify>  
            <item_ref_number type="Real">110</item_ref_number>  
            <quantity_req type="Real">15</quantity_req>  
          </Failed-Orders-Alert-Notify>  
        </connector-bundle>  
        <system>kcgl6hk</system>  
        <timestamp>2010-03-19T13:25:00.750-04:00</timestamp>  
        <loginfo>This is an event from IBM WebSphere Business Events</loginfo>  
      </connector>  
    ]]>  
  </message>  
</item>
```

5. WebSphere Process Server makes a request that a human being get involved to review the reorder process and to make the necessary adjustments to prevent future failed orders.
Additionally, if the failed orders are over a specific value, WebSphere Process Server also makes a request to get a manager involved.
6. The ESB transfers the original CBE-formatted event to Q₆ for consumption by WebSphere Business Monitor.
7. Finally, the ESB logs the event by placing the original CBE-formatted event on Q₇.



WebSphere Business Events scenario

In this chapter, we explain the flow of the scenario that is described in Chapter 4, “Overview of the application and business scenarios” on page 57 and show the definitions in WebSphere Business Events.

9.1 Development setup and WebSphere Business Events tooling

In this section, we give an overview of our scenario, look at the tooling available, and explain the parts of the scenario that are specific to WebSphere Business Events.

9.1.1 WebSphere Business Events

IBM WebSphere Business Events is a comprehensive business event processing system. IBM WebSphere Business Events software helps businesses detect, evaluate, and respond to the effect of business events based on the discovery of actionable event patterns. WebSphere Business Events offers these functions and features:

- ▶ Improves line-of-business insight and awareness around event-driven business conditions
- ▶ Enables business users to define and manage business events that facilitate taking timely, proactive actions
- ▶ Reduces total cost of ownership (TCO) through codeless implementations, enacted by business users, often without incurring IT development or implementation costs
- ▶ Provides the ability to detect, decide, and dynamically react to simple and complex relationships between people, events, and information
- ▶ Increases business agility by enabling faster responsiveness to customers, suppliers, and changing market needs
- ▶ Reduces TCO for composite applications requiring the combination of event pattern detection, traditional workflow, and activity monitoring functionality
- ▶ Enhances existing business process management (BPM) and service-oriented architecture (SOA) infrastructures

9.1.2 WebSphere Business Events development tooling

WebSphere Business Events tooling was designed for two classes of users: the IT developer and the business user.

IT developers use the Design Data tool to create events, actions, intermediate objects, and data source definitions. The required skills are primarily technical, and these tasks require that you understand the protocols and the formats of the data. After you create these objects, you load them into a common repository.

The WebSphere Business Events Repository is part of the runtime database that is used by WebSphere Business Events.

Business users, through the design tool, retrieve objects from the repository to define the business conditions that link the events and actions. For example, if event A and event B happen within three days and this filter evaluates to true, start this specific action.

Both Design Data and Design have the option to store projects as local files, or in the hosted project store. When the project artifacts are ready to be shared among other users and made available to the WebSphere Business Events runtime, they are checked into the WebSphere Business Events repository through Design Data, or published in their entirety using Design.

9.1.3 WebSphere Business Events scenario description

We describe the business scenario that will generate the events for WebSphere Business Events in Chapter 4, “Overview of the application and business scenarios” on page 57. We divide the business scenario into four parts from the WebSphere Business Events perspective: successful order, multiple high value orders within a fixed period of time, failed order for insufficient stock, and, finally, multiple failed orders for insufficient stock. The catalog application generates two types of events to be consumed by WebSphere Business Events. The first event is the order event, and the second event is the failed order event.

9.1.4 Building the WebSphere Business Events project

First, we build the WebSphere Business Events project.

Project discussion overview

Because the primary focus of this book is to show the use of Customer Information Control System (CICS) events and the enterprise service buses (ESBs), we do not describe certain small aspects of the WebSphere Business Events project in great depth. If you are interested in learning more about WebSphere Business Events in general, refer to this website:

<http://www.ibm.com/software/integration/wbe>

A more detailed source of information about event processing with CICS and WebSphere Business Events is *Implementing Event Processing with CICS*, SG24-7792.

Creating touchpoints and event definitions with Design Data

As discussed in 9.1.2, “WebSphere Business Events development tooling” on page 250, an IT specialist uses the Design Data tool to define the artifacts, which are, in turn, used by the business user to describe an event flow. To handle the two events expected in our scenario, we add a new touchpoint and create events in it from the schemas that were exported from CICS. We define a new touchpoint by selecting Insert → Touchpoint from the Design Data menu bar. We create a touchpoint named CICS Catalog Orders. We import the two CICS schema files, ItemOrder-wbe.xsd and FailedOrder-wbe.xsd, by selecting Insert → Event → Like Schema → From File from the menu bar. Figure 9-1 shows the new touchpoint and events.

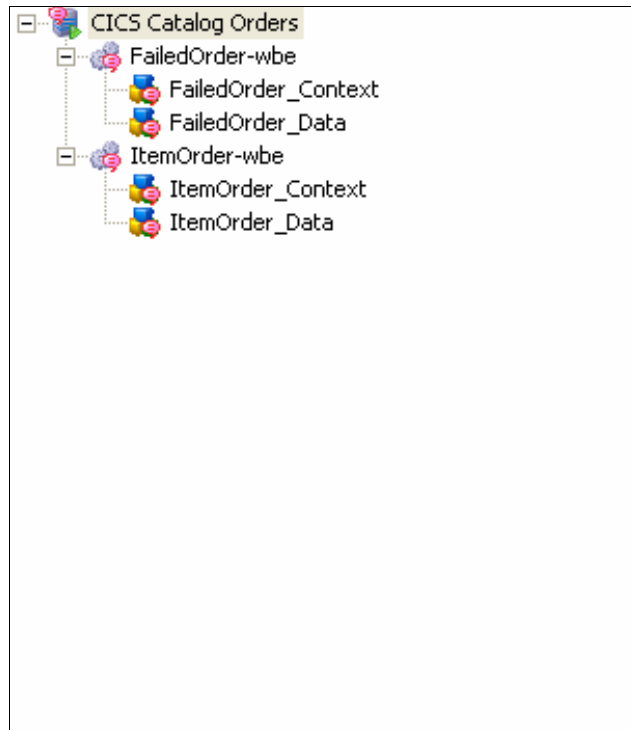


Figure 9-1 New touchpoint containing events imported from CICS

Notice that both the event and the event objects were created during the import. Example 9-1 on page 253 shows the schema emitted by CICS and imported by WebSphere Business Events for the Item Order event.

Example 9-1 ItemOrder-wbe.xsd schema

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="qualified"
    targetNamespace="http://cics.ibm.com/ItemOrder"
  xmlns:tns="http://wbe.ibm.com/6.2/Event/ItemOrder">
  <xsd:element name="ItemOrder">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
  name="ItemOrder_Context">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element minOccurs="0" maxOccurs="1" name="Binding
  user tag" type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1" name="Network
  UOWID" type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1"
  name="businessevent" type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1" name="Capture
  Spec Name" type="xsd:string" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
  name="ItemOrder_Data">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element minOccurs="0" maxOccurs="1" name="userid"
  type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1"
  name="charge_dept" type="xsd:string" />
              <xsd:element minOccurs="0" maxOccurs="1"
  name="item_ref_number" type="xsd:decimal" />
              <xsd:element minOccurs="0" maxOccurs="1"
  name="quantity_req" type="xsd:decimal" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

We now look at the event object fields that are defined in the ItemOrder_Data event object. The CICS schema defines four fields: `userid`, `charge_dept`, `item_ref_number`, and `quantity_req`. Because we want our ESB to enrich the input event with a new field, `order_cost`, we need to add that new field to the event object. Right-clicking `ItemOrder_Data` and selecting `Insert Event Object Field` open the `Insert Field` window. We name the new field `order_cost` and select a data type of `Real`. Figure 9-2 shows the `Insert Field` window.

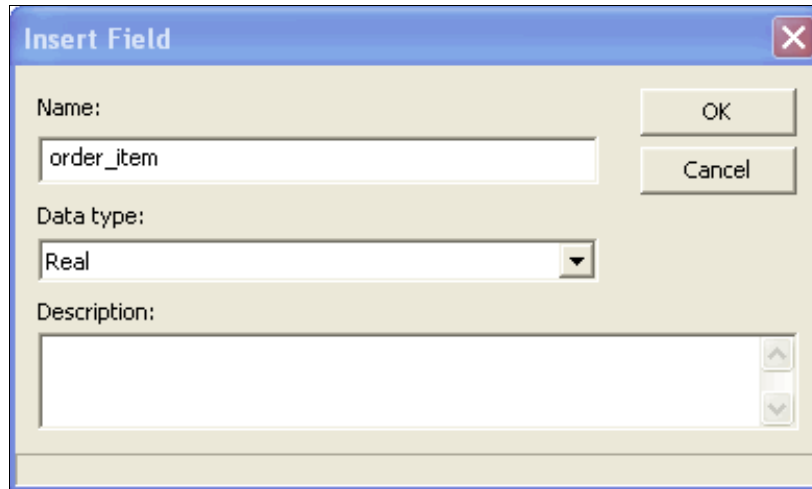


Figure 9-2 Insert the new field `order_item`

We select `OK`. Figure 9-3 on page 255 shows the updated field definitions.

The screenshot displays a configuration window for field definitions. On the left, a tree view shows four fields: 'userid', 'charge_dept', 'item_ref_number', and 'quantity_req'. The 'order_cost' field is selected, and its details are shown in a larger pane on the right. This pane is divided into two sections: 'Event object field' and 'Definition'. The 'Event object field' section contains a text box with 'order_cost', a 'Data type' dropdown menu set to 'Real', and a 'Description' text area. The 'Definition' section contains a 'Type' dropdown menu and a large 'Expression' text area.

Figure 9-3 Updated field definitions for ItemOrder_Data

The final step of the event definition process is to export the schema for the two event types: ItemOrder-wbe and FailedOrder-wbe. The ESB uses these exports to transform the Common Base Event (CBE) emitted by CICS to the format that is expected by the WebSphere Business Events runtime. We export the schema by right-clicking the event and selecting Event Properties from the context menu. On the Properties page, we select the Event tab, as shown in Figure 9-4 on page 256.

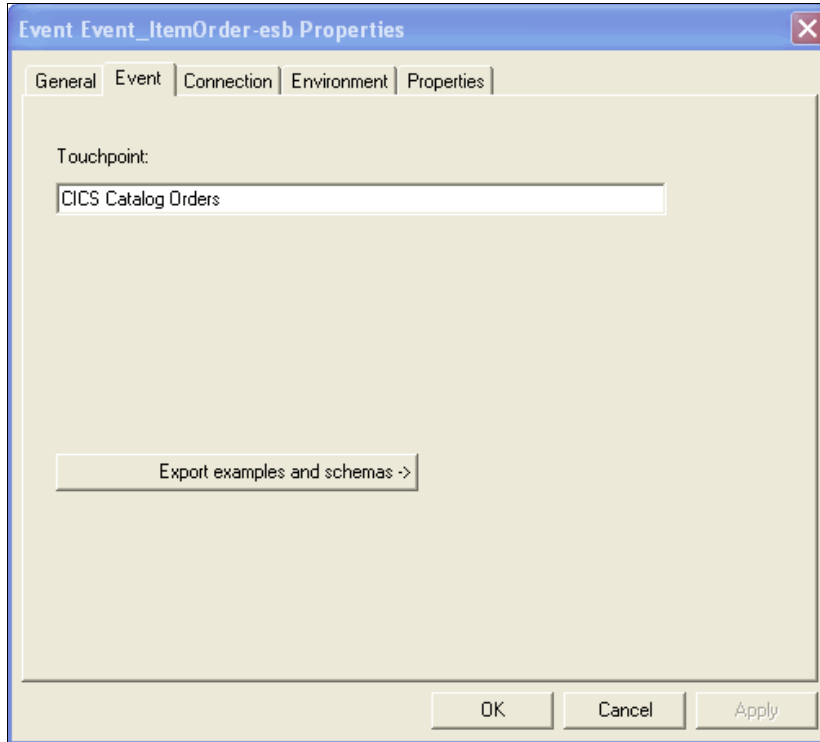


Figure 9-4 Event properties page

We then click **Export examples and schemas** → **WBE Packet Schema** → **v6.2 Format**, which opens a window containing the schema. Example 9-2 shows the schema for the item order event.

Example 9-2 *Event_ItemOrder-esb.xsd* schema

```

<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://wbe.ibm.com/6.2/Event/Event_ItemOrder-esb"
xmlns:tns="http://wbe.ibm.com/6.2/Event/Event_ItemOrder-esb">
  <annotation>
    <documentation>Changes risk being lost. Autogenerated by WebSphere
Business Events:Design Data 7.0.0 (20091124_1108) .</documentation>
  </annotation>
  <element name="connector">
    <complexType>
      <sequence>

```

```

        <element name="connector-bundle">
            <annotation>
                <documentation>Changes risk being lost. Autogenerated by
WebSphere Business Events:Design Data 7.0.0 (20091124_1108)
.</documentation>
            </annotation>
            <complexType>
                <sequence>
                    <element maxOccurs="unbounded" minOccurs="0"
name="ItemOrder_Context">
                        <complexType>
                            <sequence>
                                <element maxOccurs="1" minOccurs="0"
name="Binding-user-tag">
                                    <annotation>
                                        <documentation>(xsd:string)</documentation>
                                    </annotation>
                                    <complexType>
                                        <simpleContent>
                                            <extension base="string">
                                                <attribute fixed="String" name="type"
type="string" use="optional"/>
                                            </extension>
                                        </simpleContent>
                                    </complexType>
                                </element>
                                <element maxOccurs="1" minOccurs="0"
name="Network-UOWID">
                                    <annotation>
                                        <documentation>(xsd:string)</documentation>
                                    </annotation>
                                    <complexType>
                                        <simpleContent>
                                            <extension base="string">
                                                <attribute fixed="String" name="type"
type="string" use="optional"/>
                                            </extension>
                                        </simpleContent>
                                    </complexType>
                                </element>
                            </sequence>
                        </complexType>
                    </element>
                </sequence>
            </complexType>
        </element>

```

```

        </simpleContent>
    </complexType>
</element>
<element maxOccurs="1" minOccurs="0"
name="businessevent">
    <annotation>
        <documentation>(xsd:string)</documentation>
    </annotation>
    <complexType>
        <simpleContent>
            <extension base="string">
                <attribute fixed="String" name="type"
type="string" use="optional"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
<element maxOccurs="1" minOccurs="0"
name="Capture-Spec-Name">
    <annotation>
        <documentation>(xsd:string)</documentation>
    </annotation>
    <complexType>
        <simpleContent>
            <extension base="string">
                <attribute fixed="String" name="type"
type="string" use="optional"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
</sequence>
</complexType>
</element>
<element maxOccurs="unbounded" minOccurs="0"
name="ItemOrder_Data">
    <complexType>

```

```

<sequence>
  <element maxOccurs="1" minOccurs="0" name="userid">
    <annotation>
      <documentation>(xsd:string)</documentation>
    </annotation>
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute fixed="String" name="type"
type="string" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element maxOccurs="1" minOccurs="0"
name="charge_dept">
    <annotation>
      <documentation>(xsd:string)</documentation>
    </annotation>
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute fixed="String" name="type"
type="string" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element maxOccurs="1" minOccurs="0"
name="item_ref_number">
    <annotation>
      <documentation>(xsd:decimal)</documentation>
    </annotation>
    <complexType>
      <simpleContent>
        <extension base="double">
          <attribute fixed="Real" name="type"

```

```

type="string" use="optional"/>
    </extension>
    </simpleContent>
    </complexType>
</element>
<element maxOccurs="1" minOccurs="0"
name="quantity_req">
    <annotation>
        <documentation>(xsd:decimal)</documentation>
    </annotation>
    <complexType>
        <simpleContent>
            <extension base="double">
                <attribute fixed="Real" name="type"
type="string" use="optional"/>
                    </extension>
                    </simpleContent>
                </complexType>
            </element>
            <element maxOccurs="1" minOccurs="0"
name="order_cost">
                <annotation>

<documentation>/connector/connector-bundle/ItemOrder_Data[]/order_cost<
/documentation>

                    </annotation>
                <complexType>
                    <simpleContent>
                        <extension base="double">
                            <attribute fixed="Real" name="type"
type="string" use="optional"/>
                                </extension>
                                </simpleContent>
                            </complexType>
                        </element>
                    </sequence>
                </complexType>

```

```

        </element>
    </sequence>
    <attribute fixed="Event_ItemOrder-esb" name="name"
type="string" use="required"/>
    <attribute fixed="Event" name="type" type="string"
use="required"/>
    <attribute name="id" type="string" use="optional"/>
    <attribute name="stream" type="string" use="optional"/>
    <attribute name="workflow" type="string" use="optional"/>
</complexType>
</element>
<element minOccurs="0" name="system" type="string"/>
<element minOccurs="0" name="timestamp" type="dateTime"/>
<element minOccurs="0" name="loginfo" type="string"/>
</sequence>
<attribute name="name" type="string" use="optional"/>
<attribute name="touchpoint" type="string" use="optional"/>
<attribute fixed="6.2" name="version" type="string"
use="required"/>
</complexType>
</element>
</schema>

```

At the completion of the export process, we have the schemas for both of the WebSphere Business Events events, `Event_ItemOrder-esb.xsd` and `Event_FailedOrder-esb.xsd`, for the ESB implementation.

Developing business logic with the Design Data tool

After we have defined all the data models using Design Data, the next step is to define the business logic for the various parts of our scenario. As we mentioned in 9.1.2, “WebSphere Business Events development tooling” on page 250, we use the tool called *Design* to define the business logic for the various parts of our scenario. Starting with WebSphere Business Events Version 7.0, the Design tool is now implemented as a Business Space widget. If you have experience using Version 6.2.1 of the Design tool, you can obtain a description of the Design tool’s changes at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/Designmigration.html>

Interaction sets and filters

For both successful and failed orders, we want to send a log notification indicating the occurrence of the event; therefore, we use the interaction sets that are shown in Figure 9-5.

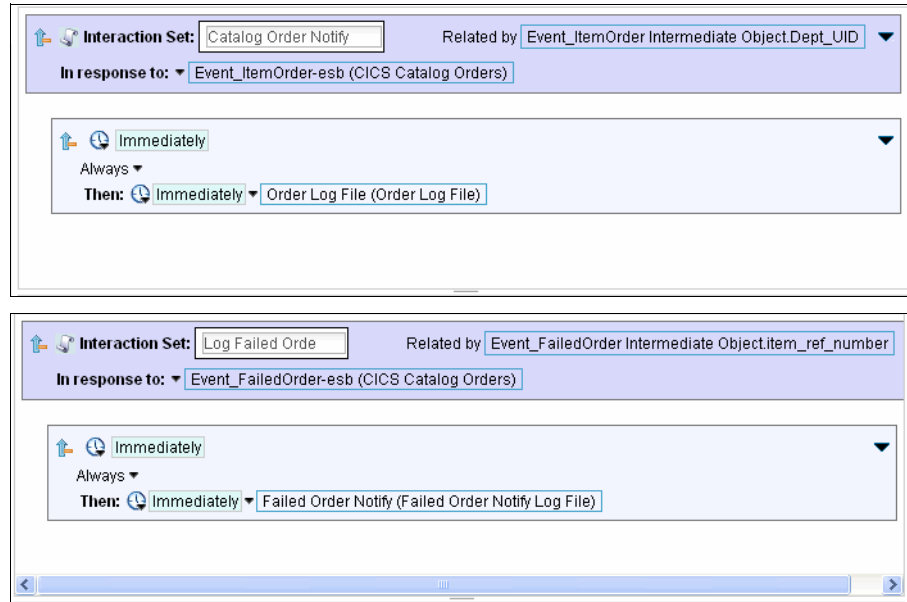


Figure 9-5 Event logging interaction sets

Each interaction block responds to an incoming event of the defined type by unconditionally firing the appropriate log action. There are no filters for these interaction blocks.

For scenario two, when the number of customer orders exceeds a certain threshold and the value of those orders is higher than a predefined amount, we want to offer that customer a discount. Figure 9-6 on page 263 shows this interaction set.

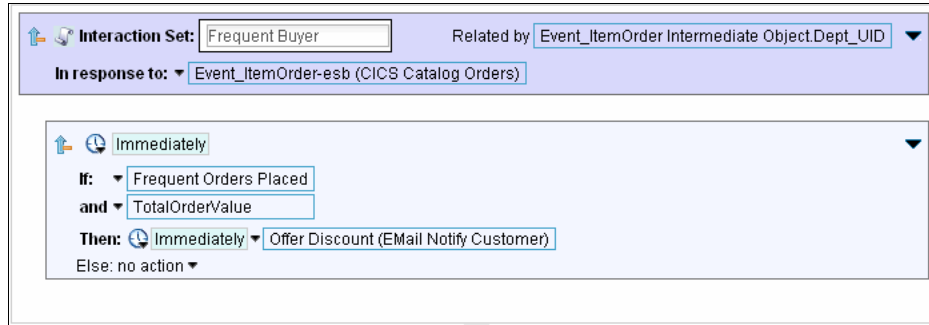


Figure 9-6 *Frequent Buyer interaction set*

Because this interaction set is related by the unique identifier that consists of the user ID and the department, WebSphere Business Events keeps track of the number of times that this event has been seen. The IT specialist who created assets in Design Data also provides a mechanism to keep the cumulative total of all orders for this user. The filter logic uses these two pieces of data to determine if the Frequent Buyer criteria are met. Figure 9-7 shows the filters.

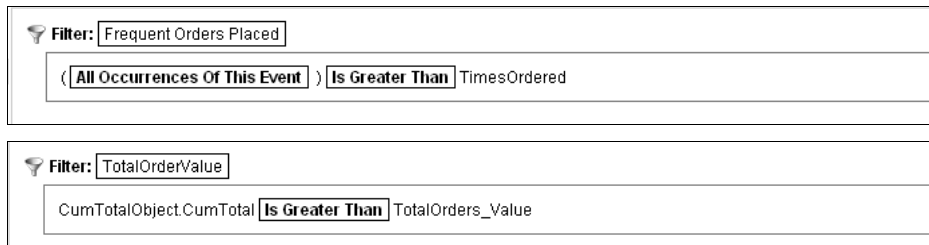


Figure 9-7 *Frequent buyer filters*

The threshold values are defined as named constants to make their meaning more obvious and to allow them to be more easily changed if necessary. When the number of occurrences of this event for this customer exceeds the frequency threshold, and the cumulative value of the orders that this customer has placed is higher than a predefined amount, the Offer Discount action fires. This action might be configured to send an email to the customer by using the WebSphere Business Events connector, or the action can be forwarded onto another component of the system (for example, back to the ESB) where the appropriate steps to fulfil the action are executed. The final step in the action processing chain is to perform pruning of the total order data being tracked for this customer. The Offer Discount action is configured to emit a result event when fired. When this result event is processed by WebSphere Business Events, logic contained in

the asset definitions for the total order value intermediate object will run and reset the total.

The fourth and final component of the scenario tracks the number of failed orders. When this number exceeds a given threshold, a Failed Orders Alert Notify action is fired by WebSphere Business Events. The purpose of this alert is to notify someone that the inventory for a particular item is being depleted too frequently and that there must be an increase in the size of the restock request. Figure 9-8 shows the interaction set, and Figure 9-9 shows the filter.

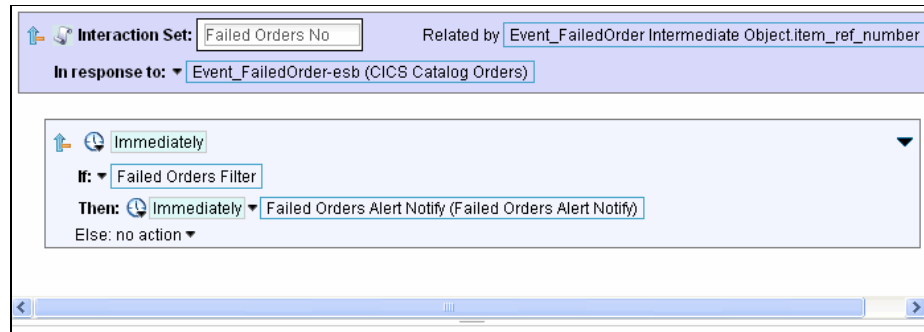


Figure 9-8 Failed Orders Notify interaction set

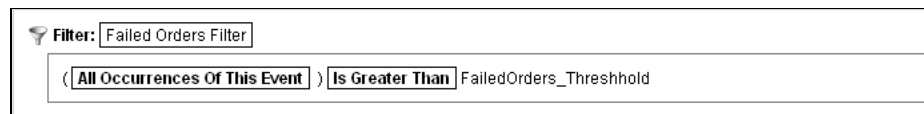


Figure 9-9 Failed Orders Filter

9.1.5 Configuring and testing WebSphere Business Events

Now, we configure and test WebSphere Business Events.

Connecting WebSphere Business Events and WebSphere Process Server

WebSphere Process Server processes the Failed Order event. In order to allow the WebSphere Business Events runtime to communicate directly with WebSphere Process Server, you must establish a connection between the two environments.

Read about the method to establish a connection between the two environments at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.integrating.doc/doc/sibx.html>

Essentially, you need to connect the service integration bus (SIBus) messaging engines in the WebSphere Application Server cells hosting WebSphere Business Events and WebSphere Process Server. The exact configuration details vary for each installation, but WebSphere Business Events provides a pair of example scripts to illustrate the required steps:

► `configure_WBE_SIB_JMS.messaging.py`

Run this script against the WebSphere Business Events installation to configure the messaging engine that is embedded in the instance of WebSphere Application Server to which the WebSphere Business Events application (`wberuntimeear`) is deployed. The script performs the following tasks:

- Creates an SIB foreign bus and a SIB link, which together with the equivalents on the WebSphere Process Server or WebSphere ESB run time, facilitate communication between the Service Component Architecture (SCA) application bus and the WebSphere Business Events SIB messaging.
- Maps the `WbeTopicSpace` defined on WebSphere Process Server or WebSphere ESB to the `WbeTopicSpace` defined on WebSphere Business Events.

► `configure_WESB_SIB_JMS.messaging.py`

Run this script against WebSphere Process Server or WebSphere ESB to configure the messaging engine that is part of the WebSphere Process Server or WebSphere ESB run time. The script performs the following tasks:

- Creates a SIB foreign bus and a SIB link, which together with the equivalents on WebSphere Business Events runtime, facilitate communication between WebSphere Business Events and the SCA application bus.
- Maps the `WbeTopicSpace` defined on WebSphere Process Server or WebSphere ESB to the `WbeTopicSpace` defined on WebSphere Business Events.
- Creates the SIB Java Message Service (JMS) topics for events, durable events, actions, and durable actions that can be used by WebSphere ESB flows to publish events into and receive actions from WebSphere Business Events.

- Creates SIB JMS activation specs for the durable and non-durable actions and a SIB JMS topic connection factory. These JMS resources are not strictly required for the integration, and you can define your own resources. However, having defined them here, it is convenient for you to have mediation flows with WebSphere Business Events interaction running without any configuration.

After the environments are connected in this fashion, actions fired by WebSphere Business Events are directly available to WebSphere Process Server without the need for the WebSphere Business Events connectors.

Using the WebSphere Business Events Business Events Tester

With Version 7 of WebSphere Business Events, you can simulate the occurrence of events and monitor the flow of actions that results from them by using the Business Events Tester.

For example, if we want to simulate the Failed Order scenario, we use the tester to generate three Event_FailedOrder-esb events that cause the Failed Orders Filter in the Failed Orders Notify interaction set to evaluate to true. The resulting notifyRepeatedFailedOrders action then fires. If WebSphere Business Events is connected to WebSphere Process Server, the action packet is delivered and processed in accordance with the flow defined by WebSphere Process Server. Figure 9-10 on page 267 shows the event selection dialog.

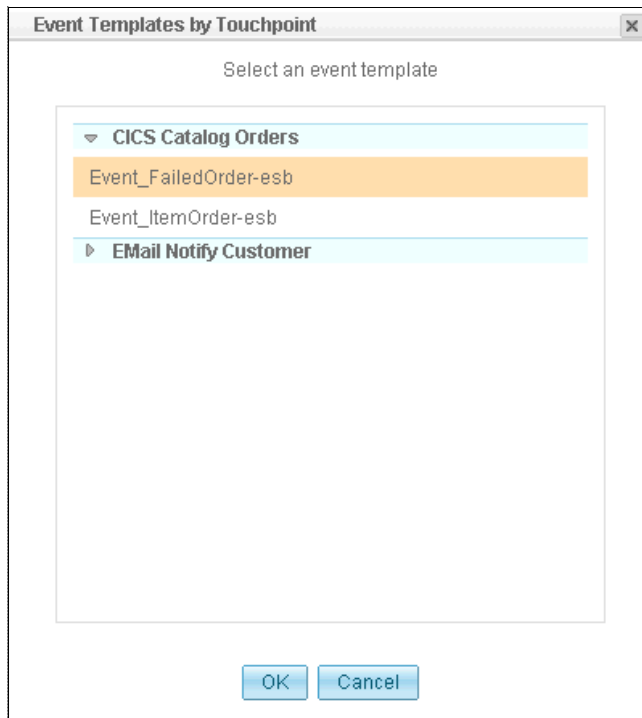


Figure 9-10 Event selection dialog

Event data is entered for the chosen event, as shown in Figure 9-11 on page 268. Selecting Send Event causes the event to be fired by the system.

Restart Testing

Send an Event Filters Actions Context Data Delayed Events and Actions

Select Event Template Send Event

Event_FailedOrder-esb (CICS Catalog Orders)

FailedOrder_Context

FailedOrder_Data

userid user1

charge_dept 100

item_ref_number 50

quantity_req 15

Figure 9-11 Sample event data

Figure 9-12 shows the sample events that were fired.

Restart Testing

Send an Event Filters Actions Context Data Delayed Events and Actions

Refresh

Type to filter on filter name Go

Filter Name	Event Name	Context Id	Event Time	Value
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:09.564	False
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:09.564	False
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:22.096	False
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:22.096	False
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:39.767	True
Failed Orders Filter	Event_FailedOrder-esb	50	Monday, June 14, 2010 08:17:39.767	True

Figure 9-12 Sample events

And finally, Figure 9-13 on page 269 shows the resulting action containing the information about the three failed orders.

Restart Testing		
Send an Event	Filters	Actions
Context Data		Delayed Events and Actions
Refresh	Type to filter on action name	
Go		
Actions		
Action Name	Action Time	Touchpoint
Failed Order Notify	Monday, June 14, 2010 08:17:09.689	Failed Order Notify Log File
Failed Order Notify	Monday, June 14, 2010 08:17:22.205	Failed Order Notify Log File
notifyRepeatedFailedOrders	Monday, June 14, 2010 08:17:39.861	Failed Orders Alert Notify
Failed Order Notify	Monday, June 14, 2010 08:17:39.861	Failed Order Notify Log File

Figure 9-13 Three failed order actions

Pruning the WebSphere Business Events context table

In all these scenarios, we rely on WebSphere Business Events to remove context information over time. The system has a global policy for removing older context data, which is described at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.admin.doc/doc/pruningthesteptable.html>

An alternate method is to keep track of more context-scoped state information and to use it in conjunction with synthetic events to maintain the desired level of data.

9.1.6 Tips and hints for developing with WebSphere Business Events

Version 7 of WebSphere Business Events has added a number of testing and debugging aids to the product to improve the application development process. You can access both the Business Events Tester and the Event Capture and Replay facilities through Business Space widgets. You can obtain a description of the Business Events Tester widget at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/testingeventlogic.html>

You can obtain a description of the Event Capture and Replay widget at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/eventcapturereplay.html>

In addition to these GUI-based tools, you can obtain information about the enhanced application logging and WebSphere Application Server trace at this website:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.admin.doc/doc/wbelogs.html>



WebSphere Business Monitor

In this chapter, we describe the inclusion of WebSphere Business Monitor in our system. With our Catalog application newly “fitted” with event bindings, and our various enterprise service buses (ESBs) consuming them and routing them to a WebSphere MQ queue on z/OS for consumption by WebSphere Business Monitor, we are ready to examine the following topics:

- ▶ WebSphere Business Monitor configuration
- ▶ Designing the monitor model
- ▶ Testing the monitor model
- ▶ Creating the Business Space dashboard
- ▶ Viewing the Customer Information Control System (CICS) common base event (CBE) dashboard

10.1 Configuring WebSphere Business Monitor

In this section, we describe the configuration tasks that we performed to enable WebSphere Business Monitor to receive events from the various ESBs in Common Base Event (CBE) format. We deployed WebSphere Business Monitor V7.0.0.0 into a stand-alone WebSphere Application Server Network Deployment (ND) V7.0.0.7 environment on a Microsoft Windows XP platform. We consumed events from the ESB using an MQ link between the Common Event Infrastructure (CEI) service integration bus (SIBus) and WebSphere MQ V7.01 on z/OS. The standard WebSphere Business Monitor environment is already enabled with CEI and Business Space. We really only need to perform three tasks for our configuration:

- ▶ Configure the destination on the CEI SIBus
- ▶ Configure the WebSphere MQ to CEI link
- ▶ Configure the WebSphere MQ channels and queues

We obtained helpful information at the following link to the WebSphere Business Monitor Information Center:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.btools.help.monitor.admin.doc/admin/admin_access_cei_factories.html

10.1.1 Defining a CEI bus destination in WebSphere Business Monitor

We do not actually define the destination at this time, because we run a script in the next step that defines the destination for us. However, we need to install an EAR file now, which is a Mediation Handler for our destination. The Mediation Handler makes minor formatting changes to allow the event to be consumed by CEI. The EAR file is named `MQtoCEIMediation.ear` and is located in `<server_home>\scripts.wbm\CEIMQ\`.

Using the Integrated Solutions Console, we install and start the application, as shown in Figure 10-1.

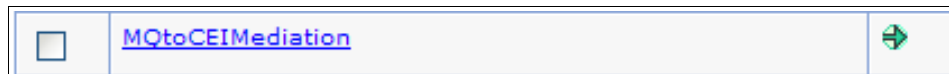


Figure 10-1 Mediation Handler application started

After the next step, we will have a new destination on the CEI SI Bus named CEIQueueAliasForMQ. It will have a Mediation Handler assigned, as shown in Figure 10-2.



Figure 10-2 Mediation Handler assigned to a bus destination

Also important, the script will set a forward routing path on this destination to the queue defined in the CEI service as event input, typically, with this name:

CEI.<cell>.BUS:<node>.<server>.CommonEventInfrastructureQueueDestination

10.1.2 Establishing the MQ to CEI link

In WebSphere Business Monitor, the CEI component provides the services for event management, including the generation, propagation, persistence, and consumption of events. CEI is configured to use a service integration bus (SIBus) for its Java Message Service (JMS) services. This task defines a link between the CEI SIBus and our MQCR WebSphere MQ Queue Manager running on z/OS. We use a script to automate this portion of our work. Its name and location are <server_home>\scripts.wbm\CEIMQ\configCEIForMQClients.bat.

The script can also define the WebSphere MQ required resources when it runs, but we leave that task for the next step.

10.1.3 Defining the MQ channels and queues

We chose to create the following definitions on the z/OS Queue Manager using the WebSphere MQ Explorer V7. The Explorer simplifies the administration of the WebSphere MQ components.

We create the following resource definitions:

- ▶ Receiver channel definition
- ▶ Sender channel definition
- ▶ Remote queue definition
- ▶ Transmission queue definition

Figure 10-3 shows the channel definition that we use for the sending channel.

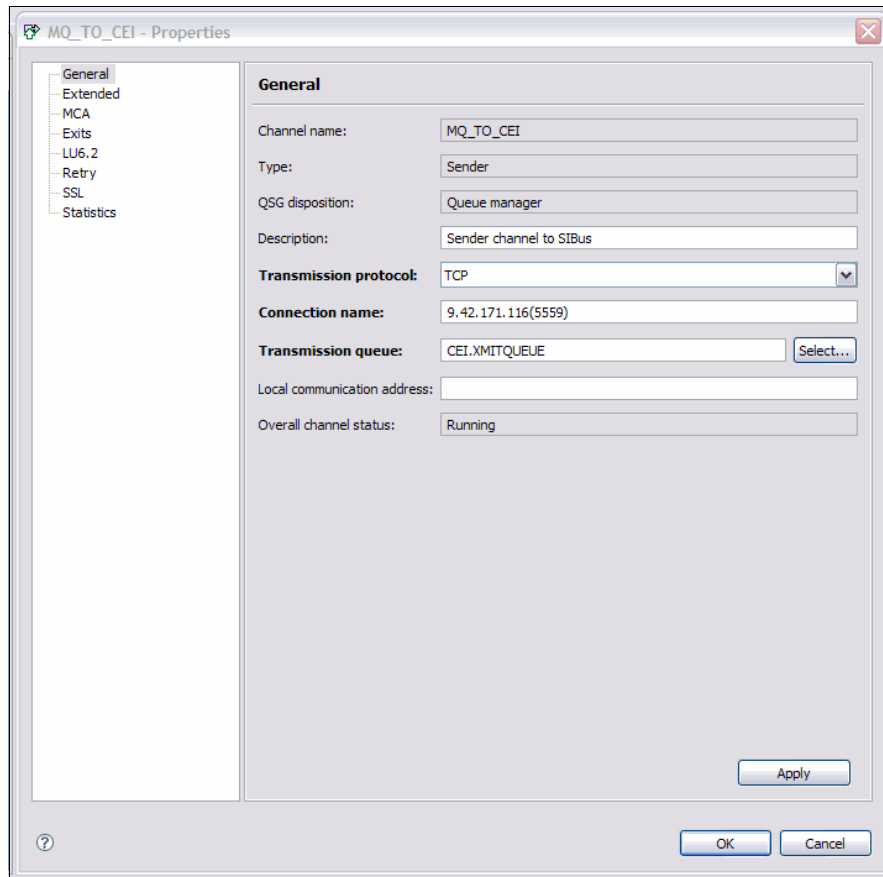


Figure 10-3 Sender channel definition

Figure 10-4 on page 275 shows the definition for the remote queue. Notice the names of the Remote Queue Manager and the remote queue.

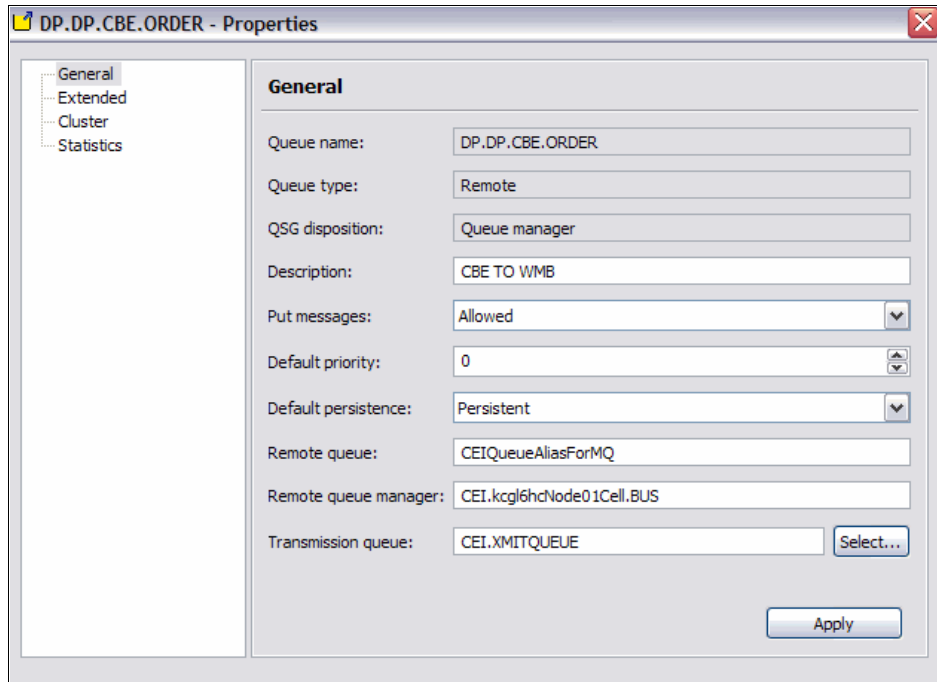


Figure 10-4 Remote queue definition

The final definition is the transmission queue, as shown in Figure 10-5 on page 276. The triggering on the queue is set to Every, which is a good choice for the trigger, ensuring that each time that an event arrives, the trigger is activated to forward the message.

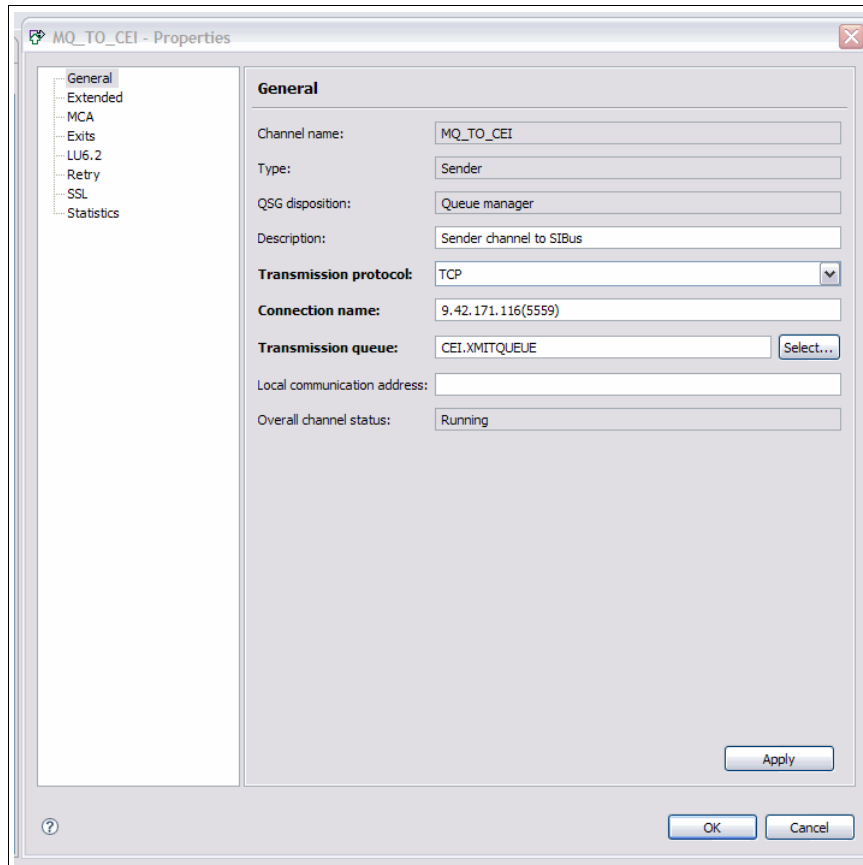


Figure 10-5 Transmission queue definition

10.2 Designing the monitor model

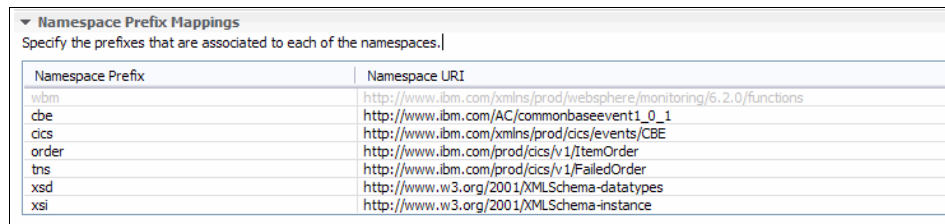
We chose to use WebSphere Integration Developer V7 for our development environment, which includes the toolkit for WebSphere Business Monitor development to create, test, and implement the monitor model. You can also use Rational Application Developer; however, you must install the toolkit into the environment. We do not intend to discuss all the details of developing monitor models in this section; however, for our scenarios, we point out a few helpful tips. The following sections describe the activities that we performed to design and implement our monitor model.

10.2.1 Creating the monitor project and model

We create a new monitor project named RedbooksMM and, within that project, a new monitor model, which we also name RedbooksMM. When we complete our development, we use the pop-up menu item to Generate Monitor JEE Projects and deploy the resulting application to our WebSphere Business Monitor server.

10.2.2 Importing CBE schema

The CICS event message consists of three separate schemas: one schema for the common base event structure, a static event structure, and a dynamic payload, which depends on which type of event is being emitted. We import the XML Schema Definition (XSD) files associated with these definitions into our monitor model, which results in the following namespace prefix mappings being defined to our model. See Figure 10-6.



Namespace Prefix	Namespace URI
wbm	http://www.ibm.com/xmlns/prod/websphere/monitoring/6.2.0/functions
cbe	http://www.ibm.com/AC/commonbaseevent1_0_1
cics	http://www.ibm.com/xmlns/prod/cics/events/CBE
order	http://www.ibm.com/prod/cics/v1/ItemOrder
tns	http://www.ibm.com/prod/cics/v1/FailedOrder
xsd	http://www.w3.org/2001/XMLSchema-datatypes
xsi	http://www.w3.org/2001/XMLSchema-instance

Figure 10-6 Required namespaces and prefixes for CICS CBE

10.2.3 Defining the monitor details model

Figure 10-7 on page 278 shows our details model. We describe its contents briefly, particularly those elements that might be more challenging to define.

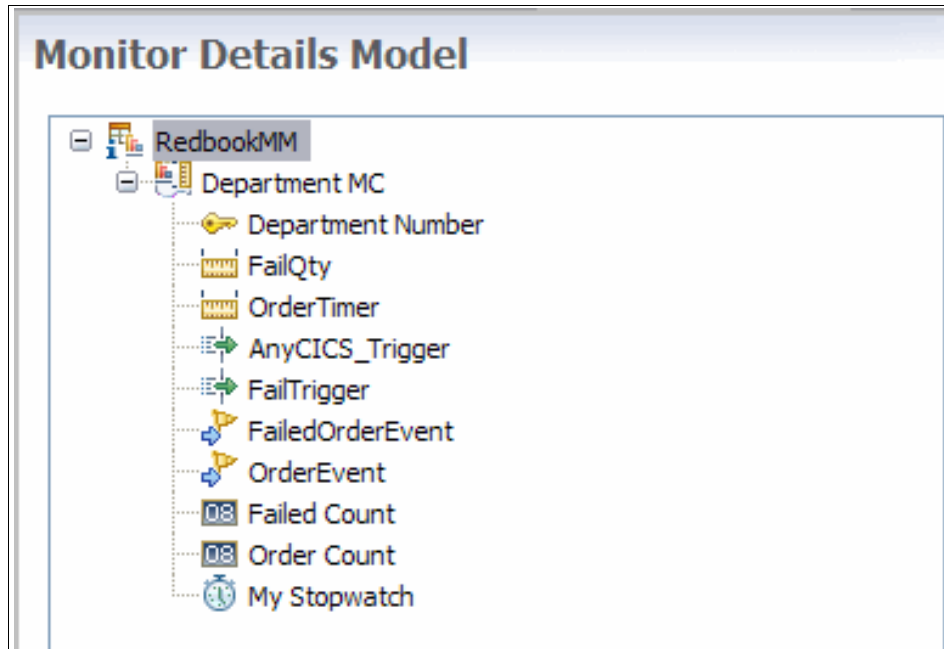


Figure 10-7 Monitor Details Model window

We start by defining a monitor context, which requires us to define a key. In our scenarios, we use the Department Number in the dynamic portion of the CBE as the key. We then define our inbound events, which are a FailedOrder event and an OrderItem event. We then add metrics, counters, a trigger, and a stopwatch.

Defining a key

We need to define two expressions for our key, as shown in Figure 10-8, because our key is found in one of two locations, depending on which event is received.

The screenshot shows a window titled "Key Value Expressions". Below the title, it says "Specify the expressions that set the value of the key." There is a table with two rows, each representing an expression for the key:

Expression
$X+Y$ =? OrderEvent/eventPayload/order:charge_dept
$X+Y$ =? FailedOrderEvent/failedPayload/tns:charge_dept

Figure 10-8 Key Value Expressions window

Defining inbound events

We have two inbound events. Our model must be aware of these events. In the case of a FailedOrder event, we define two event parts and a filter condition, as shown in Figure 10-9.

The screenshot shows the configuration interface for the FailedOrder event. It includes a section for 'Event Type Details' with an 'Extension name' field and a table for 'Event parts'. Below this is a 'Filter Condition' section with a text input field.

ID	Name	Type	Path
cicsEvent	cicsEvent	cics:event	cbe:CommonBaseEvent/cics:event
failedPayload	failedPayload	tns:payload	cbe:CommonBaseEvent/cics:event/cics:payload-data/tns:payload

Filter Condition: FailedOrderEvent/cicsEvent/cics:context-info/cics:eventname = 'FailedOrder'

Figure 10-9 FailedOrder event definition

For our ItemOrder event, we define the following details. Notice the slight differences in the definition of the event parts and the filter condition. See Figure 10-10.

The screenshot shows the configuration interface for the Order event. It includes a section for 'Event Type Details' with an 'Extension name' field and a table for 'Event parts'. Below this is a 'Filter Condition' section with a text input field.

ID	Name	Type	Path
cicsEvent	cicsEvent	cics:event	cbe:CommonBaseEvent/cics:event
eventPayload	eventPayload	order:pay...	cbe:CommonBaseEvent/cics:event/cics:payload-data/order:payload

Filter Condition: OrderEvent/cicsEvent/cics:context-info/cics:eventname = 'ItemOrder'

Figure 10-10 Order event definition

Defining metrics, counters, and triggers

The remaining elements of the detail model are fairly apparent. They rely, of course, on the inbound event definitions that we just created. We define counters for each event type, because we use these counters in our key performance

indicator (KPI) definitions and want to see them in our dashboard. We create an OrderTimer metric, which relies on the AnyCICS_Trigger. This trigger fires when we get either kind of event from CICS. When the trigger fires, we store the current DateTime. We also create a Stopwatch, which we define to start when an Order event arrives and to reset when we receive a FailedOrderEvent. Taking the average of these durations allows us to define a Meantime Between FailedOrders (MTBF) KPI.

10.2.4 Defining the KPI model

We define four KPIs, which will be rendered in our dashboard. Figure 10-11 shows the four KPIs.

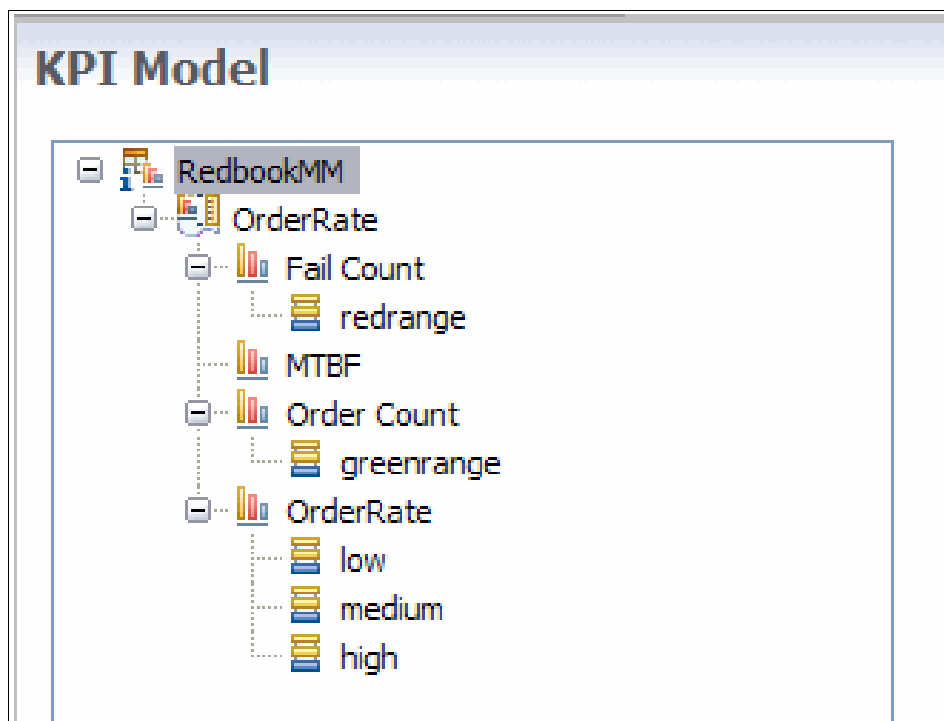


Figure 10-11 KPI model definition

We easily define the Order Count and Fail Count KPIs by using the associated metrics. We use the sum aggregation function. We add a range definition to color the KPI rendering in Business Space. The OrderRate KPI relies on the Order Count and OrderTimer metrics that we defined in the details model. Here, we simply define a 1-minute rolling interval over which to evaluate the KPI, and so, we can monitor the successful order throughput of our catalog system. We add

low, medium, and high ranges to the KPI to allow the widget to use colors as visual indicators. The meantime between failed orders (MTBF) KPI uses the stopwatch that we defined and the average aggregation function.

We complete the development of the monitor model. We have exported a project interchange containing the model, which you can download and use. It is part of Appendix A, “Additional material” on page 401.

10.3 Creating the Business Space dashboard

After creating the monitor model and deploying it to the WebSphere Business Monitor server, you can customize the appearance of the Business Space dashboard, as shown in Figure 10-12. We use the Instances widget from the palette and drag it onto our new page in our Business Space.

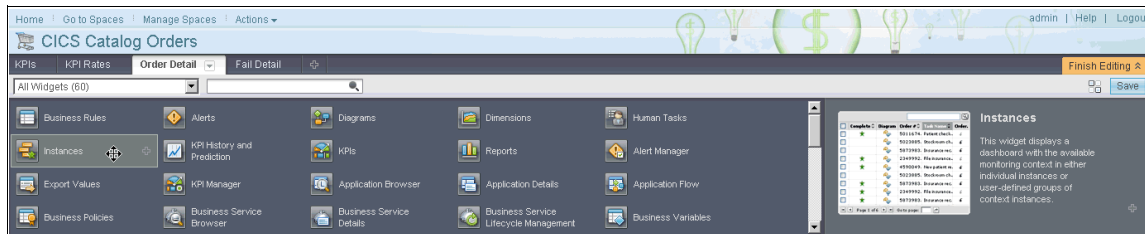


Figure 10-12 Business Space with the widget palette

After it is on the page, we can configure the widget based on the elements available in our model, as shown in Figure 10-13 on page 282. We define a page for Orders and a separate page for Failed Orders, and each page has its own Instances widget.

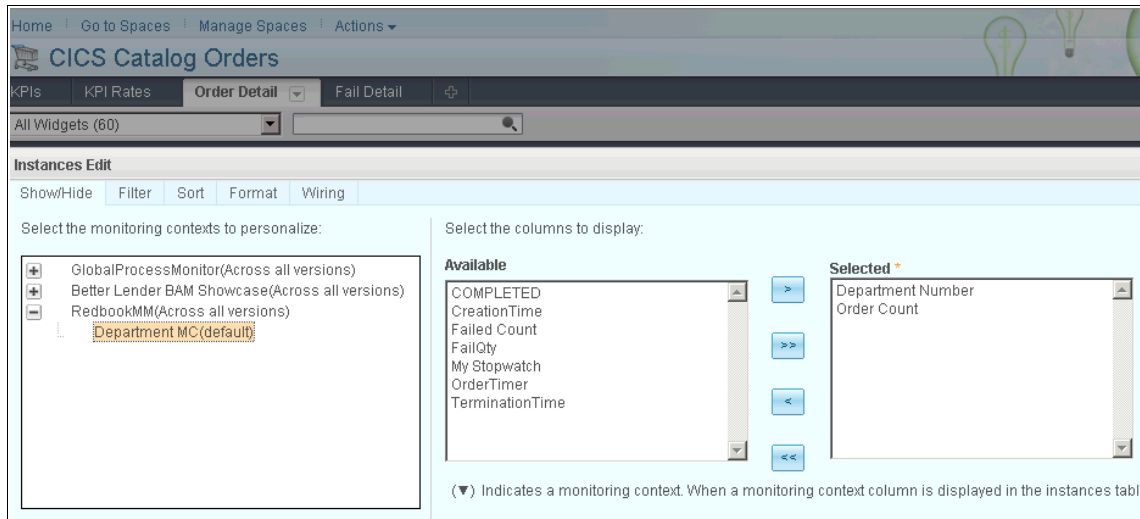


Figure 10-13 Configuring the Instances widget

In a similar manner, we use the KPI widget to build the two KPI pages in our dashboard. Configuring the KPI widget is as easy as selecting the KPI of interest, as shown in Figure 10-14.

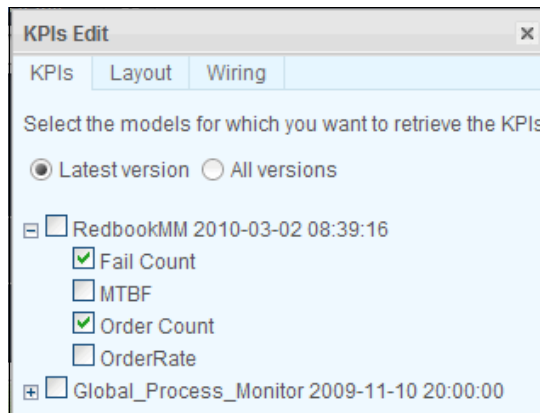


Figure 10-14 Configuring the KPI widget

We complete our tailoring of the Business Space. We have exported the Business Space definition, which is available for download in Appendix A, “Additional material” on page 401.

10.4 Viewing the CICS CBE monitor dashboard

Now that our monitor model is deployed and our Business Space is tailored, our last step is to drive the load on our catalog system. WebSphere Business Monitor Toolkit includes a stand-alone test capability called the *Integrated Test Client*, which can be used to simulate sending events to the monitor model. The Integrated Test Client is a useful facility for developing and testing monitor models in parallel to the development of the monitored application. We demonstrate how to use the Integrated Test Client, which can be used as an alternative source for populating the Business Space dashboard. We look at our dashboard in the following sections, showing a business view of all the new function that our scenarios have added to our original application.

10.4.1 Sending test events with the Integrated Test Client

If you have a WebSphere Integration Developer Version 7 environment that includes the unit test environment for WebSphere Business Monitor, you can send test events to your monitor model using the Integrated Test Client. You must have already deployed the generated enterprise application to the unit test server. Then, we right-click the monitor model to start the Integrated Test Client by selecting “Launch Integrated Test Client” against the monitor model, as shown in Figure 10-15 on page 284.

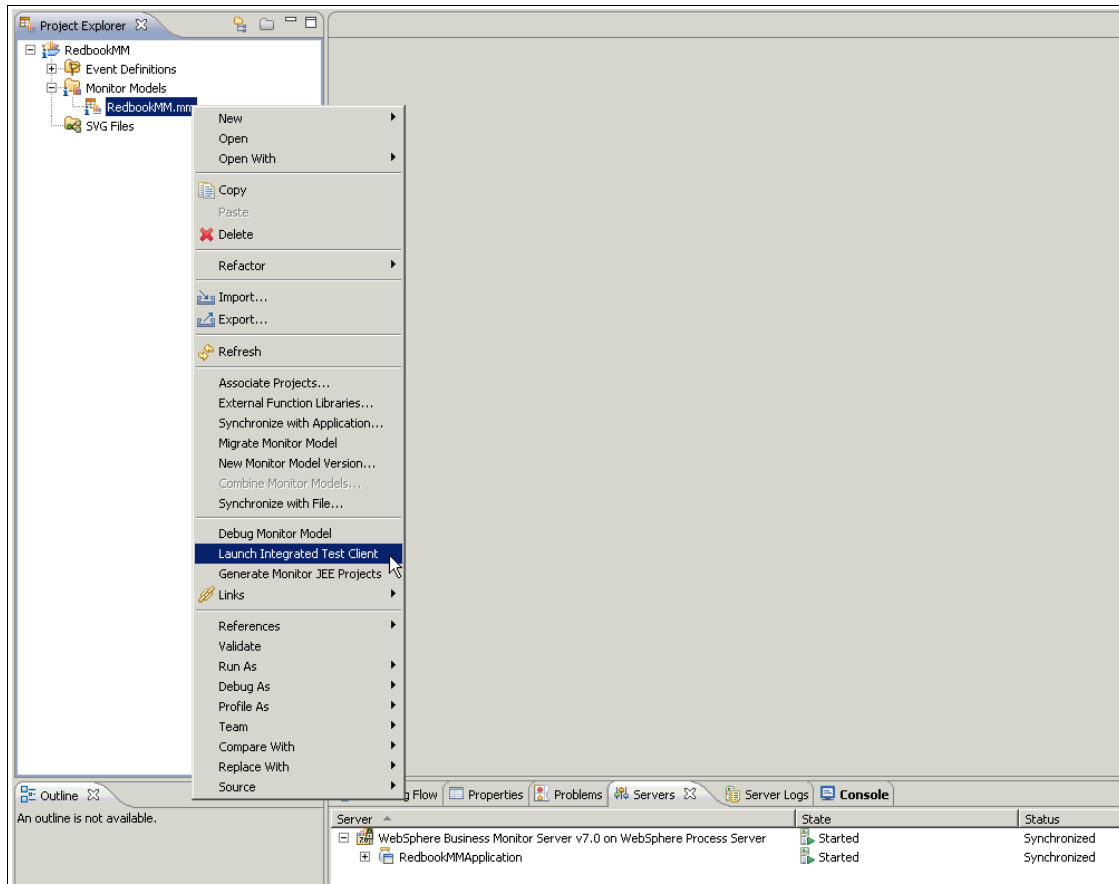


Figure 10-15 Launching Integrated Test Client

The Integrated Test Client editor opens for this model (Figure 10-16 on page 285). You can build a test script by specifying event details and adding them to the script. We have provided a sample script named `Monitor Test Events.xml` in Appendix A, “Additional material” on page 401.

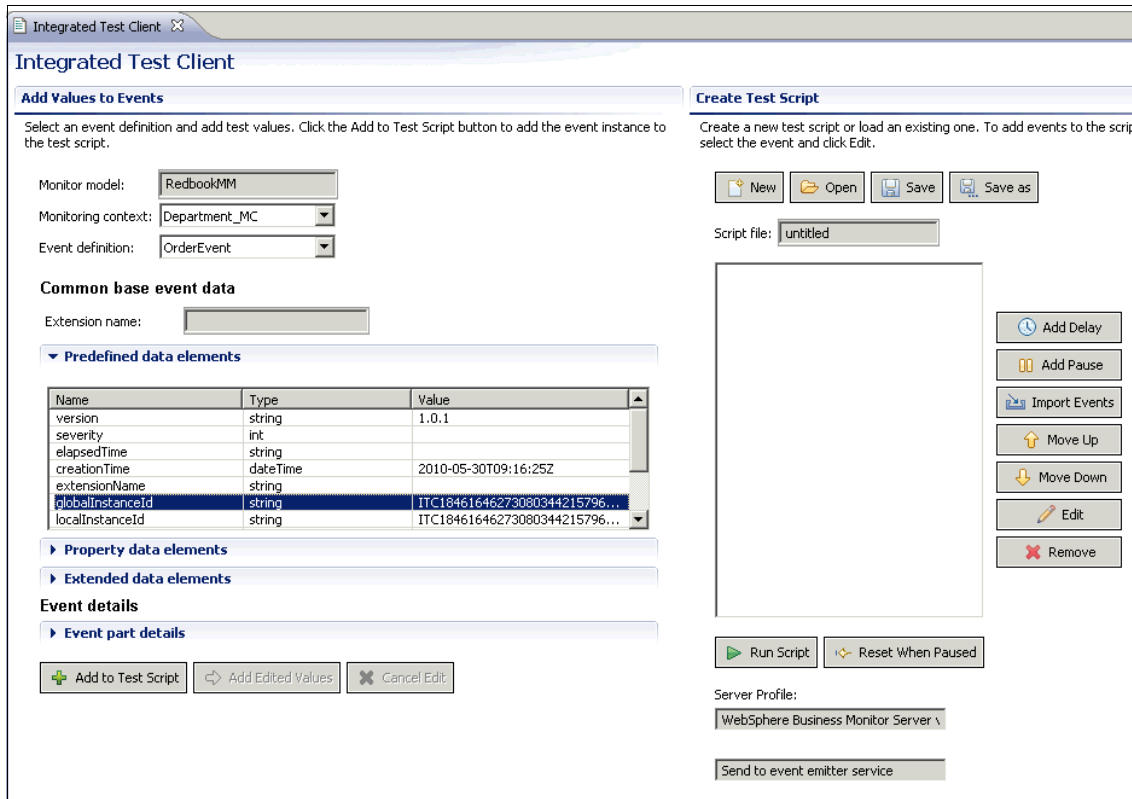


Figure 10-16 Setting predefined data elements

First, we specify a successful order event, so we chose the OrderEvent event definition. There are predefined data elements that must be present in every CBE event. We add two attributes, creationTime and globalInstanceId, to the default attributes, as in Figure 10-16. For creationTime, a calendar widget allows us to choose a date and time. The attribute globalInstanceId must be unique for every event, so we copy the localInstanceId contents and adjust it each time.

The Event part details section contains the real business data for the event, along with the event identifier (our monitor model specifies an event filter so that it recognizes which events to consume based on this event identifier). Figure 10-17 on page 286 shows how we set this event identifier.

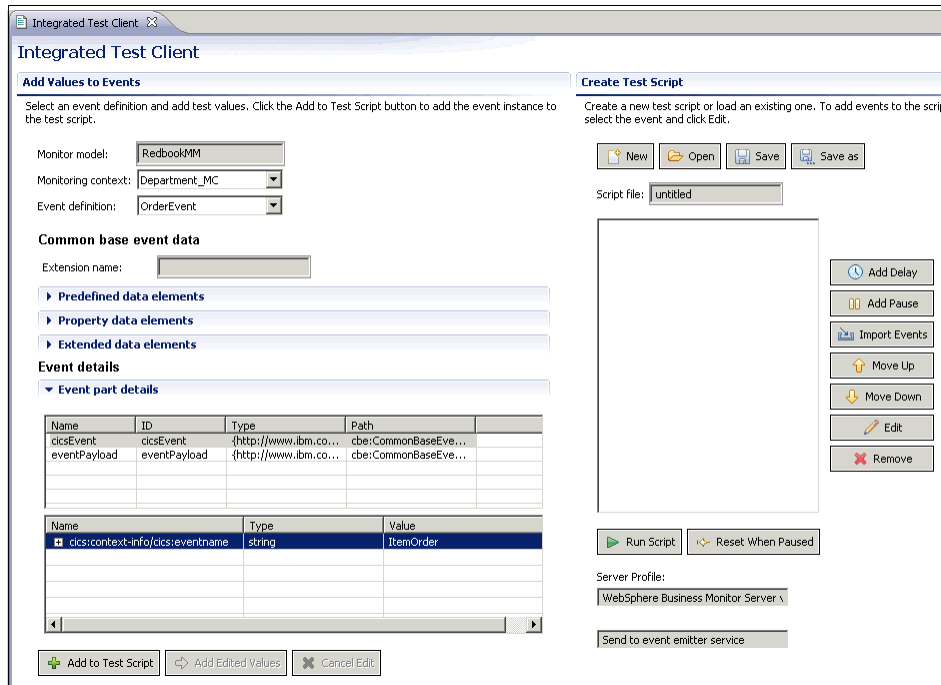


Figure 10-17 Event name for an order

The eventPayload event part contains the relevant business data that is passed with each order, in this case, a charging department, as shown in Figure 10-18 on page 287. After we have set the event details, we can add the event to the script by using “Add to Test Script”.

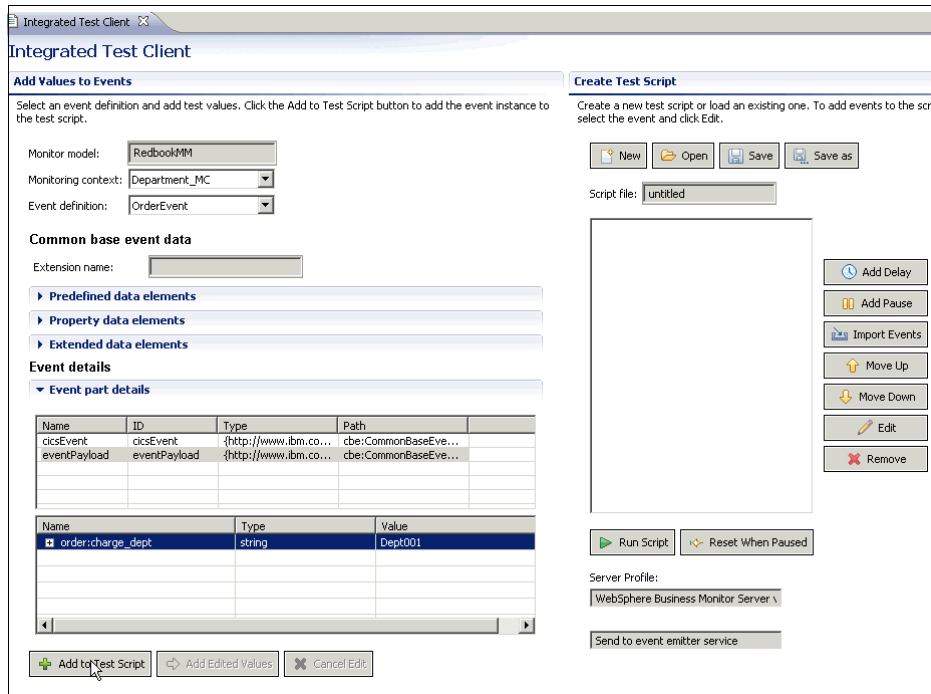


Figure 10-18 Event payload for an order

The test script now has the event added to it (Figure 10-19 on page 288). We can run the script now, if we choose.

More often, though, you build a scenario of a number of events in your test script before running the script and submitting those events against your monitor model.

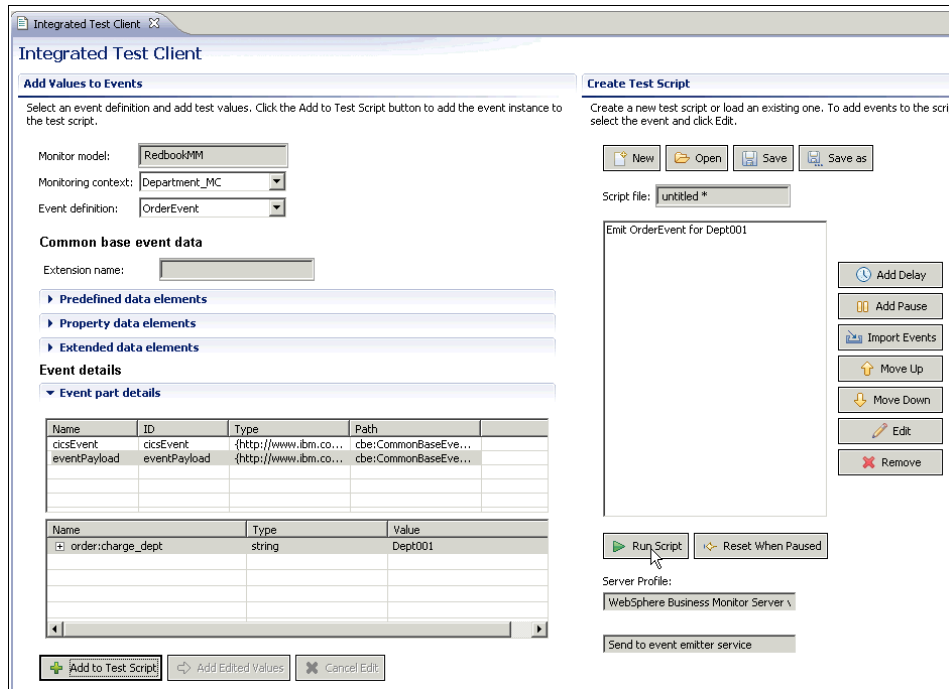


Figure 10-19 Script with event added

The monitor model also listens for failed orders, so to emulate that situation, we add a FailedOrderEvent to the script. We set the predefined data elements in a similar way as the previous event and set the event name to FailedOrder, as shown in Figure 10-20 on page 289.

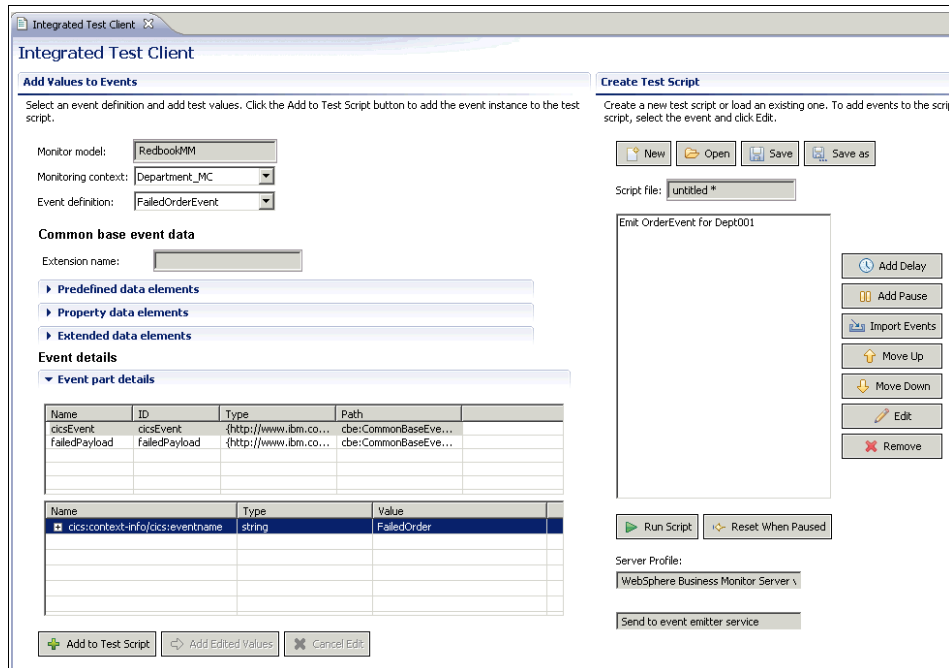


Figure 10-20 Event name for a failed order

We set the business data for the event, as shown in Figure 10-21 on page 290, and the event is added to the test script.

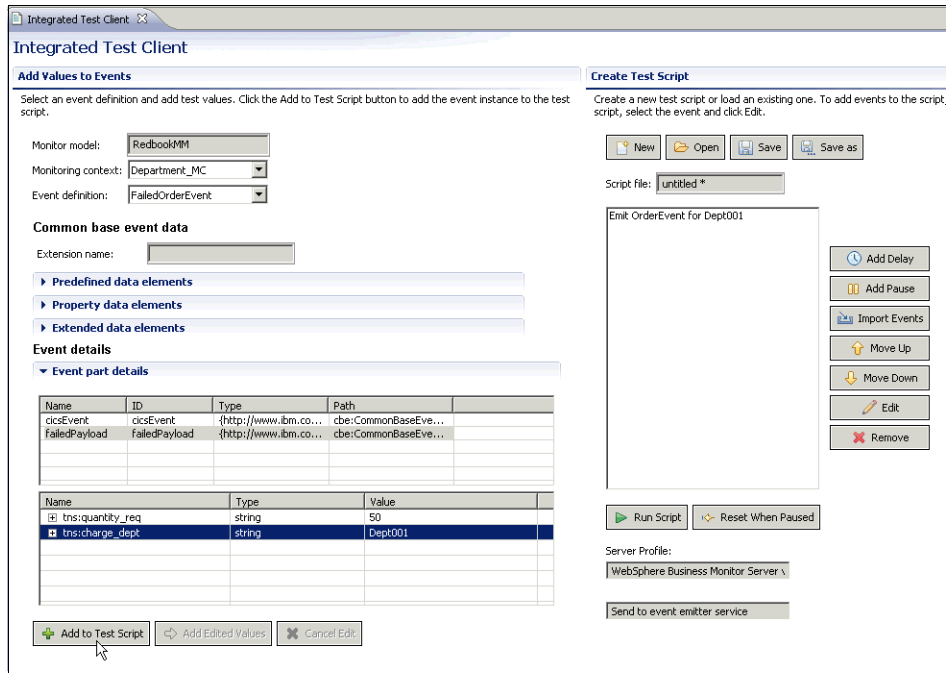


Figure 10-21 Event payload for a failed order

You can open the example script that is provided and examine its contents, as shown in Figure 10-22 on page 291.

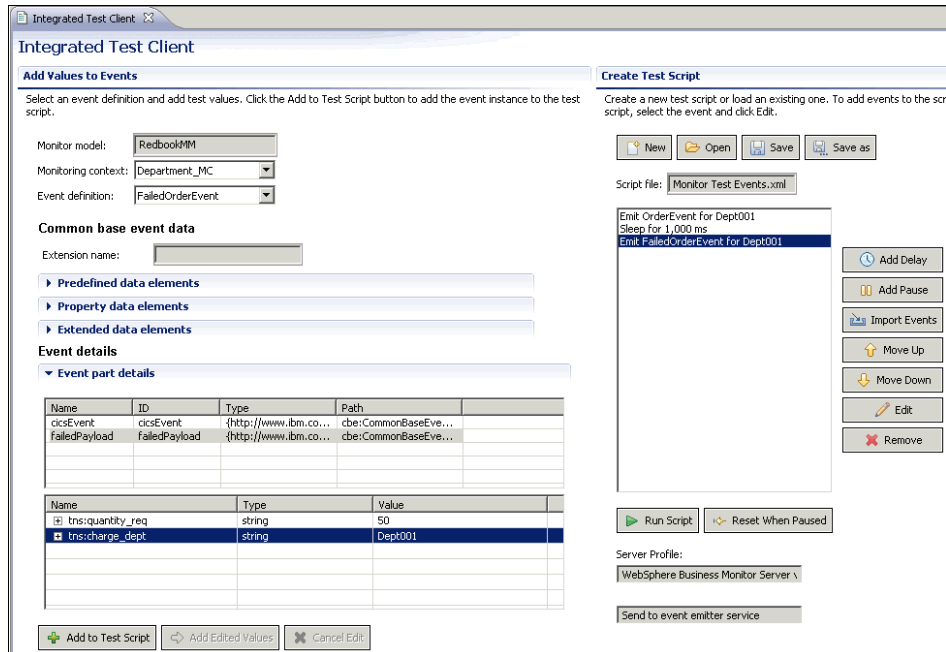


Figure 10-22 Example script file contents

Using the Integrated Test Client, you can build representative scenarios and load your monitor model with test cases that can then be examined in the Business Space dashboard. In the following sections, we show examples of the various monitoring widgets within the dashboard.

10.4.2 Successful order instances

The events generated in scenario 1 are emitted from the CICS catalog order application and are then routed to WebSphere Business Monitor without enrichment from the ESB. We show the results of scenario 1 in the Business Space Order Detail page, which is shown in Figure 10-23 on page 292.

Department Number	Order Count
Dept0001	35
Dept0002	82
Dept0003	69
Dept0004	66
Dept0005	34

Figure 10-23 Order instances

10.4.3 Insufficient stock instances

In scenario 3, the events are emitted from CICS, flow to the ESB, and are transmitted to WebSphere Business Monitor before the event is enriched. Figure 10-24 shows the Failed Count for each department in the Business Space Fail Detail page. The failed events are the results of insufficient stock on hand for the ordering department.

Department Number	Failed Count
Dept0001	11
Dept0002	4
Dept0003	17
Dept0004	0
Dept0005	12

Figure 10-24 Failed order instances

10.4.4 Total successful and failed orders

When we place a small load on the catalog system, both successful and failed order events start to occur. The counts of those events display on the KPI page of the Business Space, as shown in Figure 10-25.



Figure 10-25 Order Count and Failed Count KPIs

10.4.5 Event rate and average meantime between failure orders

Also, under load, we can see the average meantime between failed orders (MTBF) and the overall successful order throughput (OrderRate), as shown on the KPI Rate page of the Business Space, which is shown in Figure 10-26 on page 294.



Figure 10-26 Average meantime between failed orders (MTBF) and Order Rate KPIs

10.5 Summary

In this chapter, we have illustrated how easily the events emitted by our CICS applications can be exposed to business users in a meaningful business-relevant way. We have discussed the configuration tasks that we performed for our WebSphere Business Monitor server to consume the events that CICS emitted. These events were routed and written to a WebSphere MQ queue on z/OS by our ESB. We then discussed the design and development of the monitor model, including the inbound event definitions, details and KPI models. After we deployed our monitor model to the WebSphere Business Monitor server, we configured the instances and KPI widgets in Business Space, and finally began to drive a load through our catalog system. The resulting events were depicted as KPIs on our dashboard, and so, we had new business insight into our catalog application.



WebSphere Process Server

We described the multiple insufficient stock failures event in 8.4, “Scenario 4” on page 245. In this chapter, we describe using WebSphere Integration Developer to create the process that is triggered by the web service call from WebSphere Business Events when a multiple insufficient stock failures event is detected.

11.1 Process

To develop and test our process, we used WebSphere Integration Developer V7. If you are new to developing processes in WebSphere Integration Developer V7, it can seem daunting at first. The WebSphere Integration Developer V7 Information Center provides a good “HelloWorld” sample, which is relatively easy to understand and duplicate for yourself, but to then move from that to a process that actually does something useful is quite a step. There are many examples in the WebSphere Integration Developer V7 Information Center and elsewhere, but even with these examples, there is still quite a learning curve involved.

Hopefully, the description in this chapter of how we built our process helps with your understanding of how to use WebSphere Integration Developer V7 to build a process.

11.1.1 Designing the process flow

Before beginning to create the process flow in WebSphere Integration Developer V7, it is a good idea to sketch out how you want your process to work. Figure 11-1 on page 297 shows our outline of how we wanted our process to work.

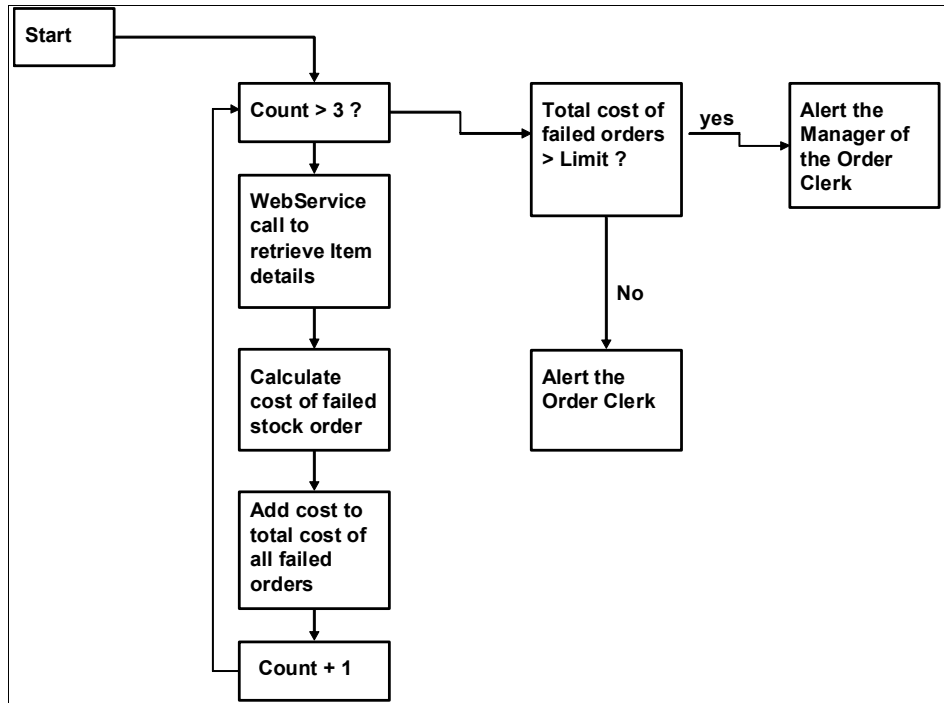


Figure 11-1 Proposed process flow

WebSphere Business Events invokes our process when WebSphere Business Events detects that there have been three failed orders for the same product item number over a defined period of time. WebSphere Business Events passes to our process the details of each of the three failed orders. In our process design that is shown in Figure 11-1, we perform a Web Service call for each item in the failed orders. This approach is a bit redundant because each failed order is for the same item; therefore, we can set up the process to call the Web Service only one time.

However, we set up the process to call a Web Service for each item for two reasons:

- ▶ We might want to cater for a situation where WebSphere Business Events is set up to detect three failed orders for any type of item but for a particular customer over a certain period of days. In which case, we can use this same process.
- ▶ We add this extra complexity to the process, even though it is not strictly needed, to provide a good opportunity to show how relatively easy it is to set up a process to handle a relatively complex flow.

11.2 Building the process

Next, we describe in detail how we built our process.

11.2.1 Products we used

We installed WebSphere Integration Developer V7.0.0 onto a Microsoft Windows XP machine. We then applied the following products:

- ▶ Service Component Architecture (SCA) Feature Pack Version 1.0.1.1
- ▶ IBM WebSphere Application Server Network Deployment (ND) Version 7.0.0.7
- ▶ XML Feature Pack Version 1.0.0.1
- ▶ IBM WebSphere Process Server Version 7.0.0.1

11.2.2 CICS Web Services Description Language

The information about the failed item orders passed from WebSphere Business Events to our process did not contain the item cost. Therefore, our process needs to obtain this information from somewhere. We decided to use a Web Service call directly to CICS to perform this task.

In an Enterprise Service Bus implementation (because our IBM Redbooks publication is focused on how to use an Enterprise Service Bus as the intermediary between separate components), this Web Service call goes to the ESB, which then routes it to the service provider. However, we did not follow this approach due to time constraints. Note, though, that it is a fairly simple matter to replace the direct Web Service call to CICS with a Web Service call to an ESB, if required.

For our process to be able to perform the WebService call to CICS, we need the Web Services Description Language (wsdl) file. We obtained this file and stored it locally in a folder on our Microsoft Windows personal computer. Later in this chapter, we show how we imported this wsdl into WebSphere Integration Developer V7.

The Web Services Description Language (WSDL) that we used was generated by a CICS-provided tooling process. The default values set in the XML file were created by this process. You can change these values to provide more meaningful names, as required.

For example, the initial wsdl had the following lines:

```
<xsd:element name="DFHOXCMNOperation" nillable="false"
type="tns:ProgramInterface"/>
<xsd:element name="DFHOXCMNOperationResponse" nillable="false"
type="tns:ProgramInterface"/>
```

Importing this initial wsdl into WebSphere Integration Developer V7 resulted in two interfaces appearing with the same name. Two interfaces with the same name subsequently leads to confusion about which interface to use when creating the process.

It is important that users performing this sort of activity for real applications give careful consideration to the names that are set in the wsdl file that is generated by CICS tooling.

11.2.3 Starting WebSphere Integration Developer V7

To create a process in WebSphere Integration Developer V7, we start the product by selecting Start → IBM WebSphere Integration Developer → IBM WebSphere Integration Developer V7.0 → WebSphere Integration Developer V7.0.

11.2.4 Creating a business integration project

To use WebSphere Integration Developer V7 to create our process, we first need to create a project that will store all the various components. In the initial WebSphere Integration Developer V7 window, we click the New link in the Projects area, select the Module type, as shown in Figure 11-2 on page 300, and, then, click Next.

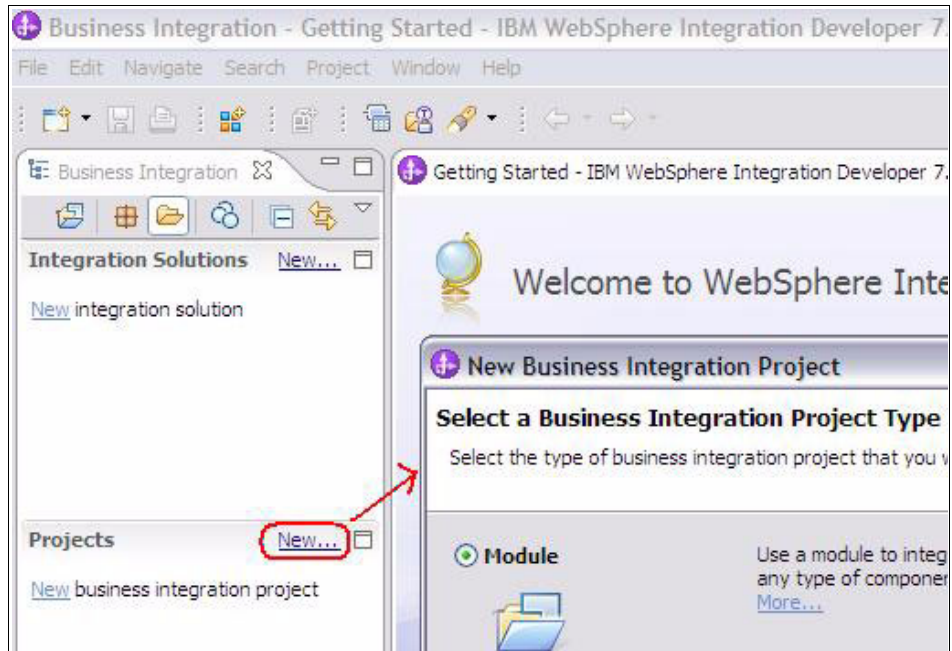


Figure 11-2 Creating a new business integration project

In the next window that opens, we enter the name of our module, `multiLowStockEvent`, as shown in Figure 11-3.

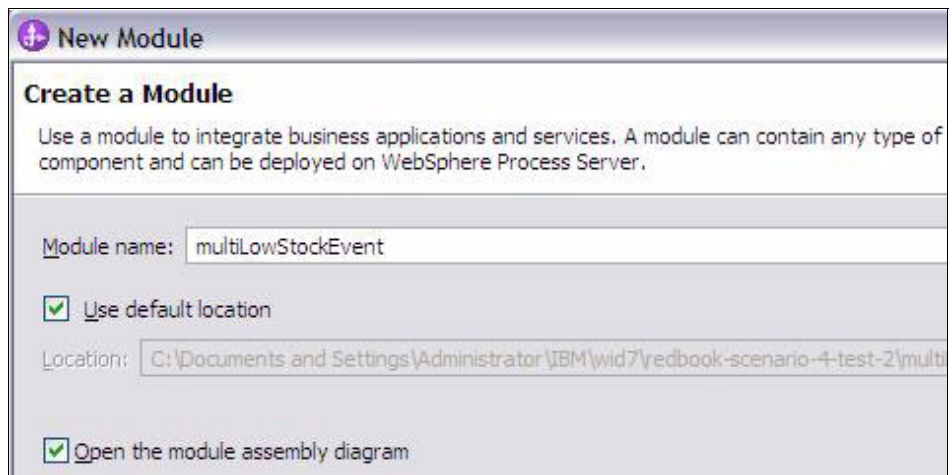


Figure 11-3 Entering the name of the new project

We click Finish. WebSphere Integration Developer V7 creates the initial project framework.

11.2.5 Defining business objects

We knew we were receiving an event from WebSphere Business Events that contained the details of three failed stock orders. Furthermore, we decided that WebSphere Business Events will send a web service call to WebSphere Process Server to initiate a process to handle the event.

Therefore, WebSphere Business Events needs a wsdl file describing the web service call to be able to perform the call. We decided to use WebSphere Integration Developer V7 to define and generate the wsdl that will be used by WebSphere Business Events.

We first needed to define business objects that map the expected data.

Defining the business object for one failed order

First, we define a business object to describe a single failed order. We select Data Types → New → Business Object, as shown in Figure 11-4.

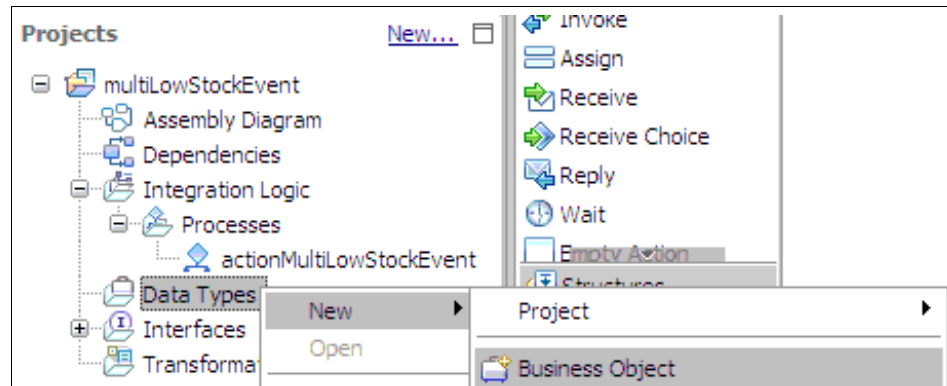


Figure 11-4 Starting process to define a new business object

In the window that opens, we set the name of the new business object to lowStockRequest, as shown in Figure 11-5 on page 302.

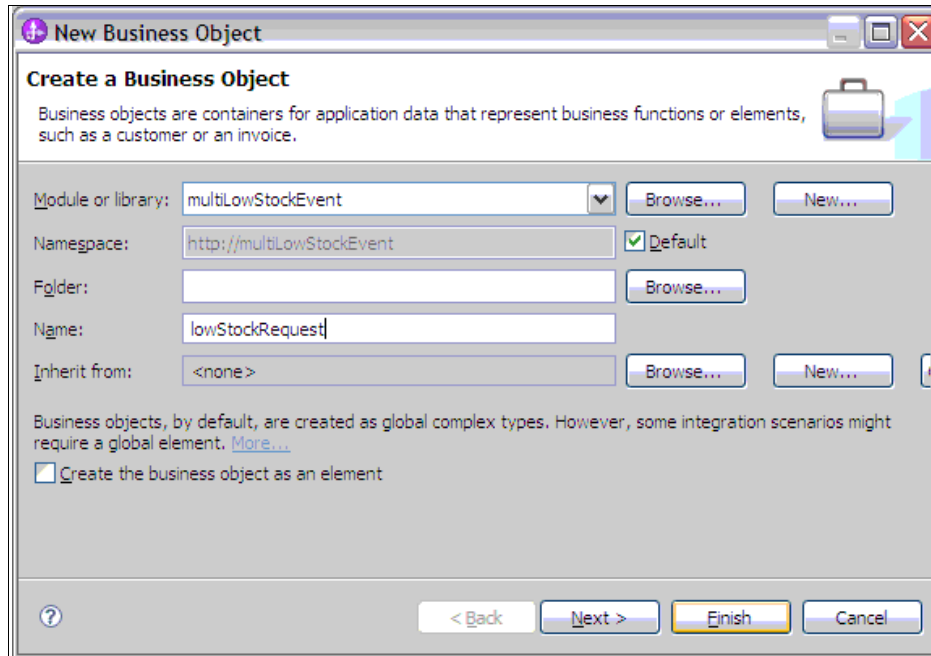


Figure 11-5 Naming the new business object

We click Finish, and WebSphere Integration Developer V7 displays the initial definition of the object.

Next, we add as many fields as needed to the object. We need to add the following fields:

- ▶ Item number
- ▶ Quantity
- ▶ User name
- ▶ Charge department

To add a field to the business object, we click the icon with an F in a blue circle, which adds a field to a business object. Figure 11-6 on page 303 shows that we clicked this symbol four times to add four fields to the business object.

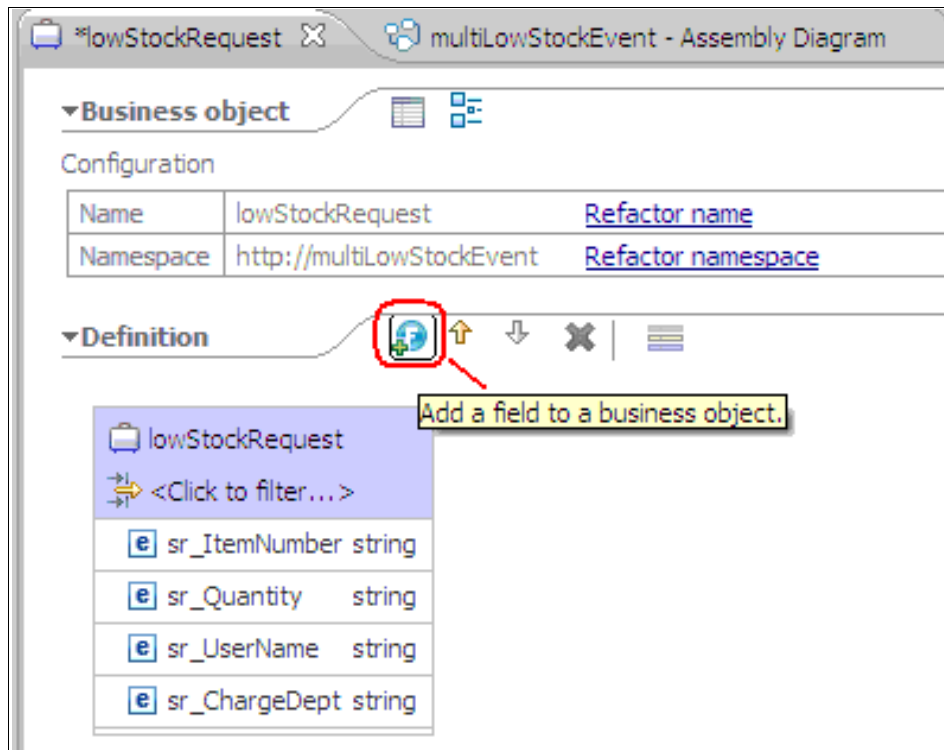


Figure 11-6 Adding fields to the business object

If you click a property, the Properties tab shows various attributes that you can set for that property, such as the type of field and the initial value, as shown in Figure 11-7 on page 304.

We did not set any properties for these fields.

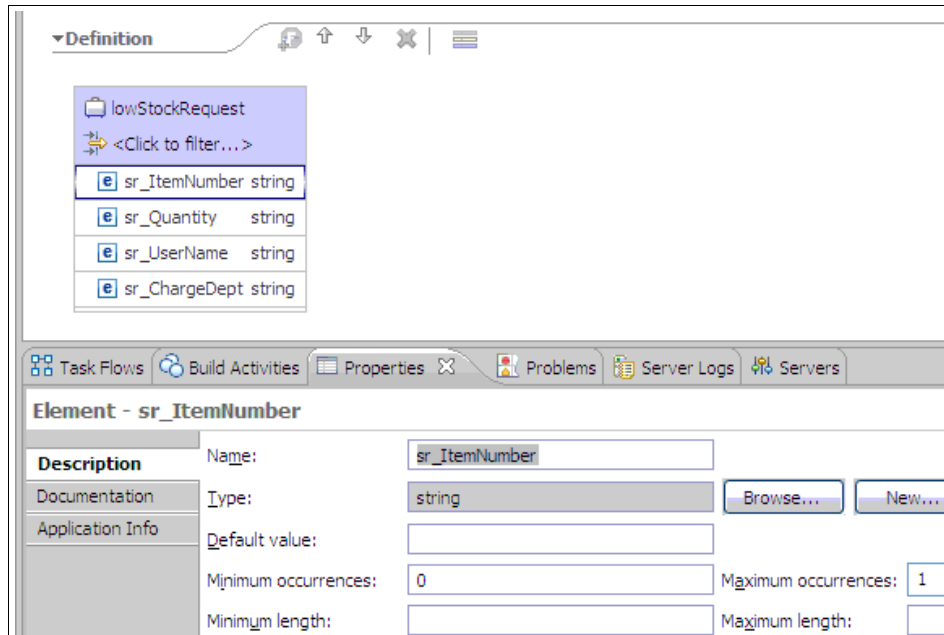


Figure 11-7 Setting properties for a field

We then close the lowStockRequest tab and click Yes to the prompt to save the changes that we had made.

Defining a business object for an array of failed orders

We then define a new business object, which is an array type object containing three individual failed business objects.

We create a new business object called listLowStockRequestEvent. In this new business object, we add one field of type lowStockRequest. Figure 11-8 on page 305 shows the business object that we created.

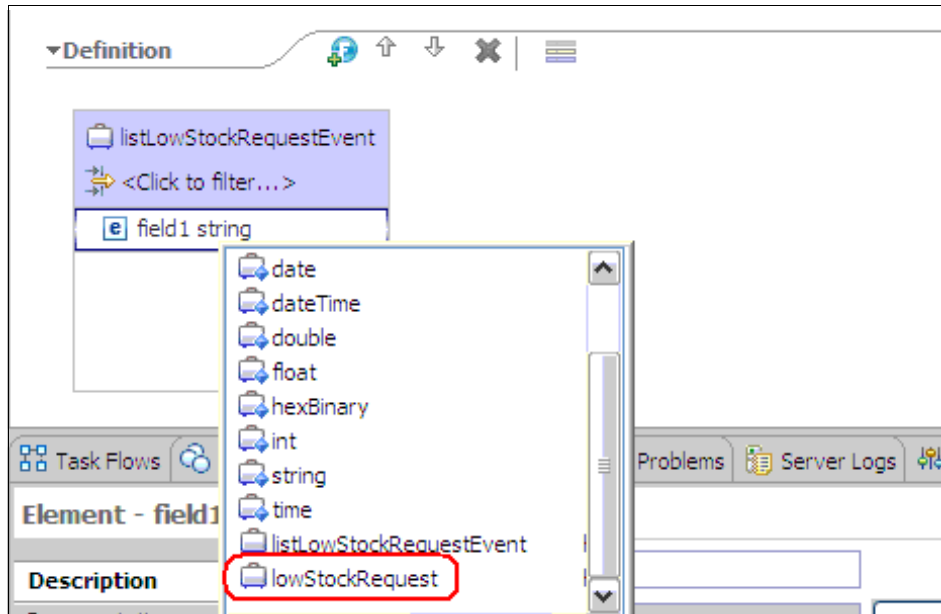


Figure 11-8 Adding a field of type `lowStockRequest`

We call the new field `sr_event`, as shown in Figure 11-8. In the properties for this field, we set the minimum and maximum occurrences to 3, because we know that WebSphere Business Events will send information about three failed events. We then close the tab for `listLowStockRequestEvent` and click Yes to save the changes (Figure 11-9 on page 306).

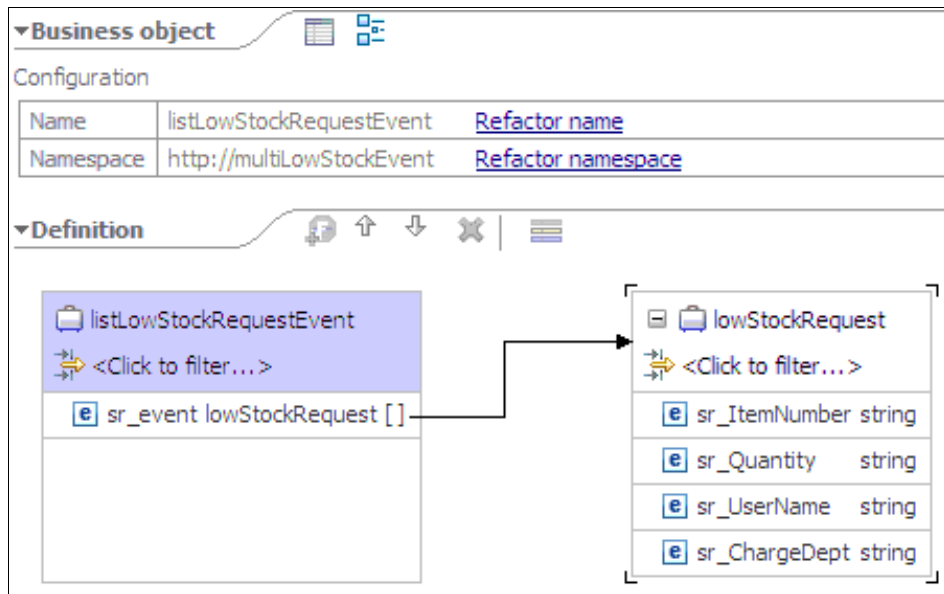


Figure 11-9 Defined business object

11.2.6 Creating a new business process

The next step is to create a new business process. WebSphere Integration Developer V7 supports top-down, bottom-up, and meet-in-the-middle approaches to creating a process. Using the top-down approach in WebSphere Integration Developer V7 simplifies the creation of the overall process and is the approach that we use. To define our process, we select the Process object from the Palette menu and drag and drop it onto the assembly diagram canvas. We then overwrite the default name with the value `actionMultiLowStockEvent`, as shown in Figure 11-10 on page 307.

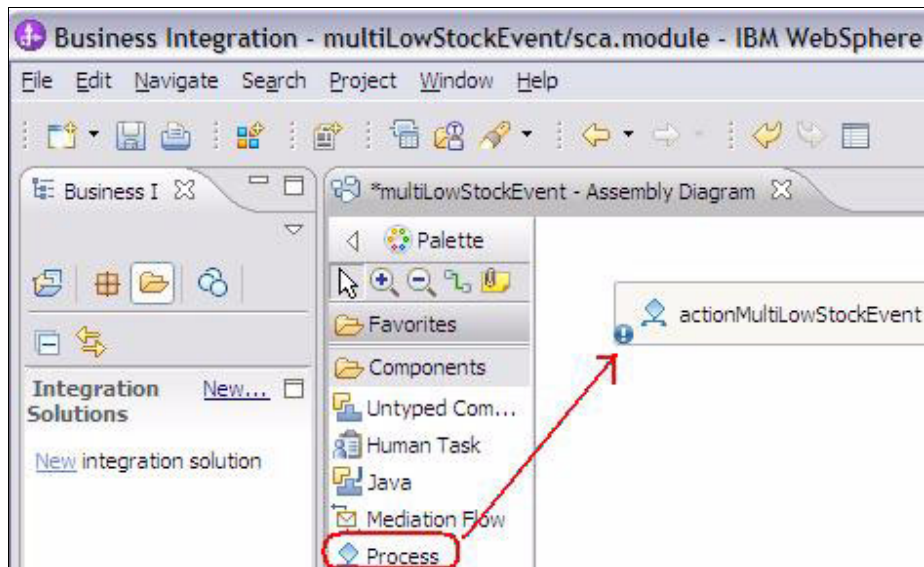


Figure 11-10 Creating a new business project

We save the assembly diagram by clicking the save icon that is located near the upper-left corner of the WebSphere Integration Developer V7 window.

When you click the save icon, you receive this warning message:

CWSCA8004W: The actionMultiLowStockEvent component has no implementation

This message means that, at this stage, there is no Business Process Execution Language (BPEL) for this process. We will generate the BPEL for this process soon.

Creating the interface to the process

Our process needs an interface to allow it to be invoked. To add an interface, we select the actionMultiLowStockEvent object. When we select it, three small icons appear over the object. If you hover the mouse on the first icon on the left, WebSphere Integration Developer V7 shows you that this icon is the Add Interface icon. We click the Add Interface icon, and then, we click New in the window that is shown in Figure 11-11 on page 308.

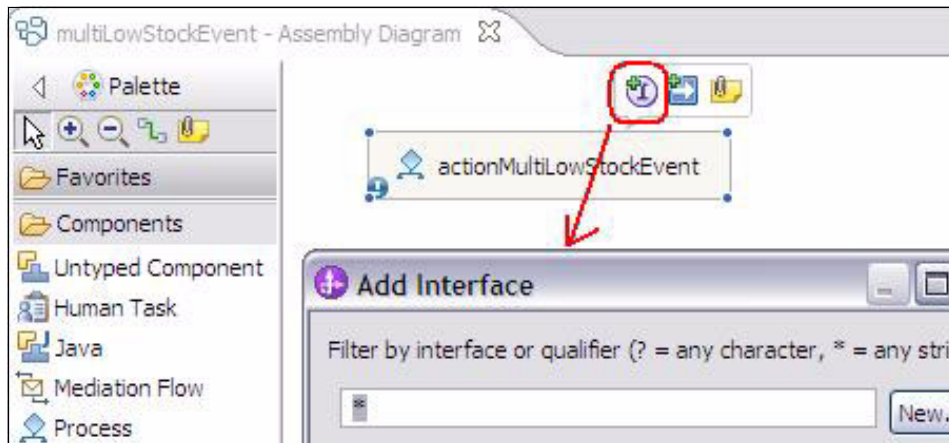


Figure 11-11 Adding an interface to the process

In the window that opens, we enter the name of the new interface, ActionMultiLowStockEvent, as shown in Figure 11-12, and click Finish.

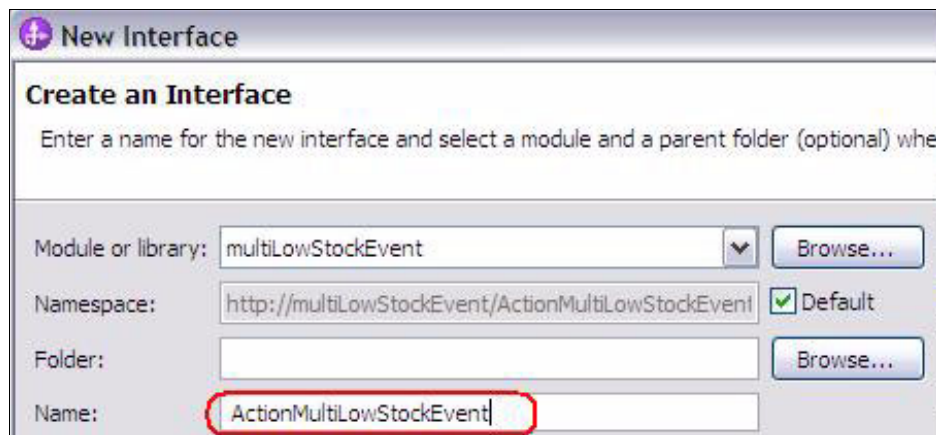


Figure 11-12 Setting the name of new interface

WebSphere Integration Developer V7 then creates the interface and opens a tab to allow us to modify the newly created interface, as shown in Figure 11-13 on page 309.

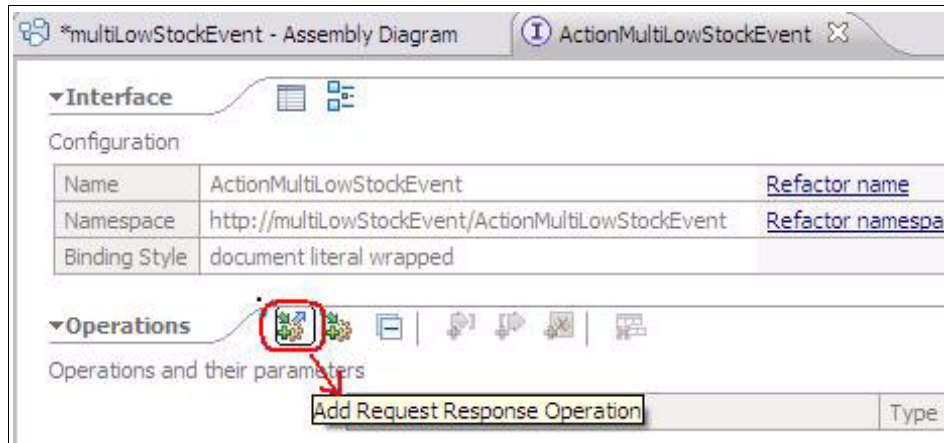


Figure 11-13 Default values in process interface

Adding a request response operation

We click the Add Request Response Operation icon, which added a default operation called operation1 with default input and output entries. We can either overtype operation1 with a new value, or in the Properties tab, enter the new name there. We set the new name to `actionMultiLowStockEvent`.

Changing the input

We then click the default input name of `input1` and changed this name to `lowStock_event`. Then, in the type for the input field, we select `string`, which brought up a selection box, from which we select `listLowStockRequestEvent`.

Changing the output

We change the name of the output field to `lowStock_event_replyMsg` and leave the type as `string`.

Then, we close the tab to save the changes.

Generating the process web service interface

Next, we generate web service bindings for our process by right-clicking the process and selecting `Generate Export` → `Web Service Binding`, as shown in Figure 11-14 on page 310.

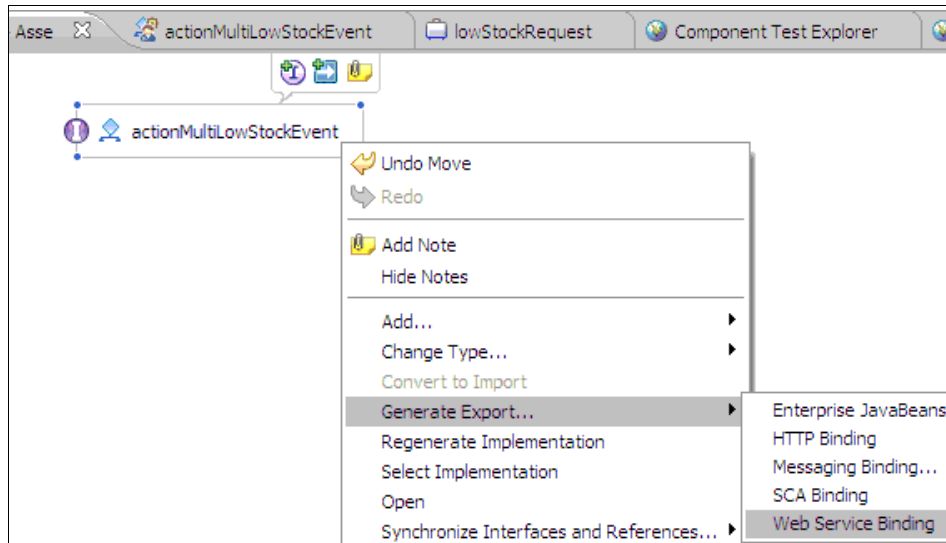


Figure 11-14 Start of process to generate Web Service binding for our process

WebSphere Integration Developer V7 opens a window in which you can select the type of transport protocol to use for the web service. In our case, we select SOAP1.1/HTTP and click Finish. After WebSphere Integration Developer V7 completes this generation process, there is a new item under the Web Service Ports part of the project. Also, an Export activity is added to our assembly. The assembly, at this point, looks similar to Figure 11-15.

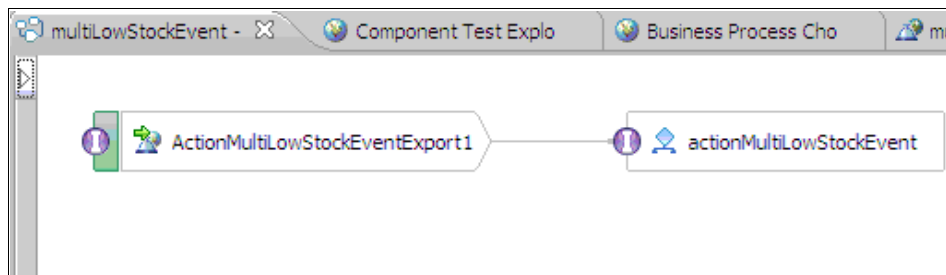


Figure 11-15 Assembly diagram after generating Web Service binding

Generating the process implementation

The next step is to generate the implementation of what we have so far designed in the Assembly Diagram view. We select the `actionMultiLowStockEvent` object, then right-click it, and select `Generate Implementation`, as shown in Figure 11-16 on page 311.

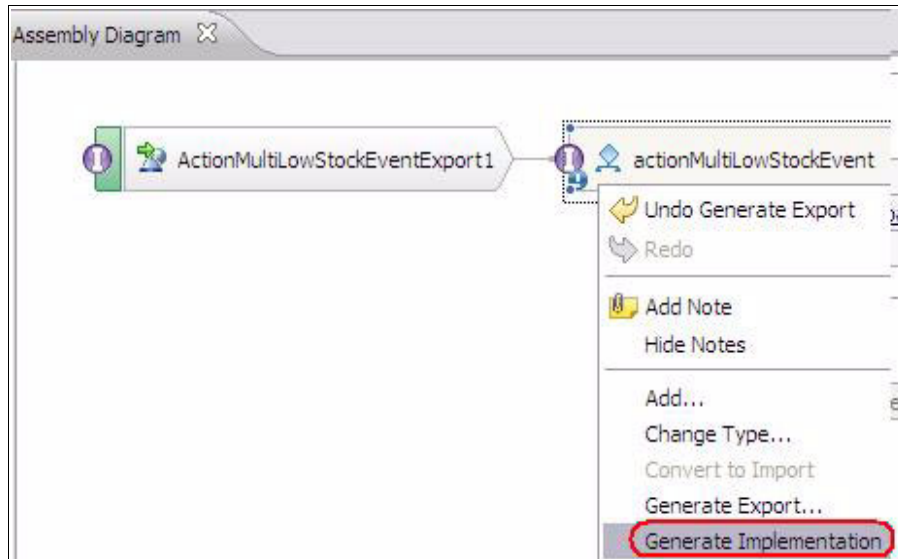


Figure 11-16 Generating the implementation of the process

This selection opens a window that is similar to Figure 11-17.

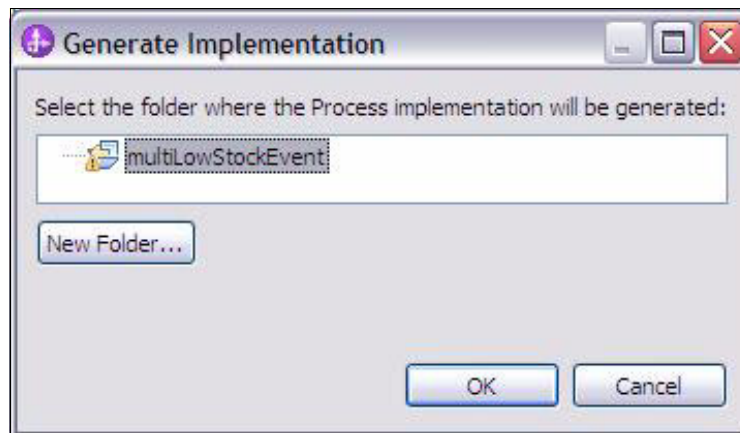


Figure 11-17 Setting location where process implementation will be created

We click OK, and WebSphere Integration Developer V7 generates the implementation of the process. Figure 11-18 on page 312 shows parts of the WebSphere Integration Developer V7 window after this action completes. The new process is listed under Integration Logic → Processes, and the initial implementation of the flow is shown in a new tab.

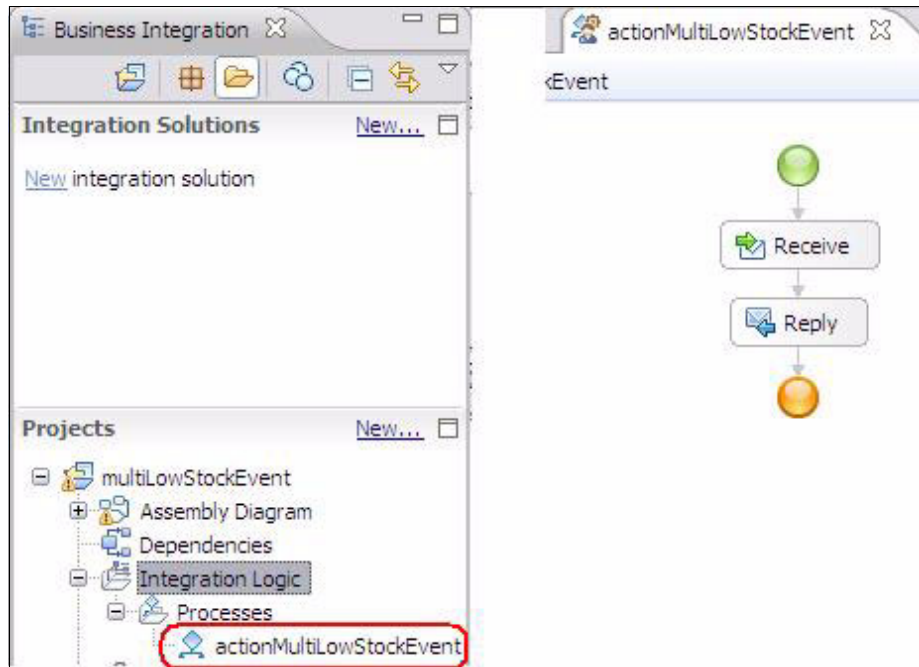


Figure 11-18 The created process implementation

We save the updated assembly diagram. There is no warning message now, because an implementation of the process exists.

11.2.7 Variables

When it generates the process implementation, WebSphere Integration Developer V7 automatically creates two variables called `lowStock_event` of type `listLowStockRequestEvent` and `lowStockEvent_event_replyMsg` of type `string`. We can see these variables under the Variables subtab on the right side of the `actionMultiLowStockEvent` process tab.

WebSphere Integration Developer V7 uses the input and output names that were set in the interface to the process that we created in “Creating the interface to the process” on page 307 to create these variables.

These variables are called *global variables*, because any activity that is in the process can reference these variables. You can also define *local variables*, which can only be accessed by a subpart of a process. Later in this chapter, you will see how we define and use local variables.

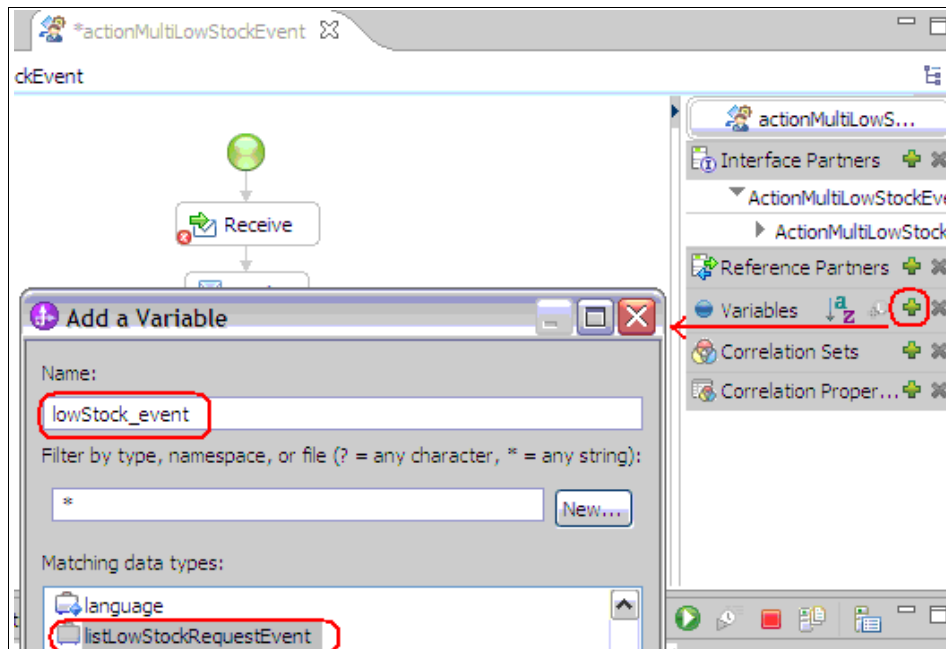


Figure 11-20 Adding a new global variable to the process

11.2.8 Adding a snippet

When developing a new process, especially if you are new to this type of task, typically, you develop it in stages and try to test it as you go along. Adding a snippet to the process, especially during the development stages, is a useful way of adding trace messages for debug purposes.

We add a snippet by clicking the snippet activity under Basic Actions and dragging it between the Receive and Reply activities, as shown in Figure 11-21 on page 315.

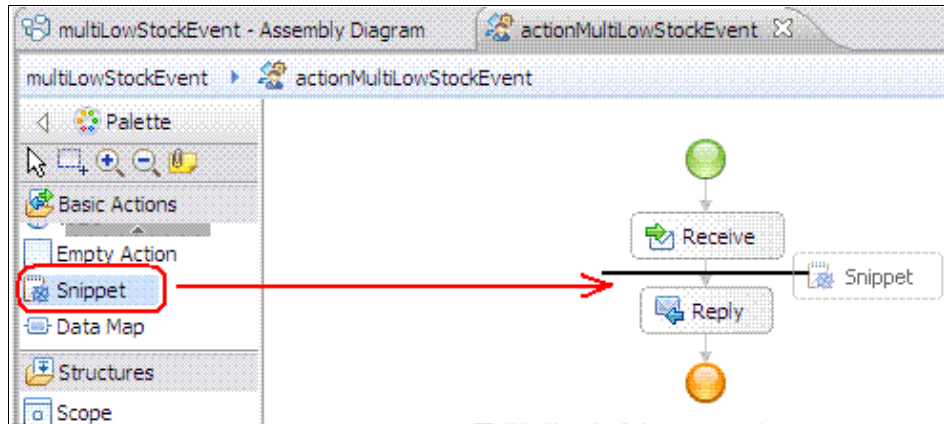


Figure 11-21 Adding a snippet to the process

You can configure a snippet visually or by adding Java code, which is the method that we use in this case. To add Java code, we select the snippet activity and, then, in the Properties tab, select Java. WebSphere Integration Developer V7 opens a window with a warning message and a question about the switch, as shown in Figure 11-22, to which we answered Yes.

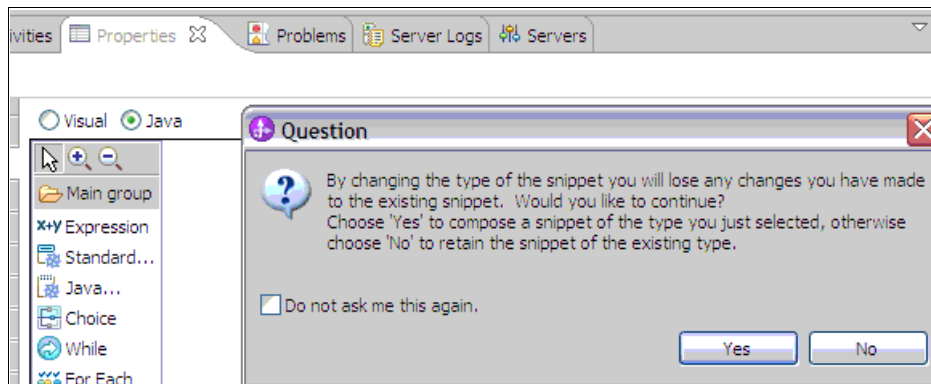


Figure 11-22 Changing to Java coding mode in a snippet

We add the Java code that is shown in Example 11-1 to our snippet.

Example 11-1 Java snippet code to display data received by the process

```
Date now = new Date();
System.out.println("Process to handle multiple low stock event started
at: " + now);
```

```
// Create a Business Object Factory
```

```

BOfactory bofactory =
    (BOfactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOfactory")
;

// Create an object for a low stock item

DataObject isr = bofactory.create("http://multiLowStockEvent",
"lowStockRequest");

// Retrieve all the low stock events received by the process

List stockRequestsList = lowStock_event.getList(0);

System.out.println("stockRequestList: " + stockRequestsList );

// Get the number of low stock items retrieved

int numberOfItems = stockRequestsList.size();

System.out.println("Number of isr = " + numberOfItems);

String sr_UserName = null, sr_ItemNumber = null, sr_Quantity = null;

// Print out info from each low stock event

for (int ii=0; ii < numberOfItems ; ii++) {

    // Get a low stock event

    isr = (DataObject) stockRequestsList.get(ii);

    // Get fields from the object

    sr_UserName = isr.getString("sr_UserName");
    sr_ItemNumber = isr.getString("sr_ItemNumber");
    sr_Quantity = isr.getString("sr_Quantity");

    System.out.println(ii + " sr_userid: " + sr_UserName +
        " sr_ItemNumber: " + sr_ItemNumber +
        " sr_Quantity: " + sr_Quantity );
}

```

When we save these changes, WebSphere Integration Developer V7 displays a number of compile errors in our Java code, because our Java code uses classes that the process does not know about. To resolve this problem, we need to add import statements to the process. We can click anywhere in the Process tab, then click the Properties tab, and there is a tab called Java Imports. In this area, we add the Java imports that are needed for the process. Example 11-2 shows all the Java imports that we added to our process.

Example 11-2 Java imports needed for the process

```
import com.ibm.websphere.bo.BOFactory;
import com.ibm.websphere.sca.ServiceManager;
import commonj.sdo.DataObject;
import java.util.List;
import java.util.ArrayList;
import java.util.Date;
```

Figure 11-23 shows adding the Java imports to the process.

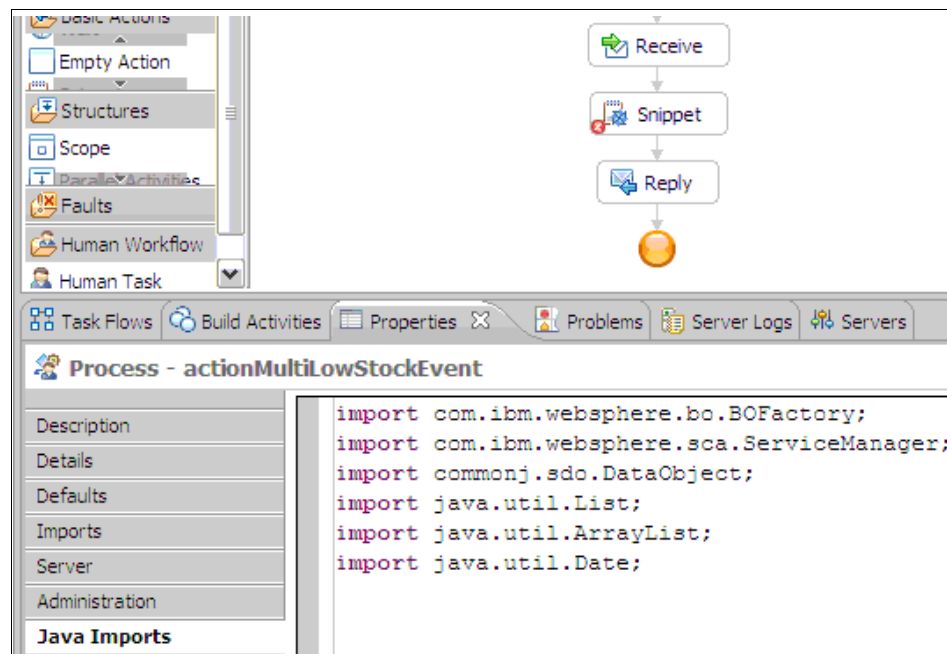


Figure 11-23 Adding Java imports to the process

Saving these changes resolves the errors in the Java snippet.

Understanding the use of the Business Object factory

These lines are in the Java code:

```
B0Factory bofactory =  
    (B0Factory)  
    ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/B0Factory")  
    ;  
DataObject isr = bofactory.create("http://multiLowStockEvent",  
    "lowStockRequest");
```

The first line creates a Business Object factory, which is then used to create a data object of a specific type. The values that are used in the second line are obtained from the defined business object. You can find these values in the Description tab of the Properties tab of the data type, as shown in Figure 11-24.

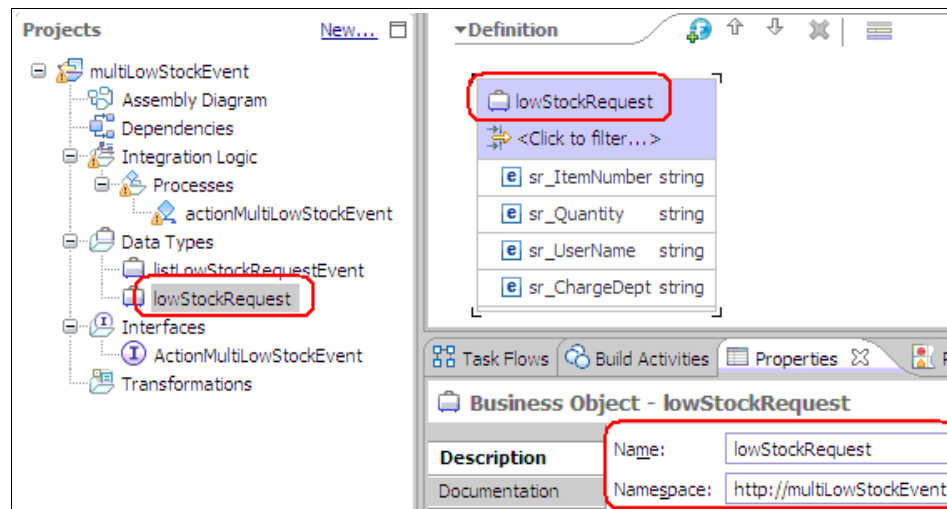


Figure 11-24 Where to find values to use when using a Business Object factory

11.2.9 Time for a first test run

After we complete setting up the assembly diagram, we can test it. We first start the WebSphere Process Server in WebSphere Integration Developer V7 by selecting the Servers tab, right-clicking the WebSphere Process Server server, and selecting Start. After WebSphere Process Server in WebSphere Integration Developer V7 starts, we need to add our project to the server. We right-click the server and select Add and Remove Projects. In the window, we select the project, click Add, and then, click Finish, as shown in Figure 11-25 on page 319.

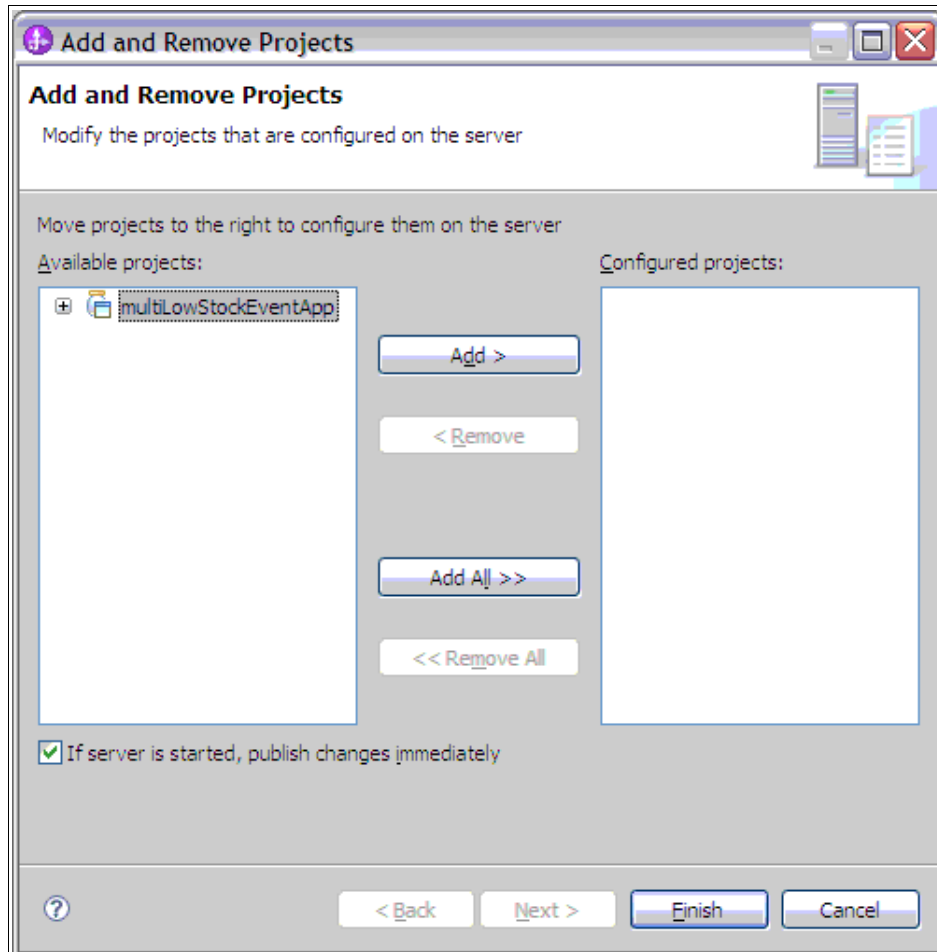


Figure 11-25 Adding the project to the server

WebSphere Integration Developer V7 provides a number of ways to test our process. We can use the built-in Business Process Choreographer Explorer or the built-in Web Services Explorer tool. Or, we can test the process from the Assembly Diagram view by using the Integrated Test Client.

Testing with Business Process Choreographer Explorer

We right-click the server, select Launch → Business Process Choreographer Explorer, and enter our user ID and password that we set when we installed WebSphere Integration Developer V7. In the window under the Process Templates heading, we click Currently Valid and our process is displayed, as shown in Figure 11-26 on page 320.

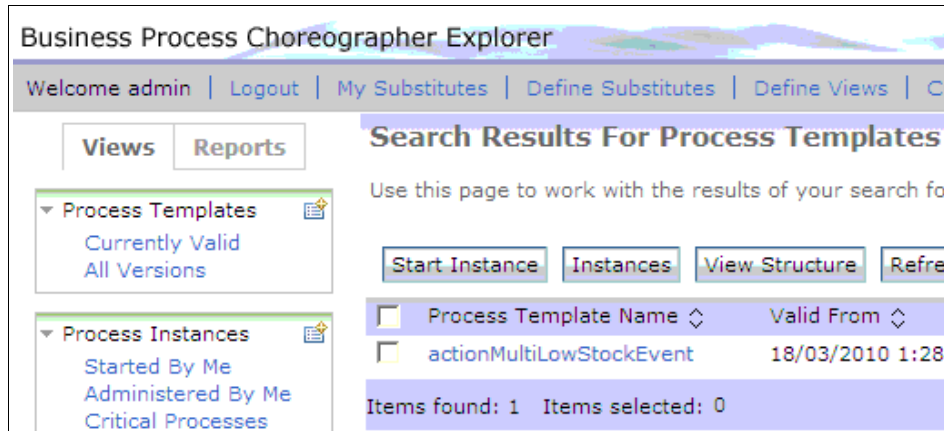


Figure 11-26 Our process displayed in Business Process Choreographer Explorer

We select the process and click Start Instance, and WebSphere Integration Developer V7 opens a window called Process Input Message. This window shows the input fields that are passed to the process. We fill in the fields with random values, click Add, and then, click Submit to invoke the process. The initial window opens.

During this time, our process runs. We verify that our process ran by looking for the messages that the snippet Java code has written to the server SystemOut log. We locate this file on our Microsoft Windows server at this location:

```
C:\zProducts\IBM\WID7_WTE\runtimes\bi_v7\profiles\qws\logs\server1\SystemOut.log
```

In the log, we see the following output, which confirmed that our process was called correctly:

```
[19/03/10 12:24:49:156 EST] 000000bc SystemOut      0 Process to handle
multiple low stock event started at: Fri Mar 19 12:24:49 EST 2010
[19/03/10 12:24:49:156 EST] 000000bc SystemOut      0 stockRequestList:
[BusinessObject: lowStockRequest@6c0c6c0c (sr_ItemNumber=1,
sr_Quantity=10, sr_UserName=first, sr_ChargeDept=alpha),
BusinessObject: lowStockRequest@6c196c19 (sr_ItemNumber=2,
sr_Quantity=21, sr_UserName=second, sr_ChargeDept=beta),
BusinessObject: lowStockRequest@6c266c26 (sr_ItemNumber=3,
sr_Quantity=23, sr_UserName=third, sr_ChargeDept=gamma)]
[19/03/10 12:24:49:156 EST] 000000bc SystemOut      0 Number of isr = 3
[19/03/10 12:24:49:156 EST] 000000bc SystemOut      0 0 sr_userid: first
sr_ItemNumber : 1 sr_Quantity : 10
[19/03/10 12:24:49:156 EST] 000000bc SystemOut      0 1 sr_userid:
second sr_ItemNumber : 2 sr_Quantity : 21
```

[19/03/10 12:24:49:156 EST] 000000bc SystemOut 0 2 sr_userid: third
sr_ItemNumber: 3 sr_Quantity: 23

Testing with Web Services Explorer

Because we set up our process to be invoked by a Web Service call, we can use the Web Services Explorer tool in WebSphere Integration Developer V7 to invoke our process through this method. Under Web Service Ports, we right-click `ActionMultiLowStockEventExport1_ActionMultiLowStockEventHttpPort` and select **Web Services** → **Test with Web Services Explorer**, as shown in Figure 11-27.



Figure 11-27 Starting the Web Services Explorer tool

Figure 11-28 on page 322 shows the initial view of the Web Services Explorer tool displayed by WebSphere Integration Developer V7.

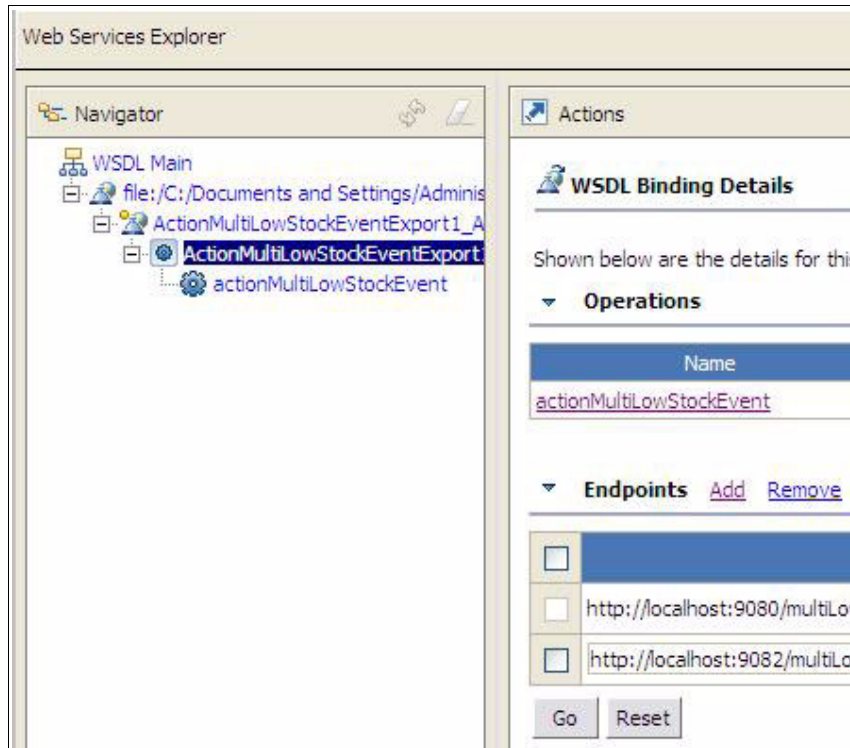


Figure 11-28 Initial Web Services Explorer view

In our case, because we defined a second process server within our WebSphere Integration Developer V7 environment, we need to add the second endpoint shown (Figure 11-28). We click Add, which repeated the first endpoint, and we then change the port from 9080 to 9082 and click Go to update the endpoint list. To prepare for the Web Service call, we click `actionMultiLowStockEvent` on the left side.

WebSphere Integration Developer V7 then displays a view that allows us to set up values to pass on the Web Service call. To set up the Web Service call, we click Add beside `sr_Event` three times and, then, for each variable, click Add and enter a value. The complete window is too large to show, but Figure 11-29 on page 323 shows a part of it.

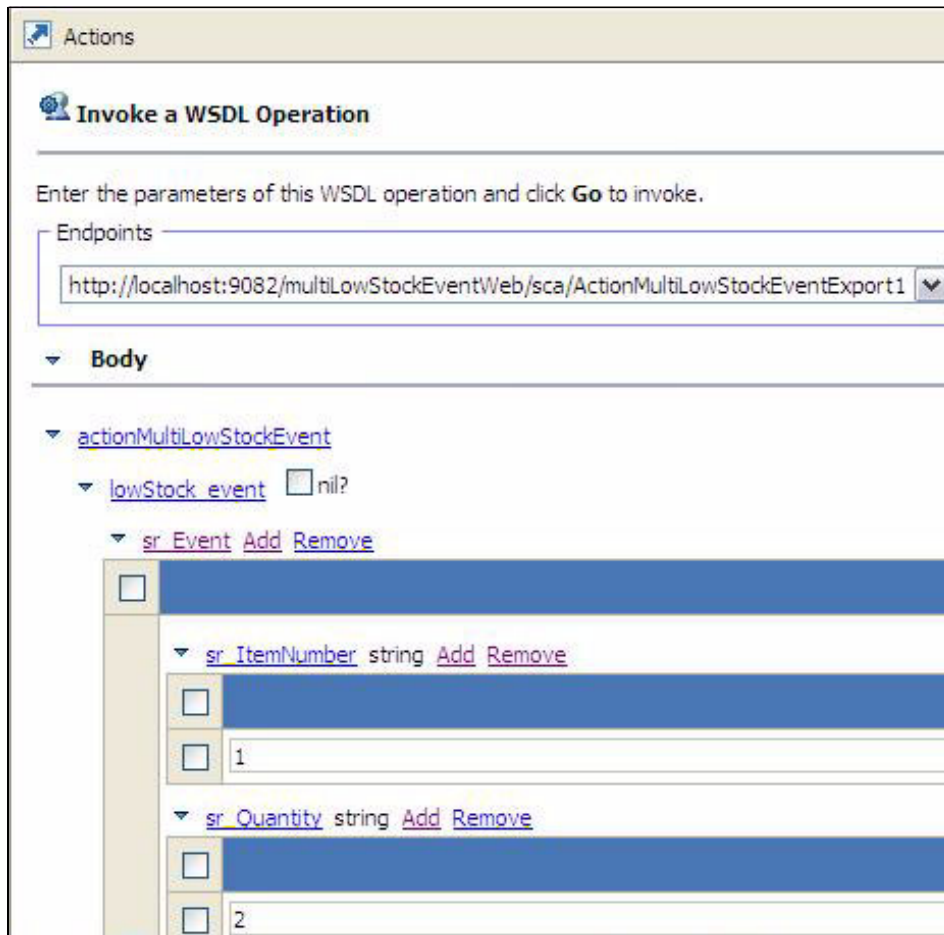


Figure 11-29 Setting values for a Web Service Call

In Figure 11-29, after we have entered values in all fields, we click **Go**, which is located at the bottom of the window to invoke the process through a Web Service call. There is another panel labelled **Status**, which shows the result of the Web Service call. If the Web Service call works successfully, nothing is returned at this stage, because the process does not yet return any value. We can check the `SystemOut.log` file of the server to find the output that was produced by the Java code of the Snippet activity, which is similar to the output of testing the process using Business Process Choreographer Explorer.

Testing with Integrated Test Client

We can test the processes on the Assembly Diagram view by using the Integrated Test Client. We start by selecting the process in the assembly diagram, right-click it, and select the option called Test Component, as shown in Figure 11-30.

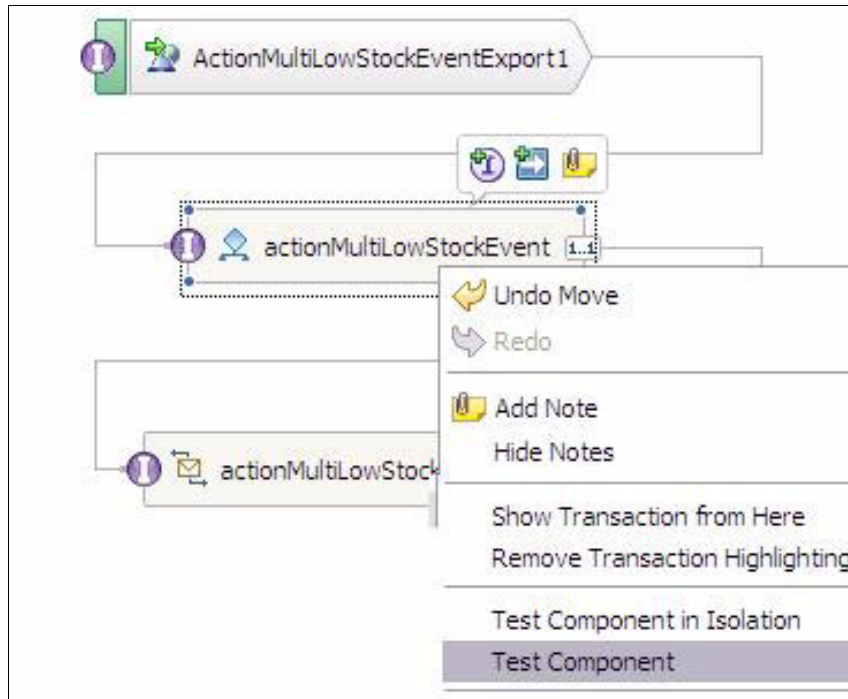


Figure 11-30 Starting Integrated Test Client

We show how to use Integrated Test Client in detail in 11.2.26, “Testing using the Integrated Test Client” on page 366 to test the completed process.

11.2.10 Adding a web service call to get item details

In our process design, we want our process to be able to perform a web service call to obtain details about the low stocked item, such as its cost and description. In this case, our process performs a web service call to CICS. We right-click Web Service Ports and select Import. In the window that opens, we select the Web Services Description Language (WSDL) and XML Schema Definition (XSD) source, as shown in Figure 11-31 on page 325.

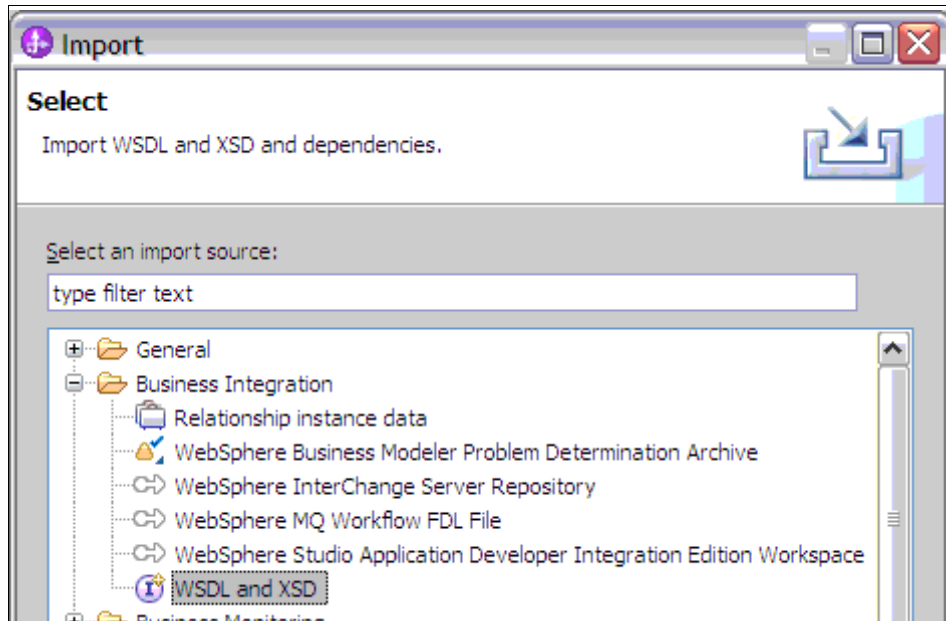


Figure 11-31 Start of process to import wsdl

We click Next. In the next window that opens, we select the option to import from the local WSDL or XSD file and click Next. We click Browse in the next window to locate the source directory folder containing the wsdl file supplied. When we select the folder, WebSphere Integration Developer V7 displays all the wsdl files available, as shown in Figure 11-32.

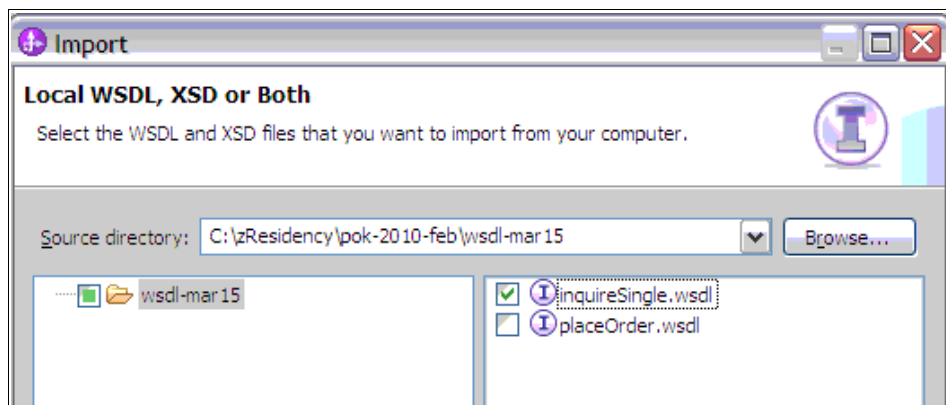


Figure 11-32 Import the wsdl that is used to call the web service to get the item details

We select inquireSingle.wsdl and click Finish. After WebSphere Integration Developer V7 completes the import of the wsdl, there is a new entry called DFH0XCMNPort under Web Services Port, a new interface called InquireSinglePort under Interfaces, and two new Business Objects under Data Types called InquireSingleInput and InquireSingleOutput.

Testing the web service

Because our process calls this web service, we need to determine what data needs to be passed when making the web service call and what data needs to be received in return. The best way to determine the data to pass and receive is to invoke the web service by using WebSphere Integration Developer V7. We right-click DFH0XCMNPort and select Web Services → Test with Web Services Explorer.

We invoke the web service and note which input fields must be filled in and which output fields in the reply contain the information about the item that we need.

Figure 11-33 shows testing the web service. Notice the red asterisk (*) beside the field ca_return_code. This red asterisk indicates that this field requires an initial value.

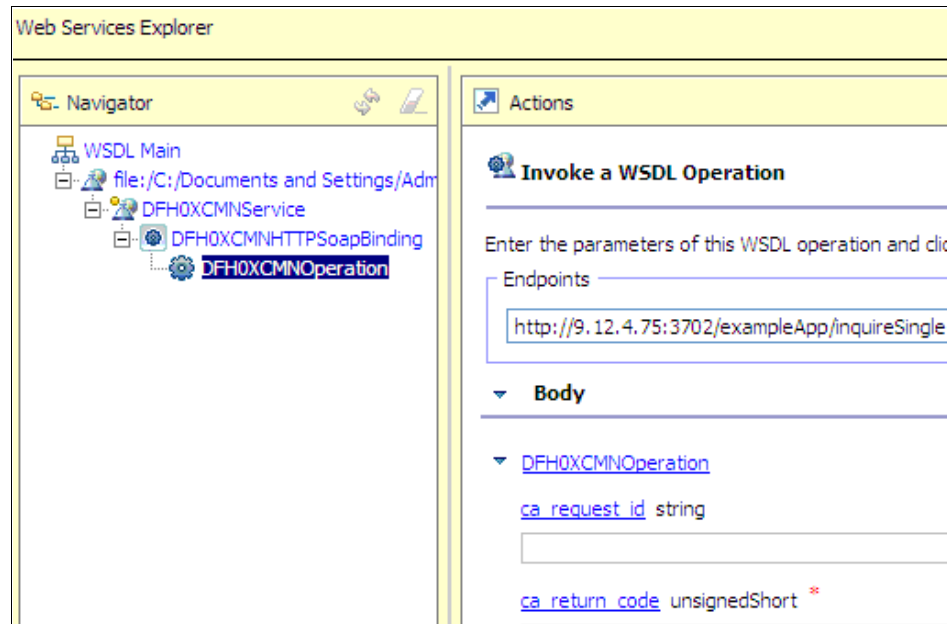


Figure 11-33 Testing the web service

When we test the web service, we enter the values that are shown in Table 11-1 in the input fields.

Table 11-1 Testing values

Field	Value
ca_request_id	01INQS
ca_return_code	0
ca_item_ref_req	30
ca_sngl_item_ref	30
ca_sngl_department	100
in_sngl_stock	0
on_sngl_order	0

Figure 11-34 on page 328 shows the data that was returned as a result of the web service call to CICS.

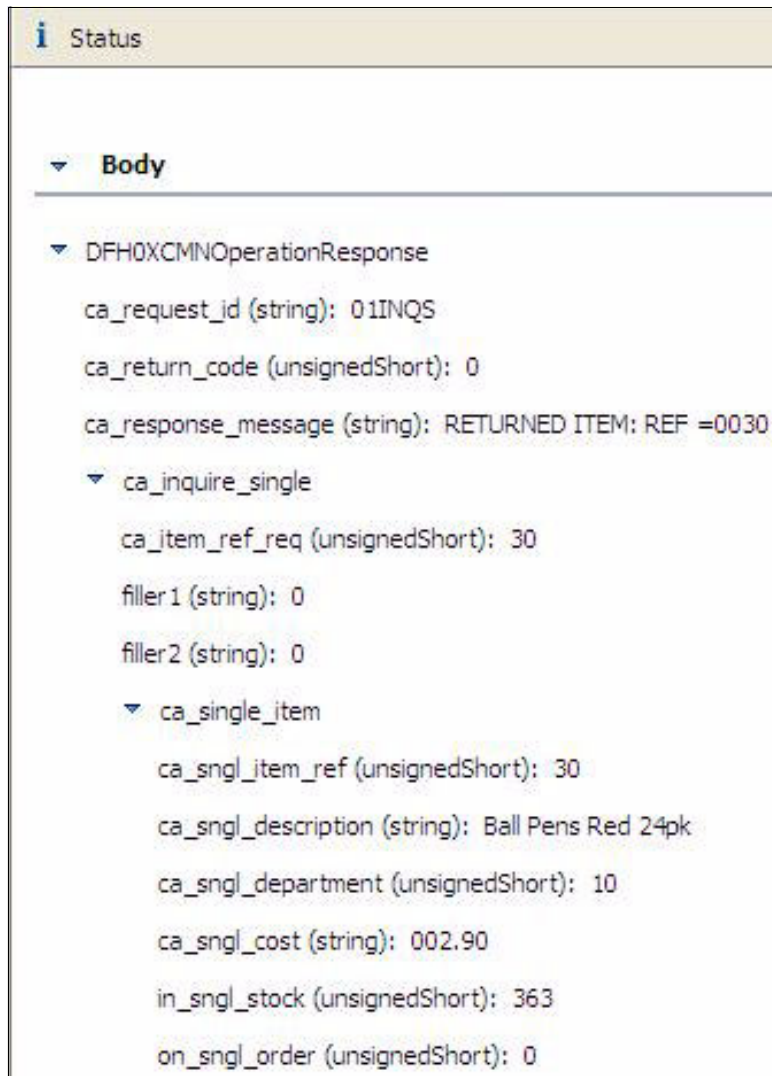


Figure 11-34 Data returned from web service call to CICS

The WSDL contains the TCP/IP address and the port of the CICS region that handles this Web Service call. We can see the TCP/IP address and the port by double-clicking DFH0XCMNPort, selecting the DFH0XCMNService icon in the wsdl display, selecting the DFH0XCMNPort within that display, and then selecting the Properties tab. The Address field shows the TCP/IP address and the port of the target CICS region, as shown in Figure 11-35 on page 329. We can change the TCP/IP address and port here, if necessary.

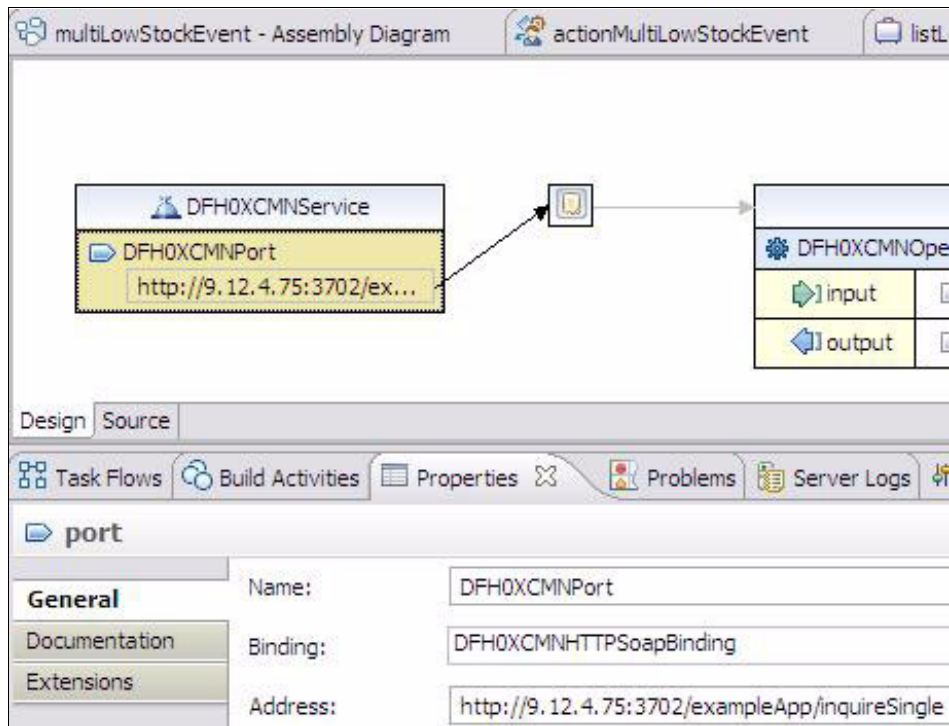


Figure 11-35 CICS WSDL address

11.2.11 Adding the ForEach activity

As part of our process design, we want to call the web service to get details about each low stocked item. The event that is passed by WebSphere Business Events has three low stock item events, and we use a ForEach activity to process each low stock item event.

In the Assembly Diagram view, we click the ForEach activity under Structures and drag it between the snippet and reply activity. We select the ForEach activity, click the Properties tab, click Description, and change the Display Name to calcCostLowStockEvent. We click Details and click Type. In the small window, we select sr_event, as shown in Figure 11-36 on page 330.

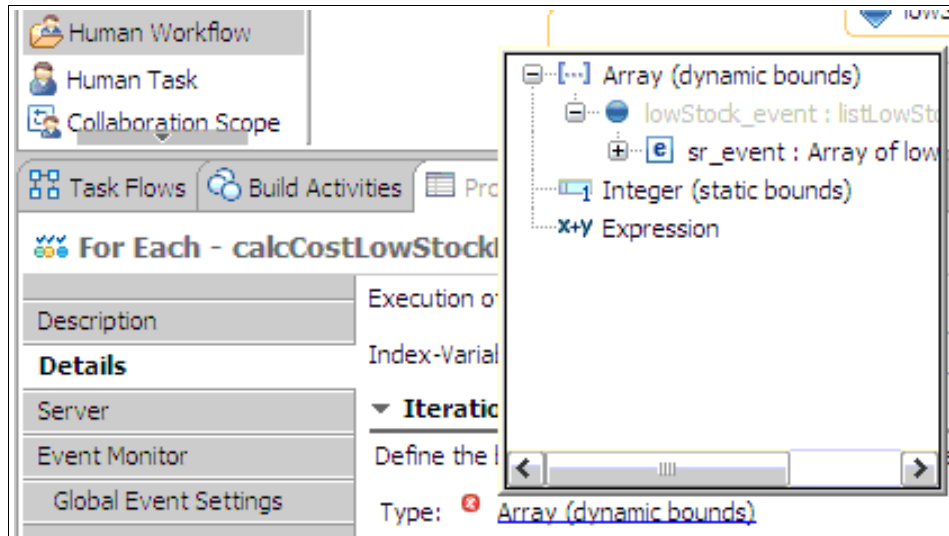


Figure 11-36 Setting bounds for the ForEach activity

The process then looks similar to Figure 11-37 on page 331.

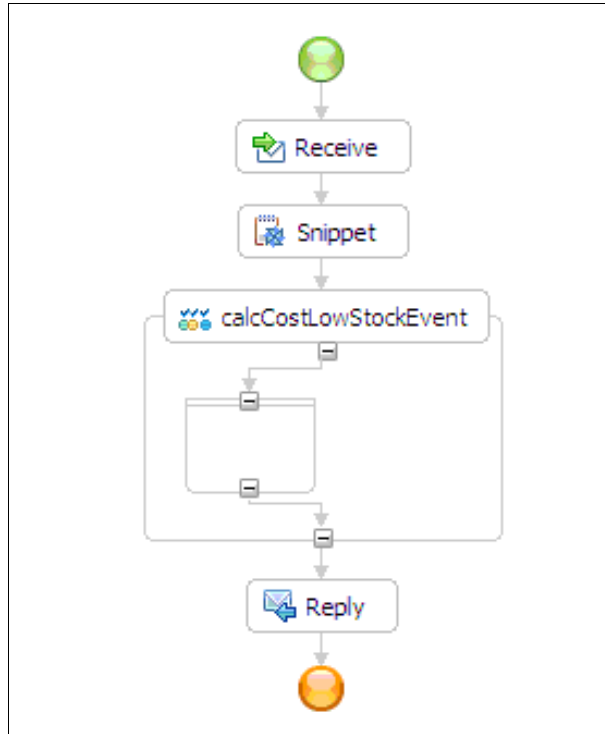


Figure 11-37 Process with ForEach activity added

11.2.12 Defining new Business Objects

Our process design shows that we want the process to invoke a human task. When the process creates the human task, we want to pass the following data:

- ▶ Data that the process received
- ▶ Details, such as cost and description, about each low stock item
- ▶ Total cost of the low stock event

In the ForEach activity, after each web service call to get item details, we need to store these details somewhere. We create new business objects to store this data.

itemDetails

We right-click Data Types, select New → Business Object, and then, set the Name to itemDetails.

Using the approach that is described in 11.2.5, “Defining business objects” on page 301, we add the following fields (all string type):

- ▶ `item_refNumber`
- ▶ `item_desc`
- ▶ `item_cost`
- ▶ `item_dept`
- ▶ `item_stock`

itemDetailsArray

We define a new Business Object called `itemDetailsArray` and a field called `items` of type `itemDetails`. In the properties for this field, we set the minimum and maximum occurrences to 3.

Important: If you do not set the minimum and maximum occurrences for this field, you get an exception in the Java code that is added, as shown in 11.2.19, “Adding the snippet to calculate the cost” on page 351.

11.2.13 Adding a global variable to hold the total cost

The purpose of the `ForEach` activity is to calculate the total cost of the three low stock events. This value is needed by other activities that we will add after adding the `ForEach` activity, so we need a new global variable to hold this value.

As described in “Adding and deleting global variables” on page 313, we add a new global variable called `totalCostLowStockEvent`. We set the type to float and the initial value of this variable to 0 (zero), as shown in Figure 11-38 on page 333.

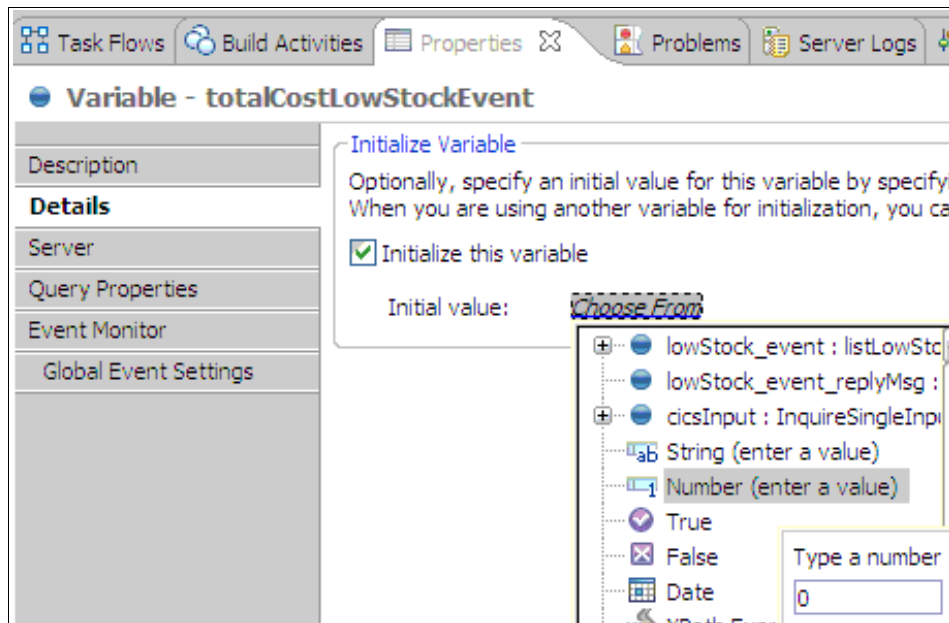


Figure 11-38 Setting the variable's initial value to zero

11.2.14 Adding the Assign activity

We next add an Assign activity to the ForEach activity. We use the Assign activity to extract fields from the data passed into the process into variables in the process.

Defining three local variables

We click the box inside the ForEach activity to set the scope in which the local variables will be created. To add a local variable, click the small symbol to the left of the green plus symbol (+) in the Variables area, as shown in Figure 11-39 on page 334. We define three local variables of type string called lowItemNumber, lowItemQuantity, and lowItemDept.



Figure 11-39 Adding local variables

Adding the Assign activity

In the assembly diagram, we click the Assign activity under Basic Actions and drag and drop it onto the box inside the ForEach activity. We select the Assign activity, click the Properties tab, click Description, and change the Display Name to getLowStockInfo.

Mapping input data to the local variables

In the Properties tab for the Assign activity, we select Details so we can specify that we want to copy data from one location to another location. We click Select From and select XPath Expression, which opens a window called XPath Expression Builder, as shown in Figure 11-40.

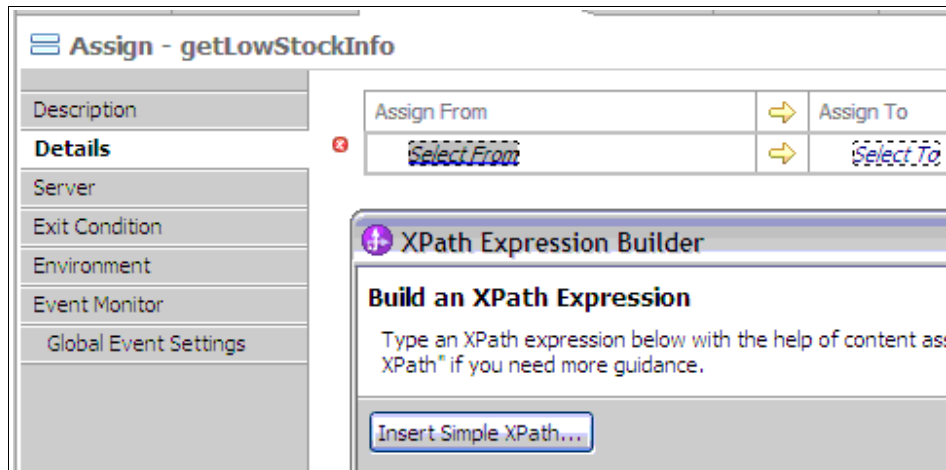


Figure 11-40 Setting up the Assign activity

We click Insert Simple XPath, which opens a window in which we can build the expression.

Creating the expression is not simple. Figure 11-41 shows the window with our completed XPath expression:

```
$lowStock_event/sr_event[position() = $Index]/sr_ItemNumber
```

We use this sequence to create this expression:

1. Expand the entries under \$lowStock_event.
2. Click sr_event, which results in the expression having a value of \$lowStock_event/sr_event.
3. In the “Add an optional filter” area under the Where column, click the drop-down symbol and select position(). Then, enter the value \$Index in the Value column, click anywhere in the window, and the expression will then have a value of \$lowStock_event/sr_event[position() = \$Index].
4. Click sr_ItemNumber to add that component to the expression.

After completing this process, the Expression Builder window looks similar to Figure 11-41.

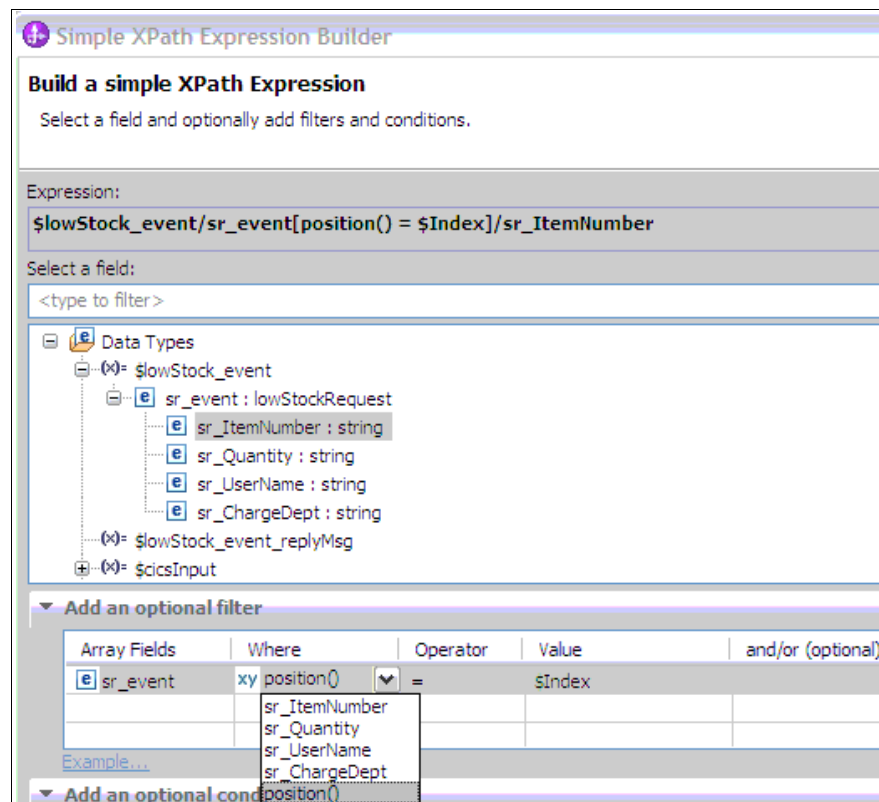


Figure 11-41 Building the XPath expression

We click Ok until we return to the Assign Properties tab. Under the Assign To column, we click Select To and select the local variable lowItemNumber.

We click Add to add another entry and repeat the previously described process, this time, selecting sr_Quantity in the Expression Builder window and assigning it to the local variable lowItemQuantity. Then, we repeat the process again to assign sr_ChargeDept to the local variable lowItemDept.

Figure 11-42 shows the end result.

Assign From	⇒	Assign To
<code>\$lowStock_event/sr_event[position() = \$Index]/sr_ItemNumber</code>	⇒	lowItemNumber
<code>\$lowStock_event/sr_event[position() = \$Index]/sr_Quantity</code>	⇒	lowItemQuantity
<code>\$lowStock_event/sr_event[position() = \$Index]/sr_ChargeDept</code>	⇒	lowItemDept

Figure 11-42 Final setup of the Assign activity

11.2.15 Adding an intermediary interface

In our process, after copying the item data to local variables, we now want to invoke a web service to get details about the item. In a process, you do not call the web service directly. Instead, you call what is called a *reference partner* using an invoke activity.

Important: The reference partner is another key point in understanding how processes are created.

The reference partner in the business process represents an external service with which you want the process to interact. In the assembly diagram, you connect the reference partner to the web service that you want to call.

We also might add the interface of the web service that we want to invoke as a reference partner; however, this approach is not considered a best practice for a few reasons.

First, using the web service interface directly ties the process to the structure of the web service. The purpose of the process is to get information about the item, and ideally, the way that the process gets this information needs to be as abstract as possible. If the process uses the web service interface directly, if at a later time, you decide to use a separate web service that requires a separate input, you must change the invoke activity in the process.

Also, to use this approach, we must set up a Java snippet to set up the input that is required by the web service, because many fields on the web service need to be initialized.

Using an intermediary interface avoids these issues. It provides an interface that requires only key pieces of data and gets back a business object with the required details. Using an intermediary interface does not require any Java snippet code to set up the input that is needed by the interface. Later in 11.2.17, “Updating the mediation flow” on page 344, we see how to use a mediation flow in the assembly diagram to set up the input that the web service requires, using the data that is passed to our intermediary interface.

The bottom-up approach in WebSphere Integration Developer V7 creates the required interface by right-clicking Interfaces in the Project view and selecting New → Interface.

Using the top-down approach though takes fewer steps. We describe the top-down approach. In the Assembly Diagram view, we select the `actionMultiLowStockEvent` and, then, click the Add Reference icon, as shown in Figure 11-43.

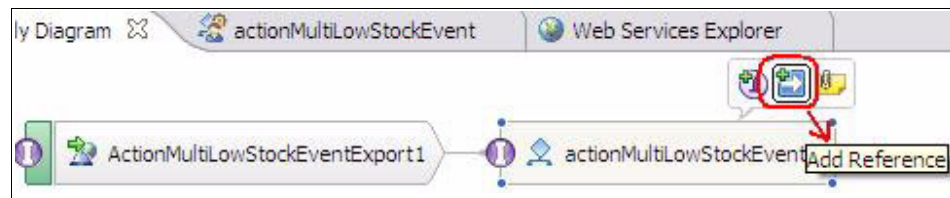


Figure 11-43 Adding a reference partner

WebSphere Integration Developer V7 opens a window titled Add Reference. We click New, which results in WebSphere Integration Developer V7 displaying a new window. We type `getItemDetails` in the name field, as shown in Figure 11-44 on page 338, and then, click Finish.

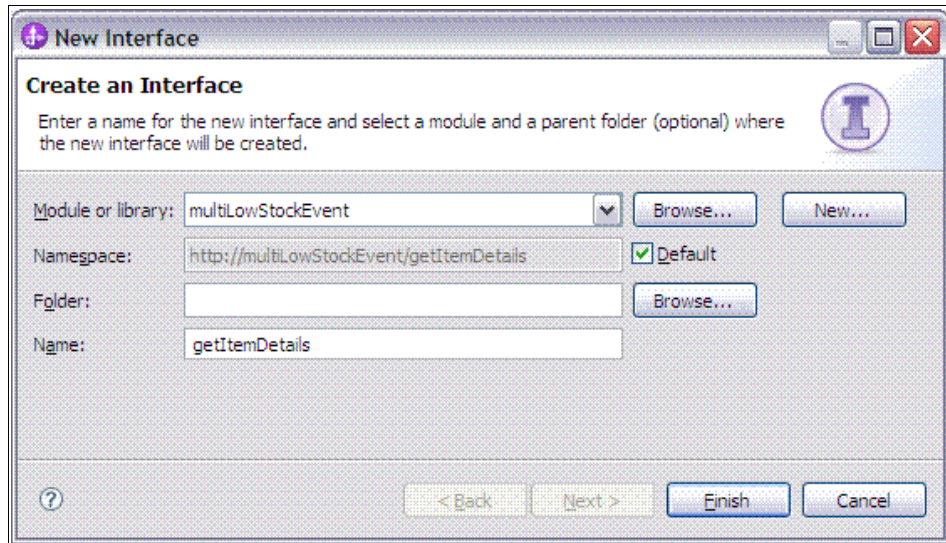


Figure 11-44 Adding an intermediary interface

The Add Reference window looks similar to the window that is shown in Figure 11-45 on page 339.

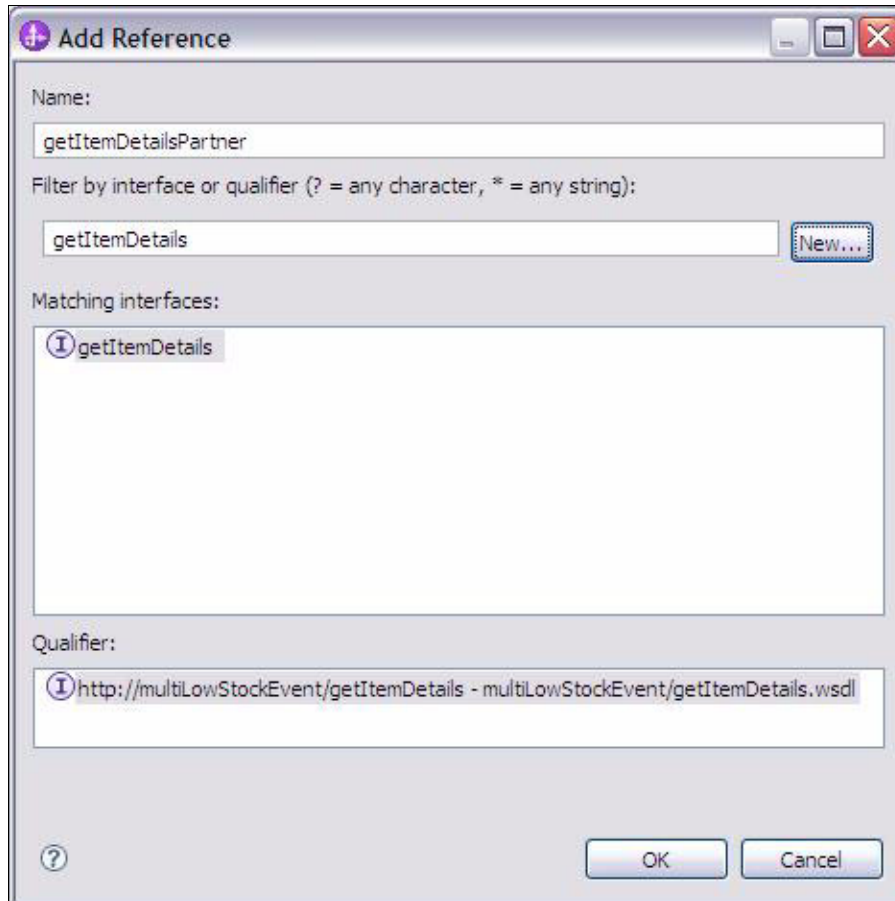


Figure 11-45 After creating a new interface

We click OK, and WebSphere Integration Developer V7 then opens a tab where we can modify the interface. We click the Add Request Response Operation icon, and we set the operation name to `obtainItemDetails`. Then, we click the Add Input icon to add a second input field. We rename the input fields to `itemNumber` and `itemDept`. We rename the output field to `itemInformation`, and we set the type to `itemDetails`. The interface looks similar to the interface that is shown in Figure 11-46 on page 340. We close the tab to save the new interface.

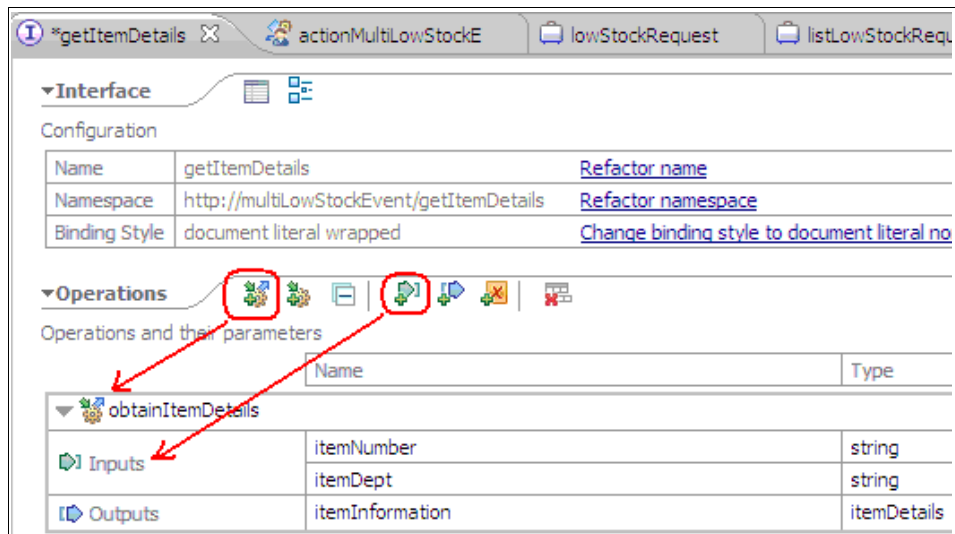


Figure 11-46 Setting input and output parameters on the new interface

In the Assembly Diagram view, if we hover our mouse over the right side of the actionMultiLowStockEvent process, a small box is displayed showing the details about the reference partner that we have just defined.

We click the save icon to save this change and notice that a small red cross now appears, which indicates that the underlying process does not know about the reference partner that we have added in the assembly diagram. To correct this condition, we right-click the actionMultiLowStockEvent process and select Synchronize Interfaces and References → to Implementation, as shown in Figure 11-47 on page 341.

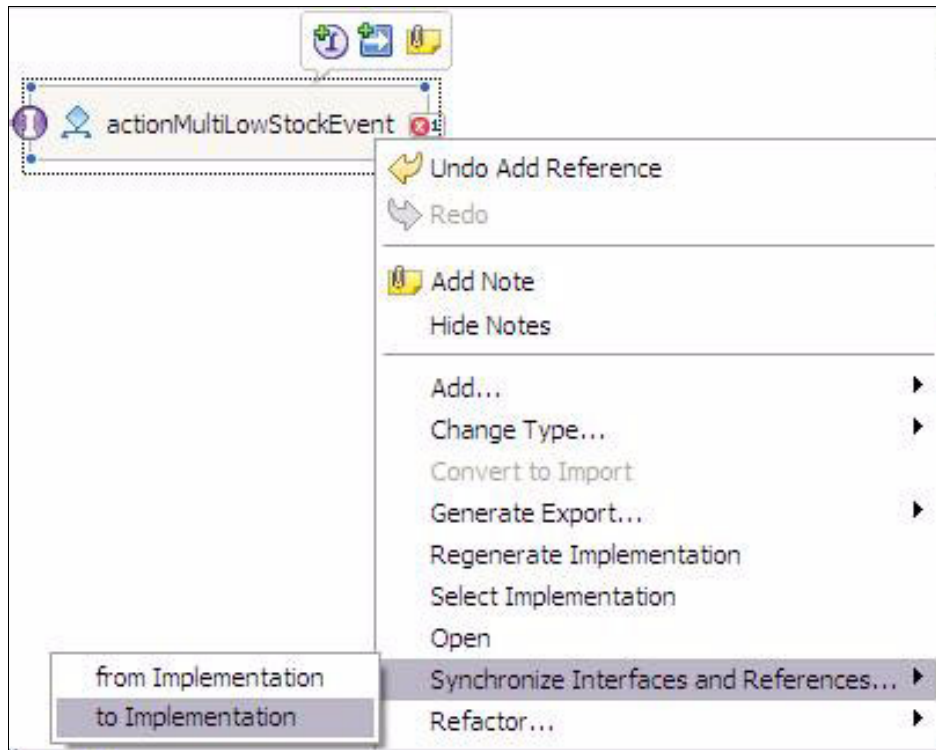


Figure 11-47 Synchronizing to the process implementation

We click Yes in the confirmation box that WebSphere Integration Developer V7 displays. WebSphere Integration Developer V7 updates the process and brings that view into focus. We then see that the getItemDetails interface has been added under Reference Partners.

In the assembly diagram, there is a warning to inform us that, at this stage, the interface that we have added is not connected to anything. We connect the interface next.

11.2.16 Connecting the process to the Web Service

We have defined a reference partner that the process invokes. We now need to connect that reference partner to something that will actually process the call.

We select the interface to the web service called InquireSinglePort and drag and drop it onto the assembly diagram. WebSphere Integration Developer V7 then opens the window that is shown in Figure 11-48 on page 342.

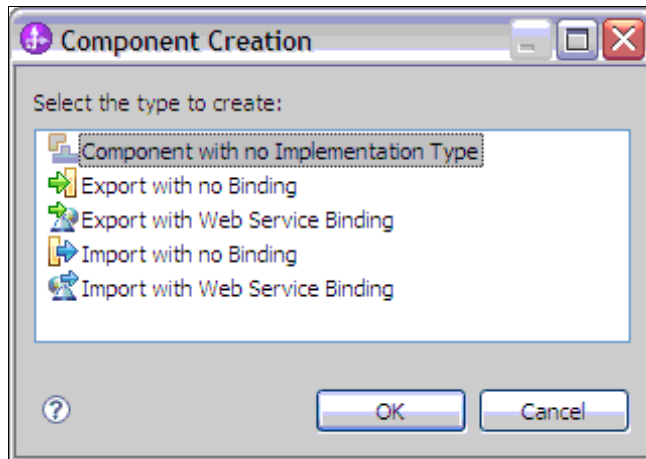


Figure 11-48 Selecting binding type when adding web service interface to assembly

We select “Import with Web Service Binding” and click OK. WebSphere Integration Developer V7 then opens the window that is shown in Figure 11-49. We click Browse, and, in the window that opens, we select DFH0XCMNPort and click OK.

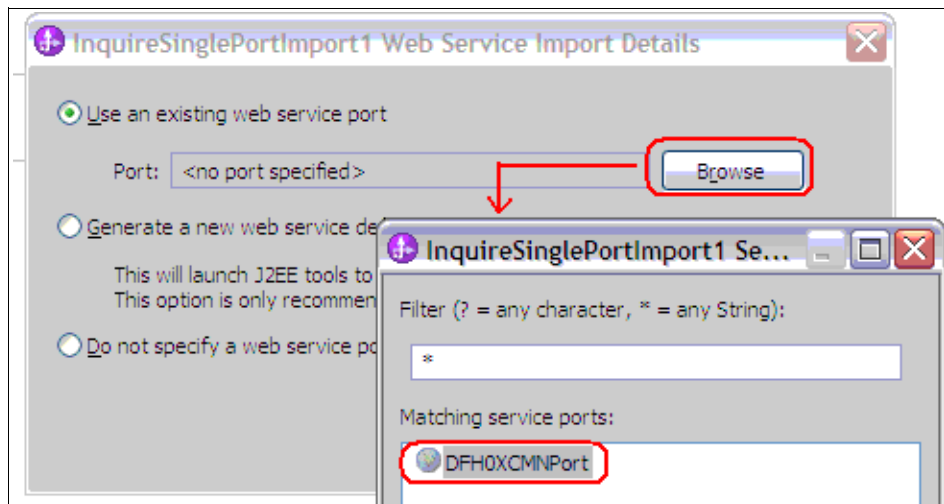


Figure 11-49 Setting up the assembly to call the web service

WebSphere Integration Developer V7 then opens a window asking what transport protocol to use, so we select the default of SOAP1.1/HTTP and click Finish. Then, we click OK to complete the action, which results in adding an Import icon called InquireSinglePortImport1 to the assembly diagram.

Then, we connect the actionMultiLowStockEvent icon to the interface on the Import icon, as shown in Figure 11-50.

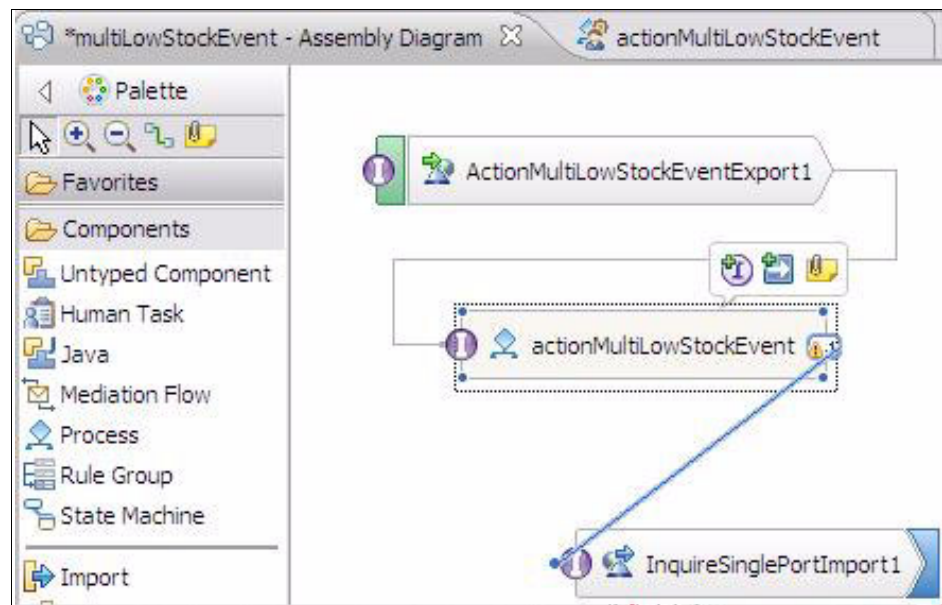


Figure 11-50 Connecting the process to the web service

WebSphere Integration Developer V7 then detects that the interfaces on the objects that you have connected do not match and opens the window that is shown in Figure 11-51.

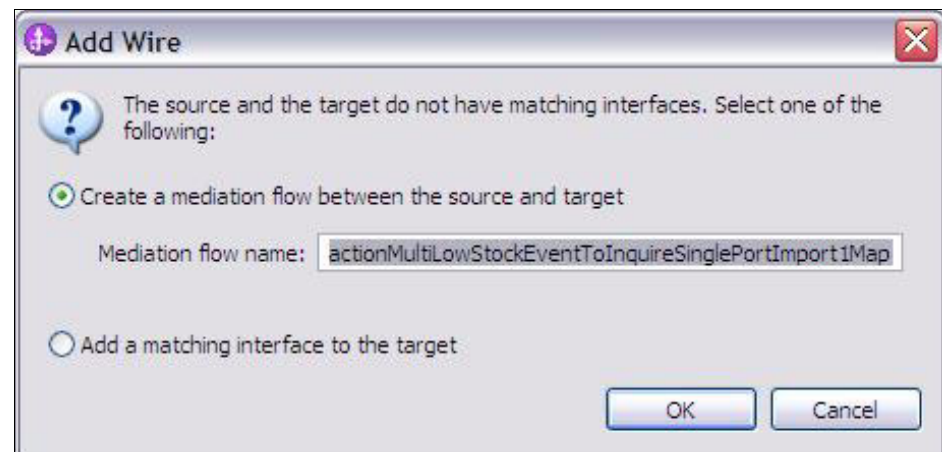


Figure 11-51 WebSphere Integration Developer V7 prompt on action to take due to mismatched interfaces

We accept the default option of “Create a mediation flow between the source and target” and click OK. WebSphere Integration Developer V7 then creates the mediation flow, which shows as an object between the process and the web service in the assembly diagram, as shown in Figure 11-52.

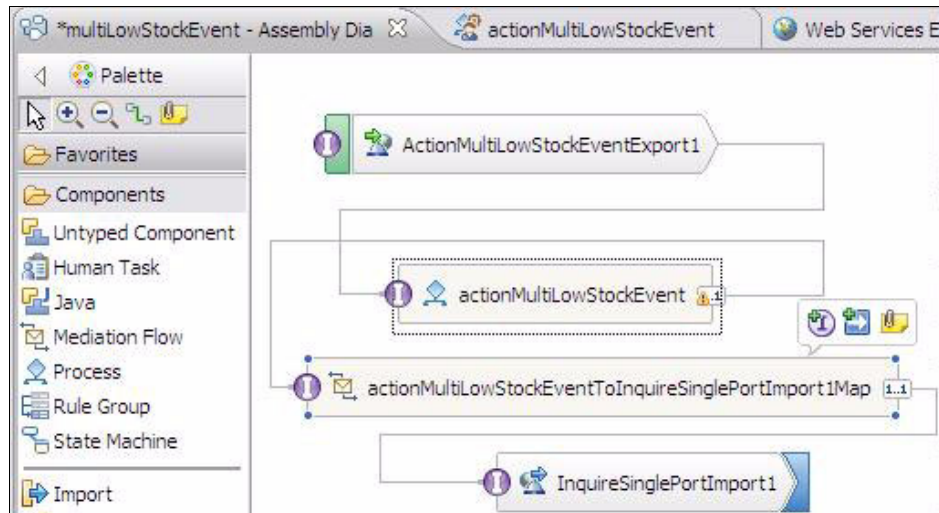


Figure 11-52 Assembly diagram updated with mediation flow

We save the changes to the assembly diagram. Now, we no longer get a warning message that the reference partner on the process does not connect to anything.

11.2.17 Updating the mediation flow

The actual web service that provides the item details requires its input in a certain structure and returns the details in a certain structure. The reason that we set up the getItemDetails interface is that we did not want the process to be tied to the input parameter structure and the output parameter structure of the actual web service. The purpose of the mediation flow is to map the inputs and outputs of our intermediary interface with the inputs and outputs of the web service.

We double-click the mediation flow in the assembly diagram to open it. Figure 11-53 on page 345 shows a slightly edited view of the initial view of the mediation flow.

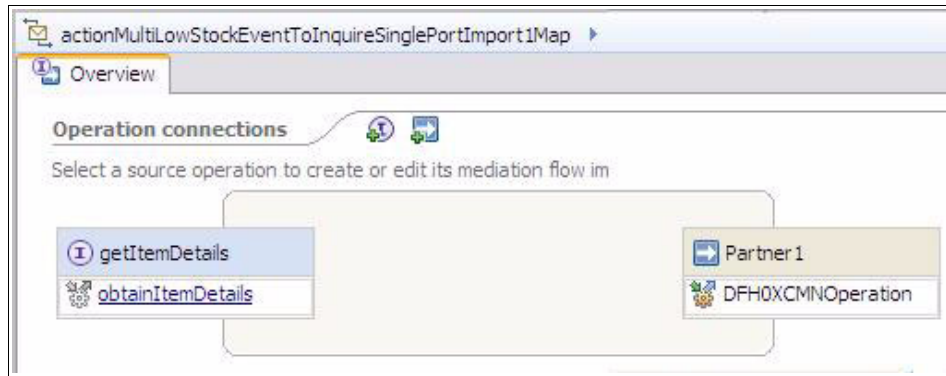


Figure 11-53 Initial display after we have defined the mediation flow

We click `obtainItemDetails` and select `Operation Map` from the list of options displayed. WebSphere Integration Developer V7 then opens a window called `Select Reference Operation`, and we select `DFH0XCMNOperation`, which, in this case, is the only available operation. `DFH0XCMNOperation` performs the web service call to CICS.

WebSphere Integration Developer V7 then opens a window similar to Figure 11-54.

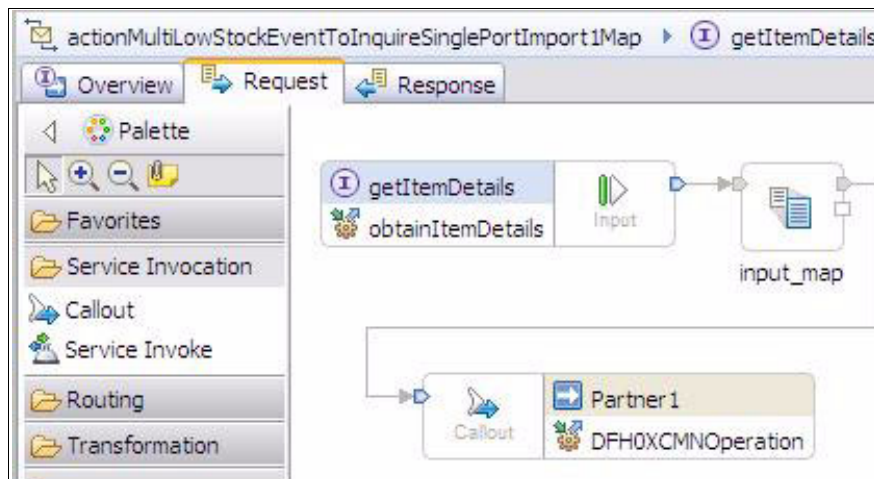


Figure 11-54 Start of process to set up the mapping of the data between interfaces

There is a `Request` tab with an `input_map` icon, which maps how data from the `getItemDetails` interface is passed to the `Partner1` interface. There is also a `Response` tab with an `output_map` icon, which maps how data flows from the `Partner1` interface to the `getItemDetails` interface.

We double-click the input_map icon, and a window called New XML Map opens. We accept the default name and click Finish.

Figure 11-55 shows the initial state of the input map.

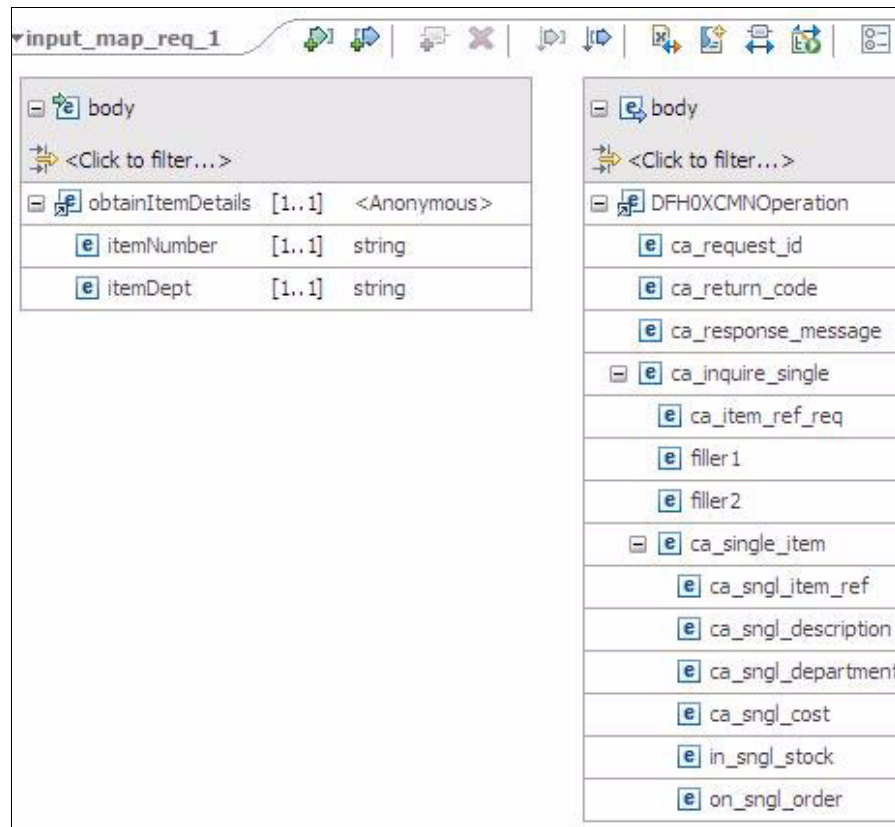


Figure 11-55 Initial state of the input map

Now, we need to connect the fields in the box on the left to the appropriate fields on the right. Also, a number of fields in the box on the right need to be filled in with an initial value. However, the interface that is called by the process does not provide these values. This situation demonstrates the value of using an intermediary interface. Because we use an intermediary interface, we remove the task of setting numerous fields that the web service requires from the actual process. If, in the future, we replace this web service with a separate web service that has other fields, we will merely need to change this input map. We will not need to change the actual process.

To connect a field in the box on the left with a field in the box on the right, we move the mouse over a field on the left so that the field is highlighted, and then,

we drag the small circle to the field on the right, which also becomes highlighted. As we drag the small circle, WebSphere Integration Developer V7 displays a line showing the connection that we make. When we release the mouse, the connection becomes permanent. There is a box in the middle of the connection that is labelled Move with a drop-down arrow beside it. The drop-down arrow, when selected, shows various operations that can be performed on the transfer of the data.

We connect `itemNumber` on the left with `ca_item_ref_req` and `ca_sngl_item_ref` on the right, and we connect `itemDept` on the left with `ca_sngl_department` on the right.

We right-click the field called `ca_request_id`, select the Create Assign option, and then, in the Properties tab, we set the value to 01INQS. We use this same process to assign a value of 0 (zero) to all other fields in the box on the right. We then close the tab.

Next, we click the Response tab and double-click the output map icon. We expand all the fields in the boxes and connect the following fields:

- ▶ `ca_sngl_item_ref` to `item_refNumber`
- ▶ `ca_sngl_description` to `item_desc`
- ▶ `ca_sngl_department` to `item_dept`
- ▶ `ca_sngl_cost` to `item_cost`
- ▶ `ca_sngl_stock` to `item_stock`

On the connections from `ca_sngl_item_ref`, `ca_sngl_department`, and `ca_sngl_stock`, we use the Convert option to convert the data from `unsignedShort` to `string`. Figure 11-56 on page 348 shows the completed map.

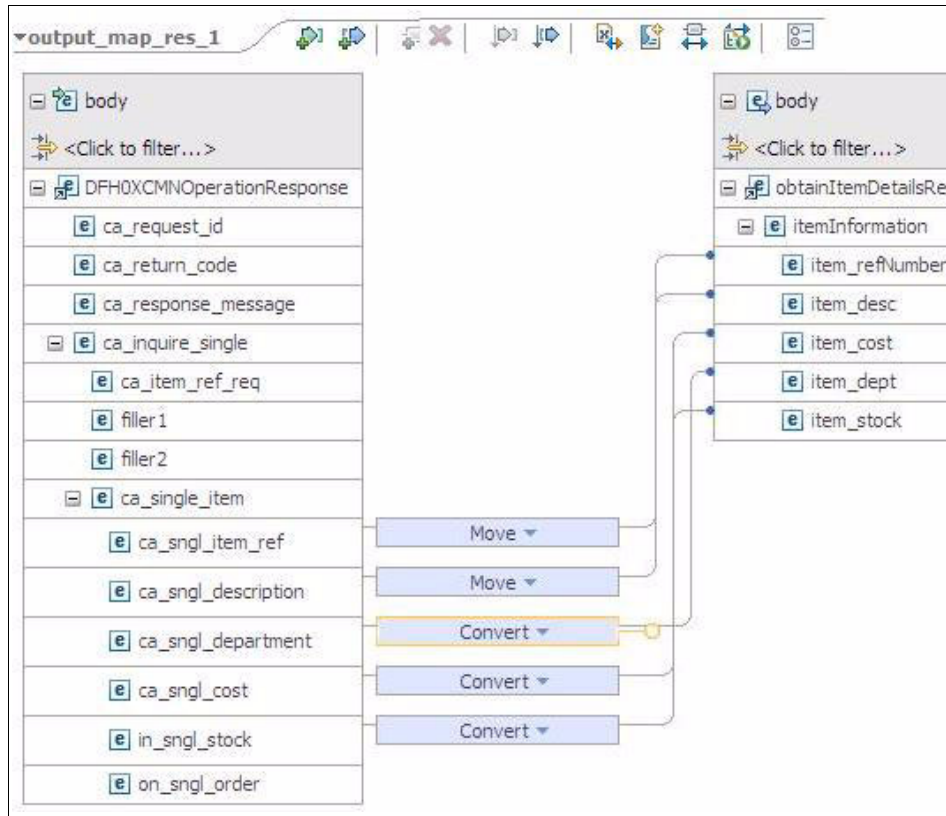


Figure 11-56 Connected output map

We close the tabs to save the mediation flow. Then, we save these changes.

Important: It is important to understand that the diagram that is shown in the assembly diagram does not depict a flow of execution. Instead, it shows how the process is invoked by external parties that want to invoke the process and how the process invokes external services, in this case, the web service call to CICS.

11.2.18 Adding the invoke activity

In the For Each activity, we need to obtain the details about the item from the failed order. We use an invoke activity.

The invoke activity requires a local variable to store the item details that are returned by the activity. We select the box inside the ForEach activity and define

a local variable called itemDetails of type itemDetails. We define a global variable called itemDetailsList of type itemDetailsArray, which will be used to store the itemDetails Business Object that is returned from each web service call.

Next, we add an invoke activity after the Assign activity in the ForEach activity. We name it invokeItemDetailsService.

In the Properties tab, we click Details, click Browse, and select getItemDetailsPartner as the partner reference to invoke, as shown in Figure 11-57.

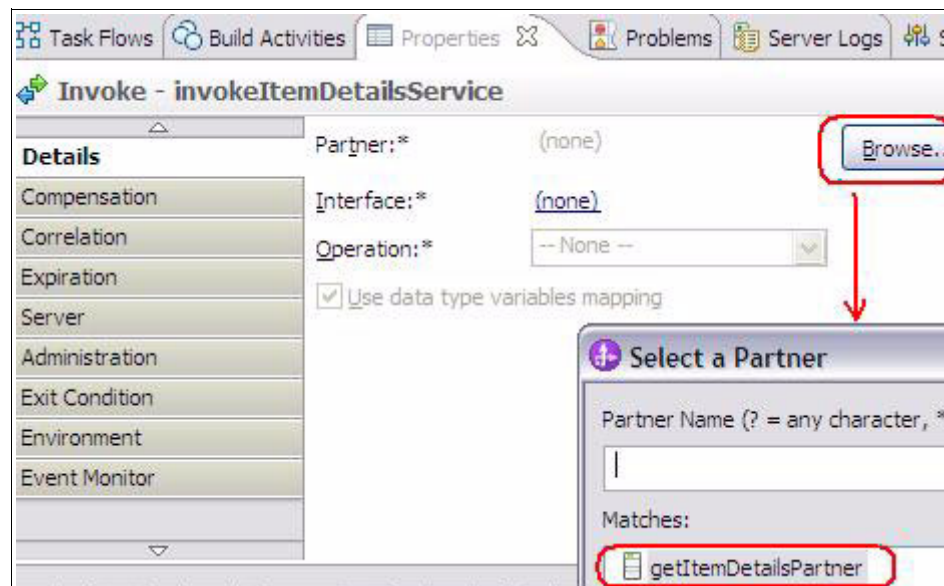


Figure 11-57 Setting the partner reference for the invoke activity to call

Using the drop-down box on the Operation field, we select obtainItemDetails.

Next, we map the variables that are read by the invoke activity for input, and we map the variable in which the invoke stores the result upon return, as shown in Figure 11-58 on page 350.

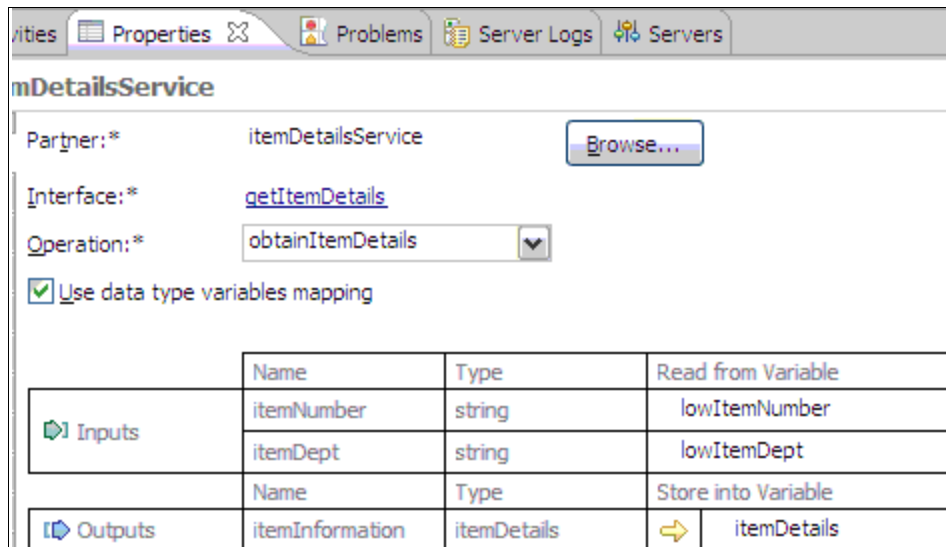


Figure 11-58 Setup of the invoke activity

Figure 11-59 shows our business process after adding this invoke activity.

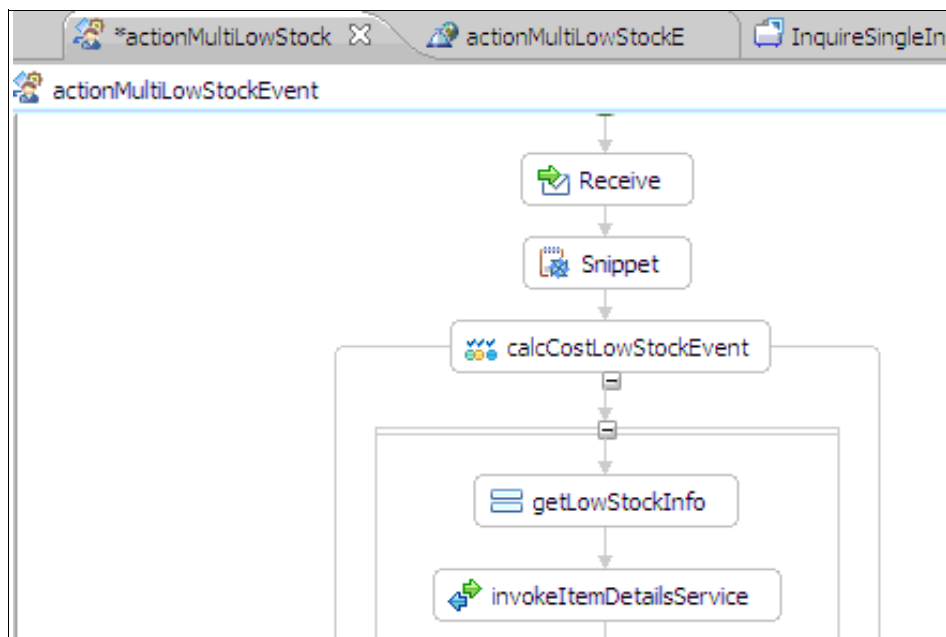


Figure 11-59 State of business process after adding invoke activity

11.2.19 Adding the snippet to calculate the cost

Having obtained information about the item, in particular the cost of the item, we next need to calculate the cost of the failed item order each time through the loop, plus calculate the total cost of the three failed orders. We use a Java snippet to perform these tasks.

We select the snippet activity and drag it into the process flow after the invoke activity and call it `sumCostAndSaveInfo`.

We add a new local variable called `itemCost` of type `float`. We add the Java code that is shown in Example 11-3.

Example 11-3 Java code to calculate total cost of low stock event

```
System.out.println(" ");
System.out.println("----- start post-invoke ");
System.out.println(" ");
System.out.println("Item details returned from webservice call: " +
itemDetails );

itemCost = new Float(itemDetails.getString("item_cost"));

System.out.println("itemCost: " + itemCost);
System.out.println("itemQuantity: " + lowItemQuantity);

totalCostLowStockEvent = totalCostLowStockEvent + ( itemCost *
Integer.parseInt(lowItemQuantity));

System.out.println("total cost order missed now: " +
totalCostLowStockEvent);

System.out.println("Index.intValue(): " + Index.intValue());
System.out.println("itemDetailsList: " + itemDetailsList);

List theLowStockItemList = new ArrayList ();

BOFactory bofactory =
    (BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory")
;
if ( itemDetailsList == null ) {
    itemDetailsList = bofactory.create("http://multiLowStockEvent",
"itemDetailsArray");
} else {
    theLowStockItemList = itemDetailsList.getList(0);
```

```

        System.out.println("in else - existing: theLowStockItemList: "
+ theLowStockItemList);
    }

    System.out.println("after if - current: itemDetailsList: " +
itemDetailsList);
    theLowStockItemList.add(itemDetails);

    System.out.println("after adding itemDetails to: theLowStockItemList: "
+ theLowStockItemList);

    itemDetailsList = bofactory.create("http://multiLowStockEvent",
"itemDetailsArray");
    itemDetailsList.setList(0,theLowStockItemList);
    System.out.println("after setList on: itemDetailsList: " +
itemDetailsList);
    System.out.println(" ");
    System.out.println("----- end post-invoke ");
    System.out.println(" ");

```

11.2.20 Testing the process

We then test our process to verify that the process can successfully invoke the web service by using the process that is described in 11.2.9, “Time for a first test run” on page 318. Example 11-4 shows part of the output from the Java snippet in the ForEach loop.

Example 11-4 Part of the process output

```

Item details returned from webService call: BusinessObject:
itemDetails@6aa16aa1 (item_desc=Green Laser Paper 201b 500/ream
, item_cost=005.35, item_dept=10, item_stock=0)
itemCost: 5.35
itemQuantity: 4
total cost order missed now: 38.870003
Index.intValue(): 3

itemDetailsList: BusinessObject: itemDetailsArray@5d155d15
(items=[BusinessObject: itemDetails@635d635d (item_desc=Ball Pens Blue
24pk
, item_cost=002.90, item_dept=10,
item_stock=0), BusinessObject: itemDetails@1b911b91
(item_desc=Highlighters Assorted 5pk
, item_cost=003.89,
item_dept=10, item_stock=0)])

```

```

in else - existing: theLowStockItemList: [BusinessObject:
itemDetails@635d635d (item_desc=Ball Pens Blue 24pk
, item_cost=002.90, item_dept=10, item_stock=0), BusinessObject:
itemDetails@1b911b91 (item_desc=Highlighters Assorted 5pk
, item_cost=003.89, item_dept=10, item_stock=0)]

after if - current: itemDetailsList: BusinessObject:
itemDetailsArray@5d155d15 (items=[BusinessObject: itemDetails@635d635d
(item_desc=Ball Pens Blue 24pk , item_cost=002.90,
item_dept=10, item_stock=0), BusinessObject: itemDetails@1b911b91
(item_desc=Highlighters Assorted 5pk , item_cost=003.89,
item_dept=10, item_stock=0)])
after adding itemDetails to: theLowStockItemList: [BusinessObject:
itemDetails@635d635d (item_desc=Ball Pens Blue 24pk
, item_cost=002.90, item_dept=10, item_stock=0), BusinessObject:
itemDetails@1b911b91 (item_desc=Highlighters Assorted 5pk
, item_cost=003.89, item_dept=10, item_stock=0), BusinessObject:
itemDetails@6aa16aa1 (item_desc=Green Laser Paper 201b 500/ream
, item_cost=005.35, item_dept=10, item_stock=0)]
after setList on: itemDetailsList: BusinessObject:
itemDetailsArray@4cb24cb2 (items=[BusinessObject: itemDetails@635d635d
(item_desc=Ball Pens Blue 24pk , item_cost=002.90,
item_dept=10, item_stock=0), BusinessObject: itemDetails@1b911b91
(item_desc=Highlighters Assorted 5pk , item_cost=003.89,
item_dept=10, item_stock=0), BusinessObject: itemDetails@6aa16aa1
(item_desc=Green Laser Paper 201b 500/ream , item_cost=005.35,
item_dept=10, item_stock=0)])

```

The output shows that the process has successfully called the web service, which in turn obtained the item information from CICS. Also, the output shows the total cost of the three low stock events that were passed to the process when it was invoked.

11.2.21 Adding the choice activity

Now that our process can determine the total cost of the low stock event, according to our initial process design, we want to test this cost against a limit to then decide which person (manager or clerk) to contact. We add a choice activity to the process. We select the choice activity listed under Structures and drop it into the process in between the ForEach activity and the Reply activity. We name this activity lowStockCostLimitCheck (Figure 11-60 on page 354).

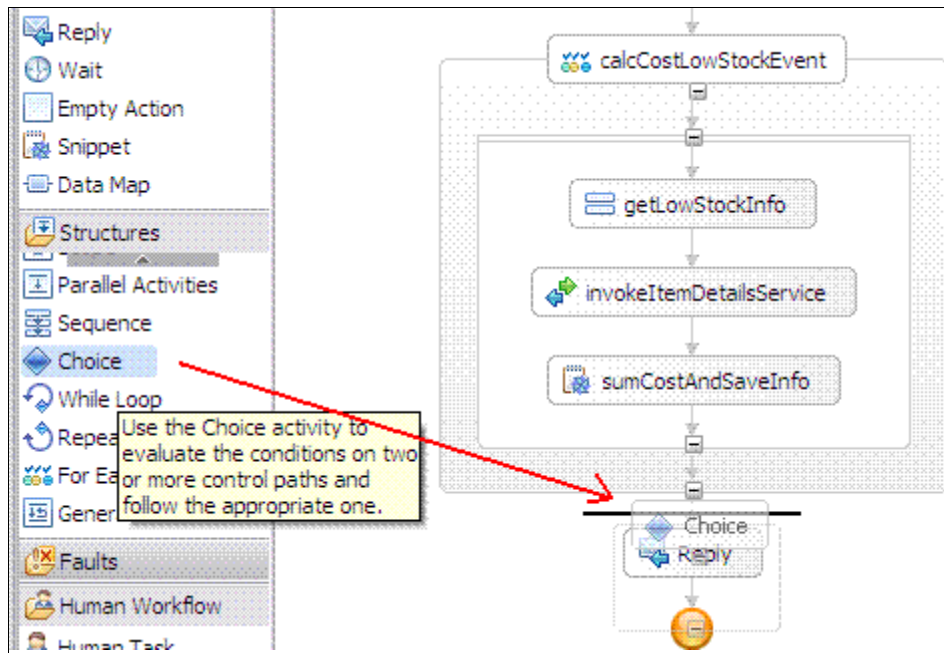


Figure 11-60 Adding the choice activity to the process

We select the choice activity in the process flow. When we hover the mouse over the choice activity, there is an icon that displays the value “Add an Otherwise Element”. We click this value to add an Otherwise activity to the choice activity. We name the choice activity alertManager. We are not allowed to rename the Otherwise activity that we will use to alert the clerk.

Constructing a test condition

We now need to create the logical test to determine whether the total cost of the low stock event is greater than a certain amount. We click the alertManager activity and then select the Properties tab. In the Details area, we select the drop-down arrow beside Expression Language and select Java. Figure 11-61 on page 355 shows how this window looks initially.

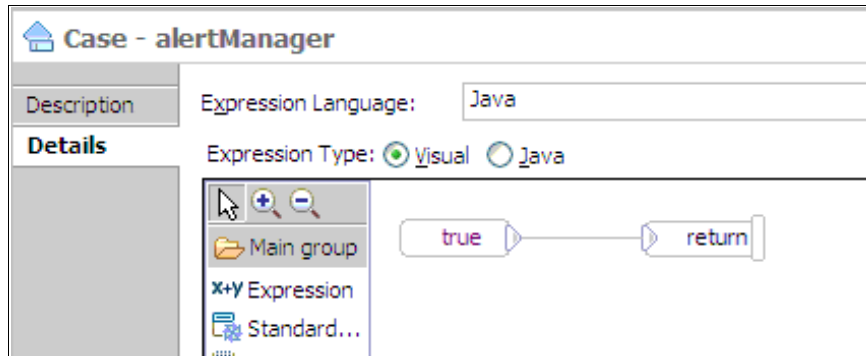


Figure 11-61 Initial state of case expression

We select the true box and delete it. Then, we select the x+y Expression icon and drag and drop it onto the visual area. We click this new icon and click inside it and a drop-down box appears. We select the variable totalCostLowStockEvent, select the greater than (>) symbol, click Number, and enter 100, as shown in Figure 11-62.

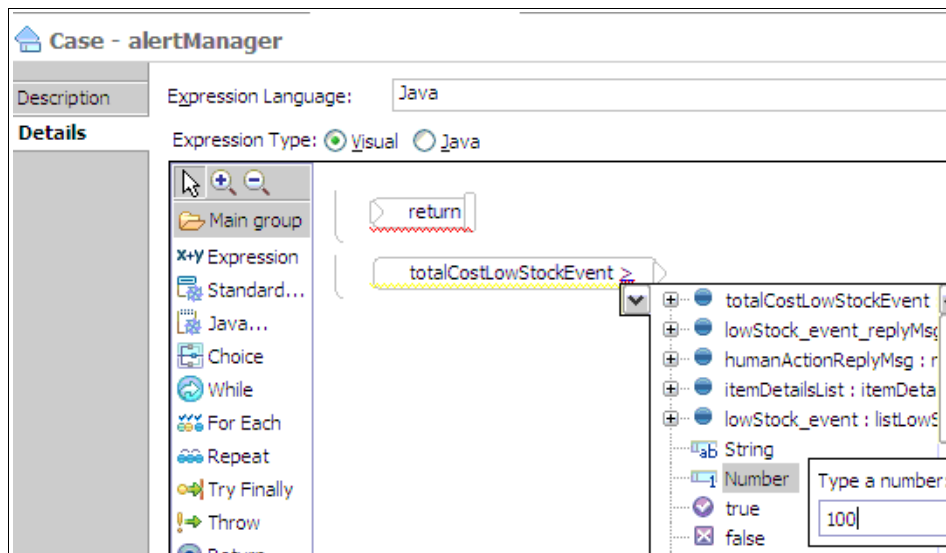


Figure 11-62 Building the conditional expression

Then, we connect the expression to the return icon and save the change. Figure 11-63 on page 356 shows the completed expression.

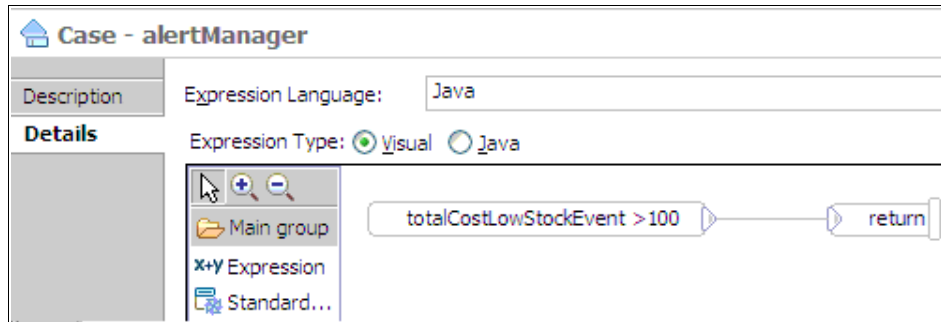


Figure 11-63 Completed conditional expression

Note that we have hard-coded the amount to test against. Alternatively, we can use a variable that had the value set by any means. We see small red crosses in the alertManager and Otherwise icons, because, at this stage, no activity has been added after these parts of the flow. We add this activity next.

Adding Java snippets

We add a snippet after both the alertManager activity and the Otherwise activity. We name one snippet logManagerAlert, and we name the other snippet logClerkAlert. We use each of these snippets to set a reply message to send back to the person who called the process. In the logManagerAlert snippet, we add the Java code that is shown in Example 11-5.

Example 11-5 Java code for alert manager snippet

```
Date now = new Date();
System.out.println("Cost of multiple low stock event: " +
totalCostLowStockEvent );

lowStock_event_replyMsg = "Task assigned to manager for action at: " +
now;
System.out.println(lowStock_event_replyMsg);
```

In the logClerkAlert snippet, we add the Java code that is shown in Example 11-6 on page 357.

Example 11-6 Java code for alert clerk snippet

```
Date now = new Date();
System.out.println("Cost of multiple low stock event: " +
totalCostLowStockEvent );
lowStock_event_replyMsg = "Task assigned to clerk for action at: " +
now;
System.out.println(lowStock_event_replyMsg);
```

11.2.22 Adding a reply activity

Next, we move the reply activity inside the choice activity after the snippet activity called `logManagerAlert`, and we rename it to `replyManagerAlerted`. Then, we select the reply activity, right-click it, and select `Copy`. Then, we select elsewhere in the Choice activity and click `paste`. Then, we move it under the snippet called `Otherwise` activity and rename it to `replyClerkAlerted`.

Figure 11-64 shows how the choice activity looks at this stage.

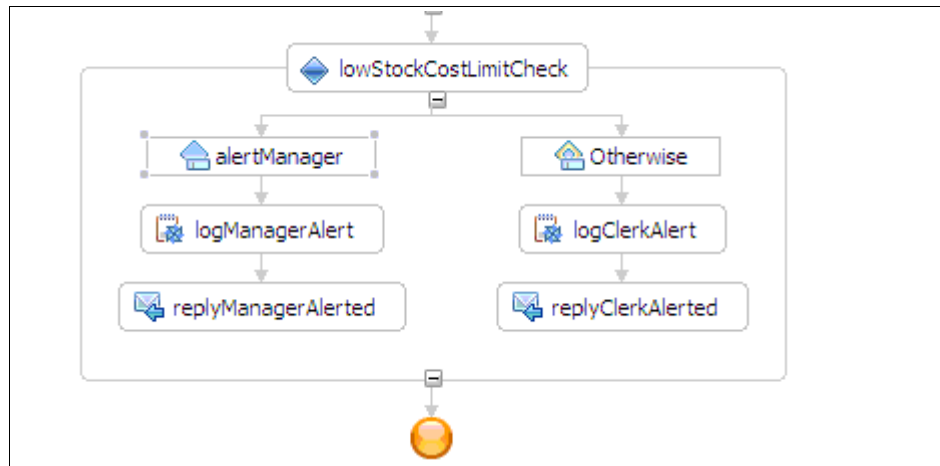


Figure 11-64 Setup of the choice activity

The presence of these reply activities results in a reply being sent to the person who called this process when this stage of the process is reached. The process flow still continues after it performs the reply activity.

Important: We have set up our process to be invoked through a Web Service, which means that the caller is expecting a reply from our process. If the process flow does not have the reply activity at this point, but it merely continues with the human task, the caller of our process does not receive a timely reply. Using the reply activity at this point in the flow means that the caller receives a reply in a timely fashion, while the process can continue on with the human task part.

11.2.23 Adding a human to-do task

Following our design, we want to alert either a clerk or a manager. Therefore, we add human to-do tasks to the process.

When a human task activity is added to a process, we must specify an interface to interact with the human task. This interface specifies the inputs and outputs to the human task.

Adding an interface

We right-click Interfaces and select New → Interface. In the window that opens, we set the name field to alertHuman and click Finish. We click the Add Request Response Operation and set the name of the operation to getLowStockActioned. We set up three input variables:

- ▶ lowStockEvents of type listLowStockRequestEvent
- ▶ lowStockDetails of type listDetailsArray
- ▶ lowStockTotalCost of type float

We set up an output variable called humanActionTaken of type string. The interface then looks similar to the window that is shown in Figure 11-65 on page 359.

▼ **Interface**

Configuration

Name	alerthuman	Refactor name
Namespace	http://multiLowStockEvent/alerthuman	Refactor namespace
Binding Style	document literal wrapped	Change binding style to document literal non-wrapped More...

▼ **Operations**

Operations and their parameters

	Name	Type
▼ getLowStockActioned		
	lowStockEvents	listLowStockRequestEvent
Inputs	lowStockDetails	itemDetailsArray
	lowStockTotalCost	float
Outputs	humanActionTaken	string

Figure 11-65 Setup of interface to the human task

We close the tab to save the changes.

Adding a global variable

We define a new global variable called `humanActionReplyMsg` of type `string` to store the message that is sent back from the human task.

Adding a human task activity

We click the Human Task activity under Human Workflow and drag it under the `replyManagerAlerted` icon, as shown in Figure 11-66 on page 360.

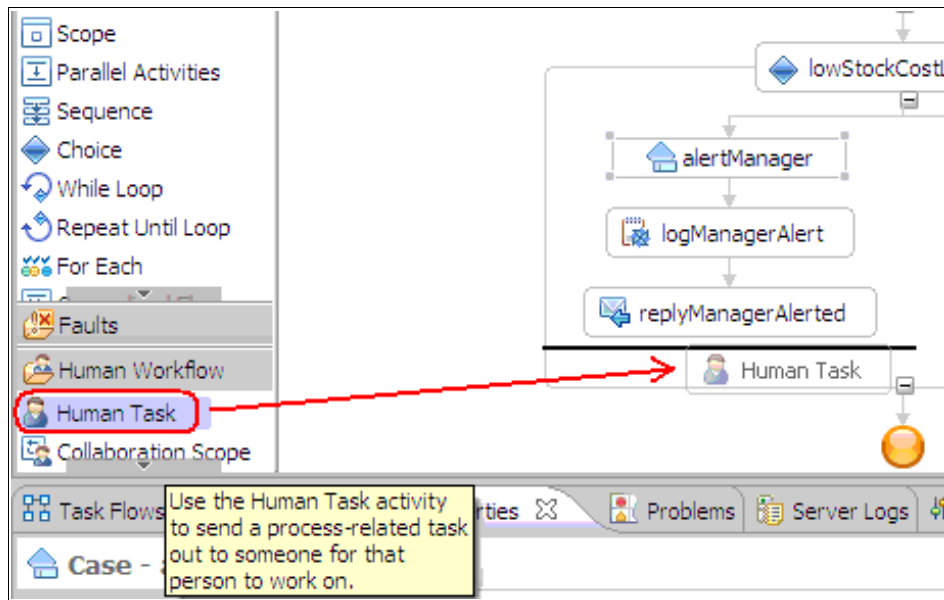


Figure 11-66 Adding a human task to the process

On the window that opens next, we set the name to `assignToManagerForAction`, select the `alertHuman` interface, as shown in Figure 11-67, and click OK.

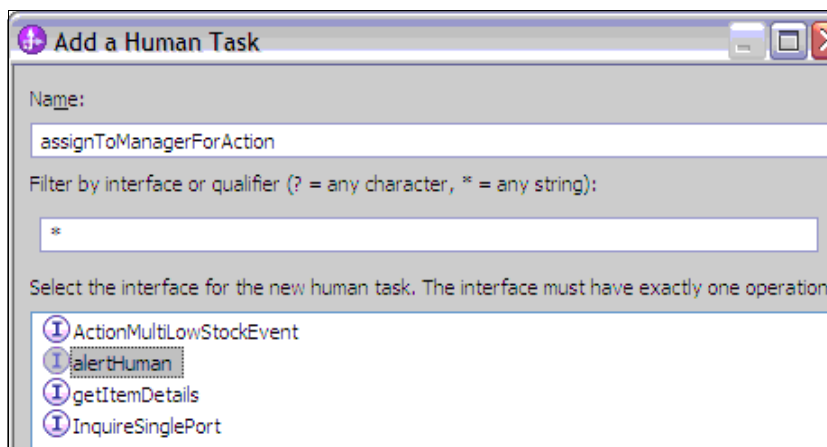


Figure 11-67 Selecting interface to use for human task

A new tab opens for the to-do task where we can perform further customization. We do not need to change anything, so we close this tab. Next, we select the human task activity just added, set its name to `assignToManagerForAction`, click

the Properties tab, and click Details. We set the names of the process variables that are used as input and output for the interface:

- ▶ Inputs:
 - For type listLowStockRequestEvent, use the variable lowStockEvent.
 - For type itemDetailsArray, use the variable itemDetailsList.
 - For type float, use the variable totalLowCostStockEvent.
- ▶ Output:
 - For type string, use the variable humanActionReplyMsg.

The end result is similar to Figure 11-68.

The staff action is implemented by a human task.

Human Task: [assignToManagerForAction](#)

Use data type variables mapping

	Name	Type	Read from Variable
Inputs	lowStockEvents	listLowStockRequestEvent	lowStock_event
	lowStockDetails	itemDetailsArray	itemDetailsList
	lowStockTotalCost	float	totalCostLowStockEvent
	Name	Type	Store into Variable
Outputs	humanActionTaken	string	humanActionReplyMsg

Figure 11-68 Setting the variables that are used with the human task interface

We repeat this process to add a second human task with a name of assignToClerkForAction after the replyClerkAlerted activity.

Additional tasks

For a more complete example, ideally, we define several users that are classified as managers and several users that are classified as clerks, and then, we assign the various human tasks to specific groups of users. In an actual environment, you must perform this additional setup work.

Adding a Java snippet

We add a snippet activity after the choice activity with a name of logHumanActionTaken. We add the Java code that is shown in Example 11-7 on page 362.

Example 11-7 Display action taken by a human being

```
Date now = new Date();  
System.out.println("At: " + now + " Action taken: " +  
humanActionReplyMsg);
```

An alternative approach

We have used a simplistic implementation approach. An alternative approach, which uses the dynamic capabilities of WebSphere Process Server, is to use a *dynamic resolution approach*. In a dynamic resolution approach, a snippet activity can set a variable that is based on the total lost opportunity, which dynamically sets the appropriate staff member to be assigned to the human task. The end result is a simpler process flow.

11.2.24 Completed process

Figure 11-69 on page 363 shows the completed process design.

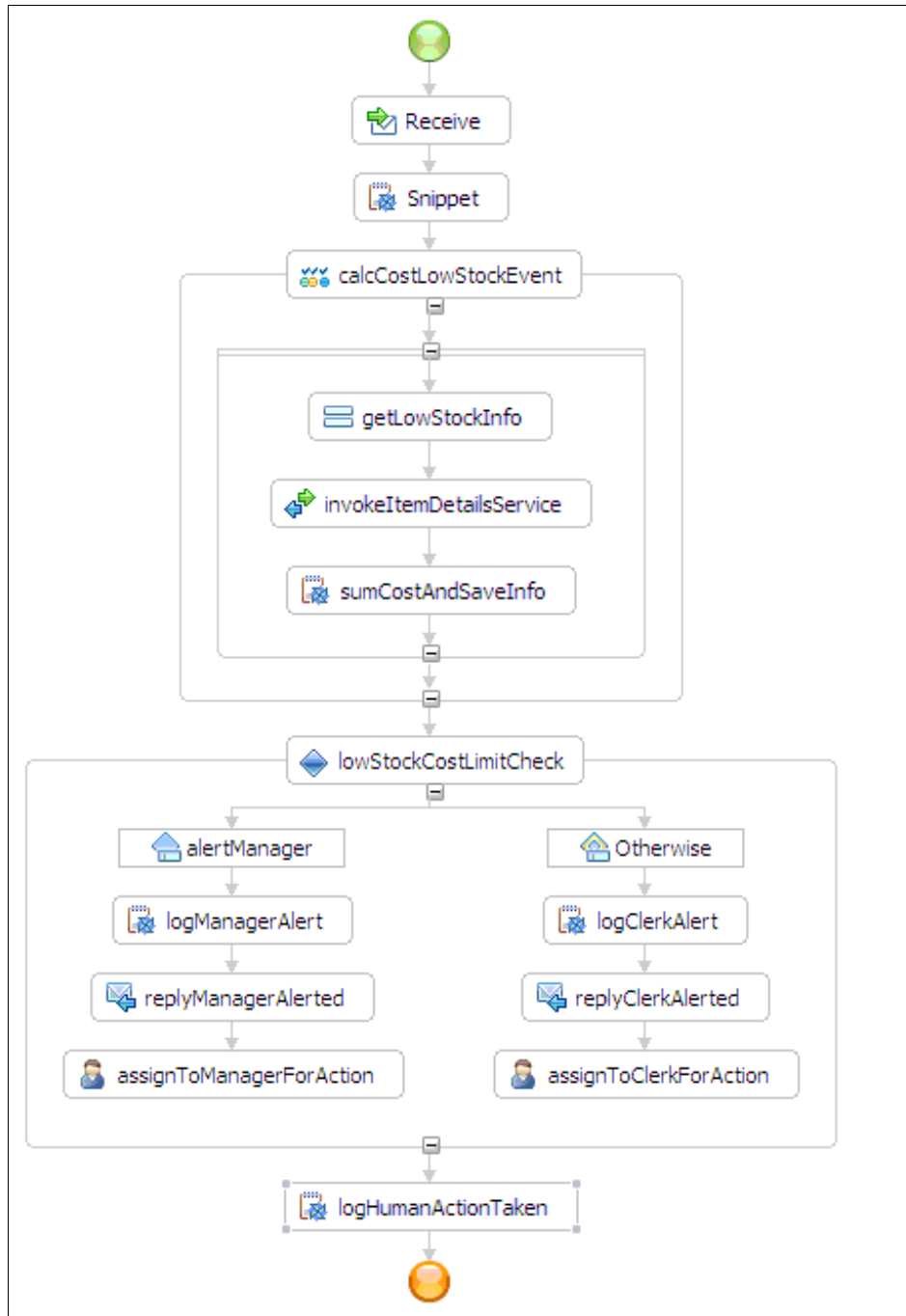
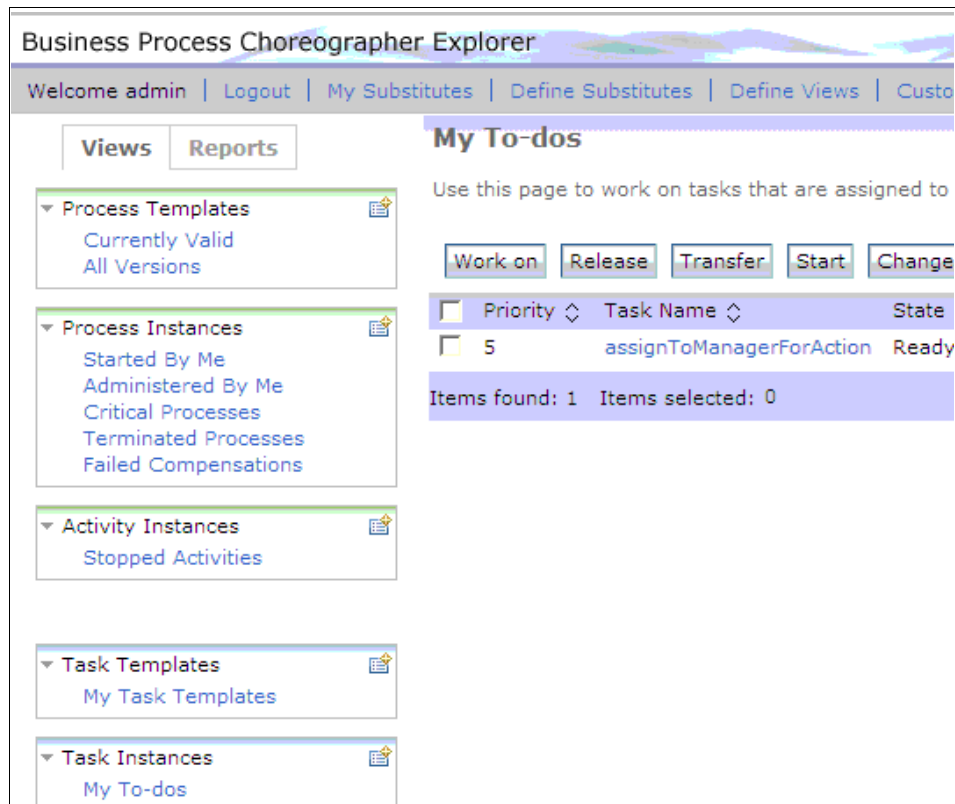


Figure 11-69 Completed process design

11.2.25 Testing the completed process

We then test our process to verify that the process worked as expected. Using the process that is described in 11.2.9, “Time for a first test run” on page 318, we start the Business Process Choreographer Explorer and start an instance of our process. We then enter data in the input fields and submit it.

We then click My To-dos under the Task Instances view and see that a human task called assignToManagerForAction has been created, as shown in Figure 11-70.



The screenshot shows the Business Process Choreographer Explorer interface. The top navigation bar includes 'Welcome admin', 'Logout', 'My Substitutes', 'Define Substitutes', 'Define Views', and 'Custo'. The main area is divided into 'Views' and 'Reports' tabs. Under 'Views', there are several expandable sections: 'Process Templates' (Currently Valid, All Versions), 'Process Instances' (Started By Me, Administered By Me, Critical Processes, Terminated Processes, Failed Compensations), 'Activity Instances' (Stopped Activities), 'Task Templates' (My Task Templates), and 'Task Instances' (My To-dos). The 'My To-dos' section is active, showing a table with one task: 'assignToManagerForAction' with a priority of 5 and a state of 'Ready'. Above the table are buttons for 'Work on', 'Release', 'Transfer', 'Start', and 'Change'. Below the table, it shows 'Items found: 1' and 'Items selected: 0'.

Figure 11-70 To-do human task created after running process

We select this task, click Work on, and see the window that is shown in Figure 11-71 on page 365.

Task Name: assignToManagerForAction

Task Input Message

Form View

lowStockEvents	sr_event	sr_ItemNumber	
		30	10
		60	20
		80	30

lowStockDetails	items	item_desc	
		Ball Pens Red 24pk	002.90
		Highlighters Assorted 5pk	003.89
		Laser Paper 28-lb 108 Bright 2500/cas	033.54

lowStockTotalCost: 1113.0

[View Source](#)

Task Output Message

Form View

humanActionTaken:

Figure 11-71 The created to-do human task

In Figure 11-71, we can see lowStockEvents, which contains the data that was originally sent to invoke the process, lowStockDetails showing (for each item) the information that was retrieved from the web service call to CICS, and lowStockTotalCost showing the total cost of the three low stock events. We then enter a message into the humanActionTaken area and click Complete.

In the SystemOut.log file of the server, we see the output that is shown in Example 11-8.

Example 11-8 Messages from the process

Task assigned to manager for action at: Tue Mar 23 14:08:35 EST 2010
 At: Tue Mar 23 14:17:22 EST 2010 Action taken: Getting a new supplier

These messages show that our process created the correct human task and that a human being acted.

11.2.26 Testing using the Integrated Test Client

As mentioned in “Testing with Integrated Test Client” on page 324, we explain how to use the Integrated Test Client to test our completed process.

We start the Integrated Test Client, as described in “Testing with Integrated Test Client” on page 324. WebSphere Integration Developer V7 opens a window similar to the window that is shown in Figure 11-72 on page 367.

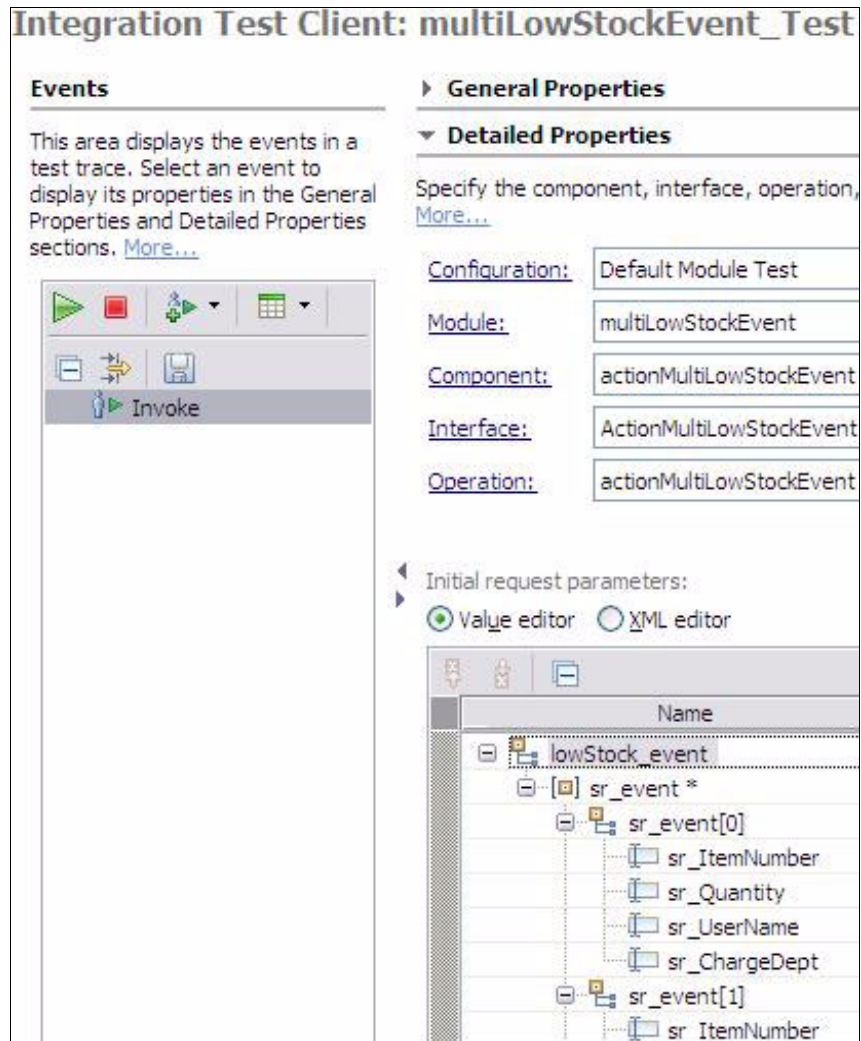


Figure 11-72 Initial view of Integrated Test Client

In the area under the “Initial request parameters” heading, we enter values for the fields that are passed to the process, as shown in Figure 11-73 on page 368.

Initial request parameters

Name	Type	
lowStock_event	listLowStockRequestEv...	[ab]
sr_event *	lowStockRequest[]	68
sr_event[0]	lowStockRequest	[ab]
sr_ItemNumber	string	[ab] 20
sr_Quantity	string	[ab] 30
sr_UserName	string	[ab] smith
sr_ChargeDept	string	[ab] 101
sr_event[1]	lowStockRequest	[ab]
sr_ItemNumber	string	[ab] 40
sr_Quantity	string	[ab] 10
sr_UserName	string	[ab] Jones
sr_ChargeDept	string	[ab] 201
sr_event[2]	lowStockRequest	[ab]
sr_ItemNumber	string	[ab] 70
sr_Quantity	string	[ab] 42
sr_UserName	string	[ab] Bloggs
sr_ChargeDept	string	[ab] 301

Figure 11-73 Setting values to pass to the process

A large green triangle shows in the upper-left corner of the window (as shown in Figure 11-72 on page 367). When we hover the mouse over this arrow, WebSphere Integration Developer V7 shows us a small box with the word “Continue”. We invoke the process by clicking this box. WebSphere Integration Developer V7 opens the window that is shown in Figure 11-74 on page 369.

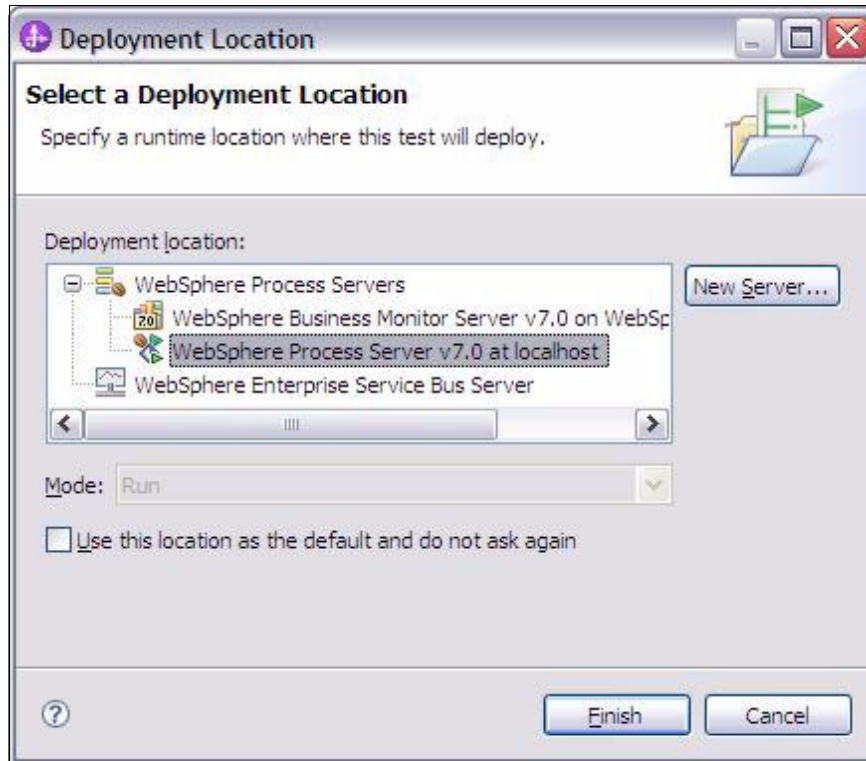


Figure 11-74 Selecting the server on which to run the test

In Figure 11-74, we select WebSphere Process Server v7.0 at localhost and click Finish. WebSphere Integration Developer V7 then prompts for a user ID and password to allow access to the Integrated Test Client, as shown in Figure 11-75 on page 370.

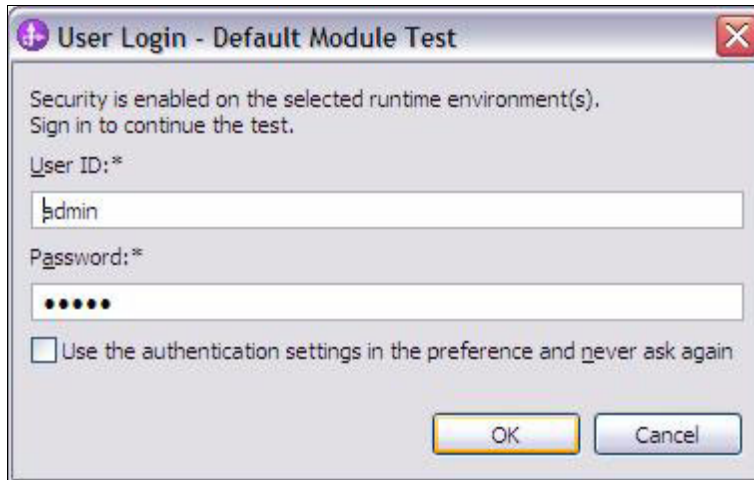


Figure 11-75 Supplying user ID and password details

WebSphere Integration Developer V7 then opens a progress window while it prepares the Integrated Test Client, as shown in Figure 11-76. Note that this task takes time.

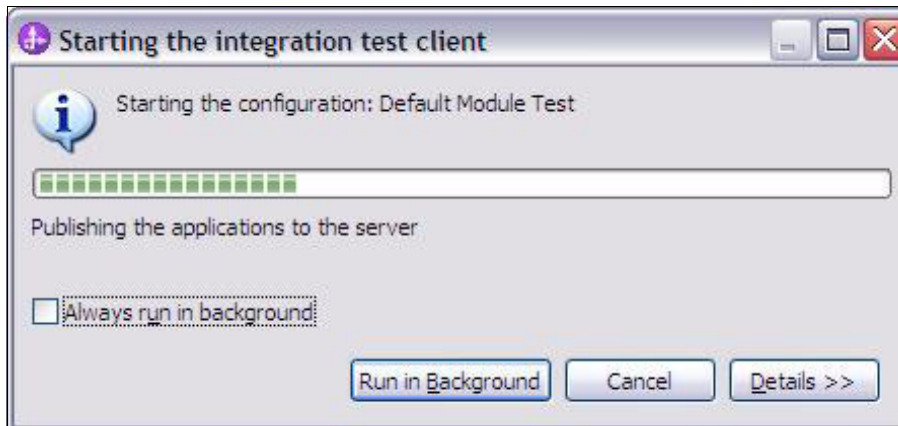


Figure 11-76 WebSphere Integration Developer V7 starting the Integrated Test Client

After the Integrated Test Client completes the test, WebSphere Integration Developer V7 shows us a trace of the activities that took place during the execution of the process (Figure 11-77 on page 371).

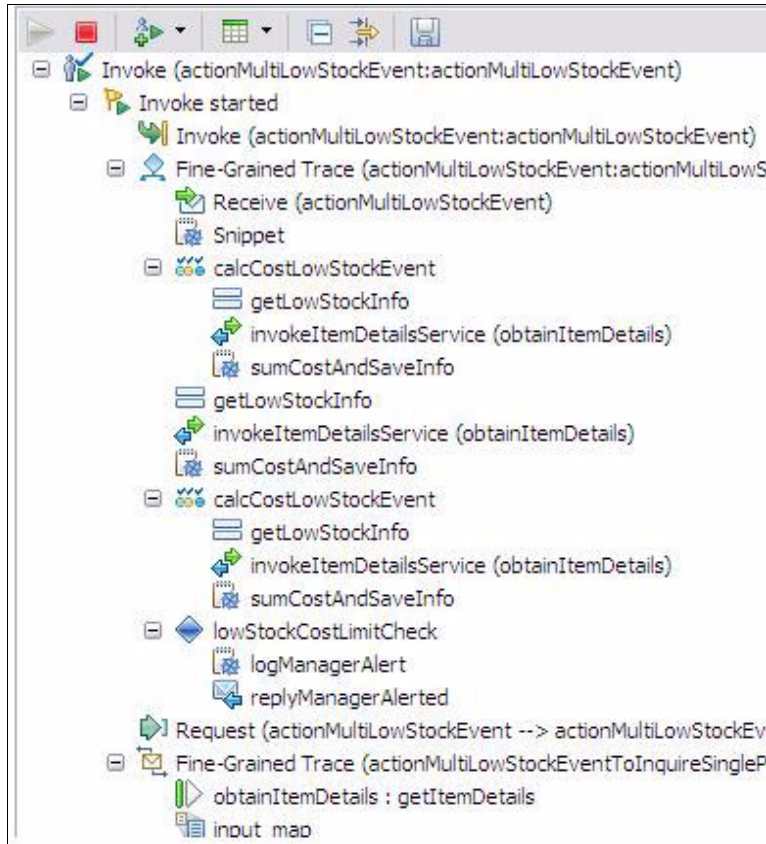


Figure 11-77 Trace of the execution of the process

You can select various entries in the trace, and if applicable, WebSphere Integration Developer displays the data that is associated with that part of the flow. For example, you can see what data was passed back from the Web Service call to CICS.

This test capability in WebSphere Integration Developer provides a useful way to debug your process flows.

11.2.27 Exporting the process wsdl

Having completed the testing of the process, we can export a wsdl file, which provides details about how to invoke this process as a Web Service. The wsdl file can be given to other individuals, who want to call this process, to incorporate into their applications.

To export the wsdl file, we select under Web Service Ports, right-click ActionMultiLowStockEventExport1_ActionMultiLowStockEventHttpPort, and select Export. In the window that opens, under Business Integration, we select WSDL and XSD and click Next.

In the Export window that opens next, we select multiLowStockEvent_ActionMultiLowStockEventExport1.wsdl. Under Export dependent resources, we *first* select “Merge dependent XSD resources into the parent WSDL file”, and then, we select “Merge dependent WSDL resources into the parent WSDL file”, as shown in Figure 11-78.

If you select the check boxes in the reverse sequence, WebSphere Integration Developer displays an error message and will not publish the wsdl file.

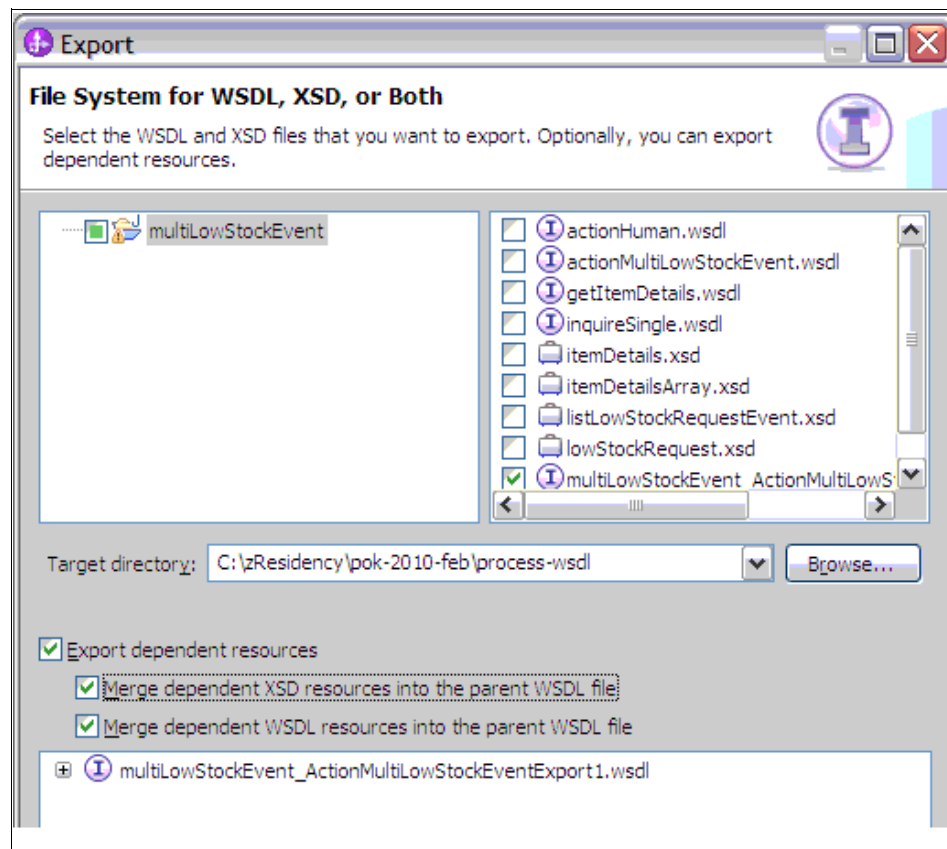


Figure 11-78 Start of process to export wsdl file

In Figure 11-78, we use Browse to select the destination directory on the Microsoft Windows personal computer and then click Finish to save the file. In

the directory, we see a single file called the `multiLowStockEvent_ActionMultiLowStockEventExport1.wsdl` file.

11.2.28 Exporting the ear file

To deploy this process into WebSphere Process Server, we select the `multiLowStockEvent` module, right-click, and select `Export`. Then, under `Java Enterprise Edition (EE)`, we select `EAR file` and click `Next`. We enter a destination folder in which to save the ear file and click `Finish`. Then, we deploy the ear file using the standard WebSphere application deployment approach.

When you deploy the ear file to a WebSphere Process Server, you need to verify that it shows as started in two places under `Applications`:

- ▶ Under `WebSphere enterprise applications` → `MultiInsufficientStockFailuresEventApp`
- ▶ Under `SCA Modules` as `multiLowStockEvent`

11.2.29 Application to test the process

We also developed an application to enable us to test the process that we had built. We developed an application, because we wanted to test running the test application and process in separate servers. We included this test application with the additional materials of this IBM Redbooks publication in Appendix A, “Additional material” on page 401.

To run this test application, deploy the ear file to a WebSphere server and enter a URL of the form:

<http://<hostIpAddress>:<port>/callLowStockEventProcessWeb/CallLowStockEventProcess>

There are input fields where you can specify the TCP/IP host and port values of the WebSphere Process Server where the process is deployed. You can use the default values and then click `Invoke process`.

11.2.30 WebSphere Business Events to process the mediation flow

Our original intent had been for WebSphere Business Events to use a Web Service to invoke our process directly. When we tried to test using WebSphere Business Events using a Web Service to invoke our process directly, we found that, at that time, WebSphere Business Events was unable to handle the array structure of the process Web Service. We did find though that WebSphere Business Events was able to write a JMS message with the array of data.

Rather than change the process to be invoked by a JMS message, we decided to create a mediation flow to run in the ESB. The mediation flow is invoked by the JMS message, the supplied data is extracted from the message, and the Web Service is invoked to initiate the process.

Using WebSphere Integration Developer, we created a mediation flow to achieve this conversion. Figure 11-79 shows the complete assembly diagram of the flow.

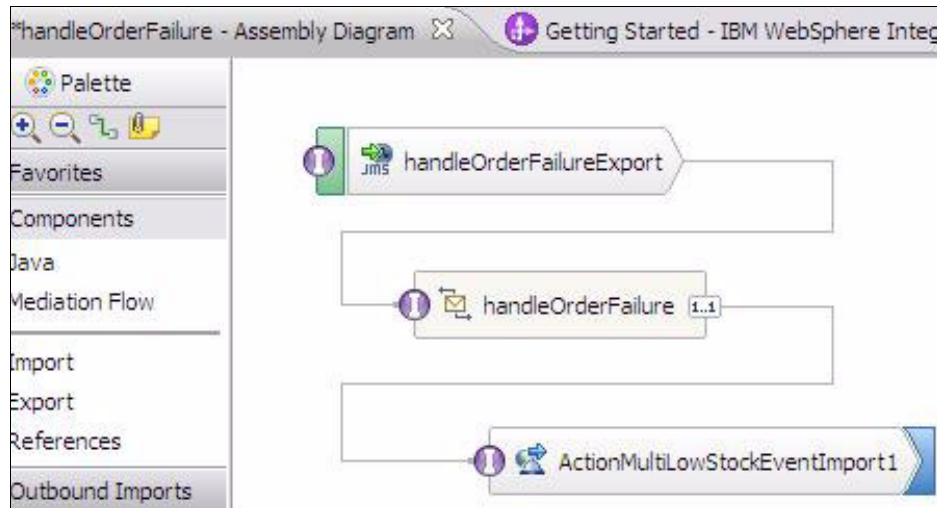


Figure 11-79 WebSphere Business Events to process mediation flow

Appendix A, “Additional material” on page 401 contains the ear file containing this mediation and the project interchange file.

11.2.31 Supplied files

We supply the following files that are associated with this chapter as part of the additional materials for this IBM Redbooks publication in Appendix A, “Additional material” on page 401:

- ▶ The `multiLowStockEventApp.ear` file contains the process application.
- ▶ The `scenario-4-wid-process.zip` project interchange file contains the project for the process.
- ▶ The `callLowStockEventProcess.ear` file contains the application that can be used to test the process through a web service call.
- ▶ The `appToTestScenario-4-process.zip` project interchange file contains the project for the test application.

- ▶ The `handleOrderFailureApp.ear` file contains the WebSphere Business Events to process the mediation flow.
- ▶ The `wbe-to-process-mediation-flow.zip` project interchange file contains the project for WebSphere Business Events to process the mediation flow.

11.2.32 Summary

In this chapter, we have described in detail how to design and create a process in WebSphere Integration Developer. Our process was logically fairly simple, but it still has taken quite a bit of explanation to describe how to build and test the process. If you are new to using WebSphere Integration Developer to create processes, we hope that this chapter has given you a better understanding of how to build a process with a certain degree of complexity. Becoming proficient in the use of WebSphere Integration Developer requires learning how to understand the mechanism around building a process. After you have a grasp of the fundamental way to use WebSphere Integration Developer to create a process, the principles that you have learned will enable you to build more complex processes and to extend your knowledge of WebSphere Integration Developer.



Part 4

Best practices

In this part of the book, we look at successful practices, including governance, security, and performance.



Best practices

In this chapter, we explain how to plan and organize your Customer Information Control System Transaction Server for z/OS (CICS TS) V4.1 event strategy. We describe governance, how CICS events can help maintain your information technology (IT) governance, and additional considerations for governing CICS events. We then discuss security for CICS events and conclude with guidance for troubleshooting, including a few points of confusion that we ran into with our environment.

12.1 Plan and organize

Perhaps the most important step in deploying events is to plan and organize all aspects of your business event strategy. Mapping out a clear understanding of who will create the event specifications, what the names will be, where they will be stored, and how they will be maintained are key areas to address.

Defining what products you will use in your implementation and how the products will communicate with each other is also an extremely important step. Depending on the scope of your implementation, for example, if you are designing an entire business events process involving multiple products, we recommend that you have a project leader that is responsible for overall management of the project. It is also vital to have representation and a leader for *each* of the products involved in your solution.

Frequent meetings during the planning stage to solidify your design will save time later. For example, during the design stage of this IBM Redbooks publication project, we had input from each of the product areas, which led to a graphical representation of our environment that detailed how each product will interact. After we agreed on our scenarios, we decided how to build our environment to accomplish our goals. We agreed on our naming conventions, transport mechanisms, communication formats, events emitted by CICS, processing of the events by the enterprise service bus (ESB), and how the events will be consumed.

We use three ESBs in our scenarios: WebSphere Enterprise Service Bus, WebSphere DataPower, and WebSphere Message Broker. Other products in our environment include WebSphere Business Events, WebSphere Business Monitor, and WebSphere Process Server. See Chapter 4, “Overview of the application and business scenarios” on page 57 for a complete overview of our environment.

12.1.1 Naming conventions

Document your naming conventions in the planning phase so that other people on the project know the agreed upon names for their products. In our case, we agreed on a naming convention for our WebSphere MQ queue names and field names for the payload data. The latter was not necessary from a processing design perspective, but it made it easier to discuss our scenarios during our planning and implementation meetings. We also decided on our Web service names, defined the input and output of each Web service, and shared the Web Services Description Language (WSDL) with the requesters of each service.

For example, our MQ queue naming uses this convention:

ESB-Product.Source.Format.Name

So our MQ queue WMB.CICS.CBE.ORDER indicates that WebSphere Message Broker is the ESB being used, CICS is the source of the information (CICS put the information onto the queue), the message is in CBE format, and the logical name of ORDER indicates this name is for an order event.

12.1.2 Transport

CICS offers several event processing (EP) adapters. In our scenario, we use the WebSphere MQ (WMQ) EP adapter to put the event message on an MQ queue.

See the CICS Transaction Server for z/OS V4.1 Information Center for details about the event processing adapters:

<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>

12.1.3 Communication formats

Because only certain combinations of format and transport are valid, or supported, with the EP adapters supplied by CICS, you select the EP adapter for the required transport and then select the format. In our case, we use MQ as the transport and have three choices for format: CICS flattened event (CFE), Common Base Event (CBE), or WebSphere business event (WBE). We emit the CICS event in CBE format and the ESB passes the message onto WebSphere Business Monitor in CBE format. The ESB transforms the CBE format to WBE format and passes the message onto the WebSphere Business Events product.

Base the format that you choose on the products that are the predominant consumers for the events. If the event is to be passed along through the ESB to a final event consumer that requires another event format, you can have the ESB transform the event into the new format. Alternatively, the event format emitted from CICS can be the format that the final event consumer expects, assuming the ESB can process the same format.

12.2 Governance

Service-oriented architecture (SOA) is a compelling technique for developing software applications that best align with business models. However, SOA increases the cooperation and coordination that are required between business and IT, as well as among IT departments and teams. The cooperation and

coordination are provided by SOA governance, which covers the tasks and processes for specifying and managing how services and SOA applications are supported. Events are a form of more loosely coupled (or decoupled) SOA.

Governance is the means of 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 by 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 determines who has the authority and responsibility for making the decisions. Management is the process of making and implementing the decisions. Governance says what must be done. Management makes sure that it gets done.

IT governance is about who is responsible for what in an IT department and how the department knows that the responsibilities are being performed. Specifically, IT governance establishes the following areas:

- ▶ Decision making rights that are associated with IT
- ▶ Mechanisms and policies that are used to measure and control the way that IT decisions are made and carried out

SOA adds the following unique aspects to governance:

- ▶ Acts as an extension of IT governance that focuses on the life cycle of services to ensure the business value of SOA
- ▶ Determines who will monitor, define, and authorize changes to existing services within an enterprise

SOA governance is the intersection of business and IT governance. It focuses on the life cycle of services to ensure the business value of SOA. SOA governance is the effective management of this life cycle, which is the key goal to SOA governance. Figure 12-1 illustrates this concept.

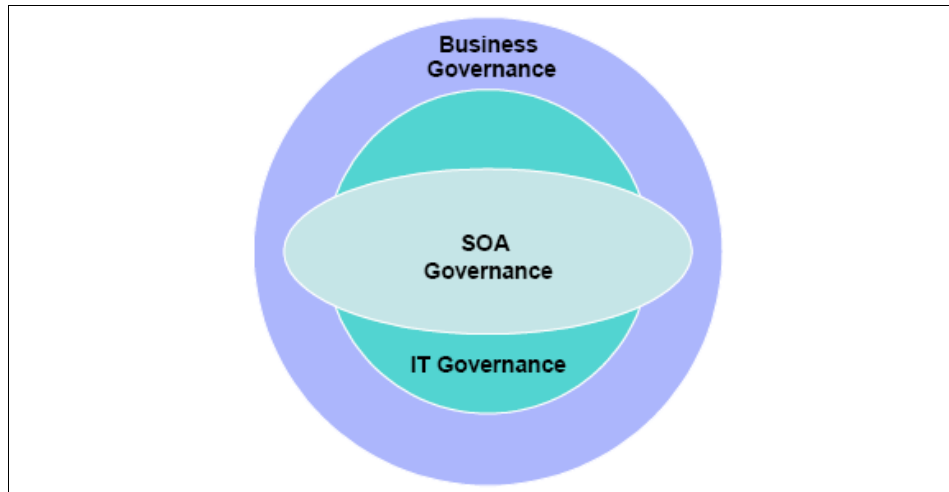


Figure 12-1 SOA governance in relation to business and IT governance

Governance becomes more important in SOA than in general IT. In an SOA, separate departments can develop, run, and manage consumer and provider services. Working together successfully requires complex coordination. For SOA to succeed, multiple applications need to share common services, which means that the departments need to coordinate making those services common and reusable. Governance issues are 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, they can also ideally improve overall IT governance. Employing SOA governance is key for companies to realize the benefits of SOA. For SOA to be successful, SOA business and technical governance is not optional, it is essential.

Automation is critical for successful governance. No matter what level of regulation the enterprise decides to have, automation will make it easier to deliver high-quality services on time and on budget. Those individuals who are subject to governance will welcome a governance process that gives near real-time feedback about what must change. A governance process saves time and effort and helps build in quality.

Getting to an acceptable governance maturity level does not happen by accident. An effective and evolving governance framework must be intentional and

focused. It requires leadership. It must define clear roles and responsibilities. It must enable well-thought-out and consistently implemented policies and procedures.

Products: In this IBM Redbooks publication, we do not describe specific products that can be used for governance. We leave the discussion and choice of software configuration management (SCM) tools to other publications.

12.2.1 CICS events and governance

Events allow businesses to be more responsive and flexible, and to address governance and compliance concerns.

The mainstream adoption of service-oriented architecture (SOA) has opened new opportunities for highly responsive business solutions. SOA brings greater flexibility to business processes, and it helps bring business and IT in line with each other. Enterprises are challenged by seeking to maximize SOA solution advantages (such as speed to market), at the same time complying with business controls, industry standards, and government legislation.

Business governance and compliance are increasingly important in many industries. These terms cover a crucial range of issues, including financial transparency, information privacy, and process control. The Sarbanes-Oxley Act, the Health Insurance Portability and Accountability Act (HIPAA), or Basel II and their associated information requirements are a few of the standards that are required for business compliance and governance.

Governance describes a formalization of the decision-making processes within an organization. It can cover many aspects of business and depends on accurately maintaining and auditing which decisions can be freely made and which decisions need specific approvals, and it determines who can make them.

Compliance is about ensuring adherence to mandated standards and governance policy. This adherence includes the definition of information about which governance decisions are based and maintaining accurate operational control to ensure that business application execution meets the required enterprise, industry, and government standards.

Using CICS events can help maintain the IT governance of your business applications by capturing events without the need to change the business application. Events can be created to monitor business processing, to capture data of interest, or to perform seasonal processing, all without changing the application.

CICS events add new aspects to IT governance that you need to consider:

- ▶ Determine who will define, monitor, and authorize changes to existing business events within an enterprise
- ▶ Ensure that any application program changes are communicated to the business events area of the enterprise

Due to the decoupled nature of event processing, you need to consider the interaction between application programs and event specifications when making changes. This area is similar in nature to an application program accessing a file. If the file record layout is changed, you must review the application program and possibly alter it to adjust to the new file record layout. Likewise, any application program change must be reviewed to understand how a change might affect the event specifications for the business process. If you make your event capture specification as specific as possible, this specification can help determine if program changes will affect your event. For example, if you know the event is only emitted from a particular program, you can include a filter on the current program name to ensure that you capture the required event. Creating a filter for a specific program also raises the awareness that any change to the application program must be reviewed to see if it affects the event capture specification; you will need to register this interaction with your IT governance product or process.

If you decide not to use the noninvasive capture points and instead to use the EXEC CICS API call SIGNAL EVENT within a program, the governance of the event is somewhat simplified. That is, any changes to the program and event capture points can be managed within the program. Typically, you will want to use noninvasive capture points, but SIGNAL EVENT might simplify portions of IT governance in certain environments. However, you still have governance of the capture specification, because the capture specification is separate from the program code.

12.2.2 Artifacts

Include the artifacts for your business events as part of your governance policy. We do not discuss specific products that can be used for governance in this IBM Redbooks publication. We leave the discussion and choice of software configuration management (SCM) tools to other publications.

CICS event artifacts

The artifacts for CICS events can consist of the following components:

- ▶ Schema definitions (.xsd) if using CBE or WBE formats
- ▶ COBOL copybooks
- ▶ CICS event bundle

Schema definitions or COBOL copybooks

You can export the event specification from the Adapter tab of the CICS Explorer. If you use the WebSphere MQ (WMQ) EP adapter, you can specify a data format of CICS Flattened Event (CFE), common base event (CBE), or WebSphere Business Events (WBE). The event specification will be a COBOL copybook for the CFE format, and an XSD for either the CBE or WBE format. In addition to the COBOL copybooks exported from the Event Binding Editor, you might have copybooks that are imported into the editor that define interfaces (such as COMMAREA layouts) that are used in capture specifications.

CICS event bundle

You create an event binding in a CICS bundle project using the CICS Event Binding editor within the CICS Explorer. The bundle can be exported directly to the z/OS UNIX file system (zFS) or to your local file system. In either case, you can use your software configuration management (SCM) tool of choice to manage the governance of the CICS bundle.

There are two ways to replace a deployed bundle resource in CICS after the bundle resource is installed, if necessary:

- ▶ Disable, discard, and then, install the changed version of a bundle with the same name. No events will be emitted from the moment that the bundle is disabled until the moment that the installation completes successfully.
- ▶ You can replace a bundle without disabling the event binding by creating a new bundle. The new bundle name must differ from the original bundle's name, and the new bundle must contain the event binding with the same name. Events will continue to be emitted until the bundle installation completes successfully, at which point, the new binding will replace the previous version.

By using the second method, you can make the change and continue to capture events automatically. This mechanism also allows you to back out the change by putting the old definition back into effect; simply disable the new bundle resource and enable the old bundle resource again.

Other artifacts used in this Redbooks publication

In our Redbooks project, we also have schema (.xsd) from WebSphere Business Events that the ESBs use to transform the CBE format message to WBE format. We also have WSDL to describe our Web services. These artifacts need to be part of your governance policy so that they are stored and managed by the policy for your enterprise.

IBM WebSphere Service Registry and Repository

The CICS TS V4.1 Web Services assistant includes interoperability support for the IBM WebSphere Service Registry and Repository. IBM WebSphere Service Registry and Repository can help you manage and govern your services and processes. You can use IBM WebSphere Service Registry and Repository to find the Web services that you are requesting more quickly and to enforce version control of the Web services that you provide.

IBM WebSphere Service Registry and Repository is also a good place to store the schemas for CBE or WBE events, because they define a kind of interface between the event producer and the event consumer.

Both DFHLS2WS and DFHWS2LS include parameters to interoperate with IBM WebSphere Service Registry and Repository. DFHLS2WS also includes an optional parameter so that you can add your own customized metadata to the WSDL document in IBM WebSphere Service Registry and Repository.

If you want the Web services assistant to communicate securely with IBM WebSphere Service Registry and Repository, you can use Secure Sockets Layer (SSL) encryption. Both DFHLS2WS and DFHWS2LS include parameters for using SSL encryption. See the CICS Transaction Server for z/OS V4.1 Information Center for additional details:

<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>

12.2.3 Audit and change control

CICS monitoring and statistics can be useful for gathering and analyzing information about CICS events. There are also other options (depending on the EP adapter) that allow you to specify the transaction ID that runs when the event is triggered or to specify a user ID under which the EP adapter transactions will run (12.3.4, “Runtime security” on page 395). We describe several of the major points for monitoring and statistics, but for more information, refer to these websites:

- ▶ The CICS Transaction Server for z/OS V4.1 Information Center:
<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>
- ▶ Problem determination in the Troubleshooting chapter of *Implementing Event Processing with CICS*, SG24-7792:
<http://www.redbooks.ibm.com/abstracts/sg247792.html?Open&pdfbookmark>

Monitoring

CICS monitoring collects data about the performance of all user and CICS transactions during online processing, for later offline analysis. The records that are produced by CICS monitoring are MVS™ System Management Facility (SMF) type 110, and they are written to an SMF data set. Various methods can start the monitoring:

- ▶ CICS system initialization parameter MN
- ▶ CICS master terminal transaction CEMT SET MONITOR ON
- ▶ Transaction CEMN
- ▶ CICS System Programmer Interface (SPI) command EXEC CICS SET MONITOR

CICS TS 4.1 adds the following new performance class data fields, containing event processing information to the DFHCICS group:

- ▶ EICTOTCT
The total number of EXEC CICS commands that are issued by the user task
- ▶ ECSIGECT
The number of EXEC CICS SIGNAL EVENT commands that are issued by the user task
- ▶ ECEFOPCT
The number of event filter operations that are performed by the user task
- ▶ ECEVNTCT
The number of events that are captured by the user task

You can use the CICS Explorer, with the CICS Performance Analyzer for z/OS plug-in, to view the event processing monitor data. The CICS Performance Analyzer for z/OS plug-in interfaces with the CICS Performance Analyzer tool, so you need to have CICS Performance Analyzer for z/OS installed on your z/OS system.

Statistics

CICS gathers statistical data about the system resource usage and the performance of the CICS system during online processing. The CICS statistics domain creates MVS System Management Facility (SMF) type 110, sub-type 2 records, which are written to an SMF data set. Statistical data is useful both for performance tuning and for capacity planning.

You can start recording statistics by using the following methods:

- ▶ The CICS system initialization parameter STARTRCD=YES, which causes interval statistics to be recorded every three hours, by default
- ▶ CICS Master terminal transaction CEMT SET STATISTICS(ON)

- ▶ SPI command EXEC CICS SET STATISTICS RECORDING(ON)

You can use the sample statistics utility program (DFH0STAT) to generate reports about the event processing statistics. You can request global statistics for EVENTPROCESSING, and you can request global and resource statistics for EVENTBINDINGS and CAPTURESPECS.

Install the resource group DFH\$STAT in your CICS region and run the STAT transaction. When you are in the STAT application, press PF4 Reports and scroll to the page containing the event processing, EVENTBINDINGS, and CAPTURESPECS reports. You can also select the BUNDLES report, which provides details of all the currently installed bundles.

12.3 Security considerations for CICS events

Event binding files define when CICS is to emit an event. You create these event binding files using a workstation-based Event Binding Editor. The event binding file and associated artifacts are copied to a directory on the zFS and installed in CICS using a BUNDLE resource.

The person creating the event binding uses the Event Binding Editor, which is supplied with both the CICS Explorer and Rational Developer for System z (RDz). Although a Business Analyst describes the need for the event and might be involved in the initial creation and description of the event binding, the event binding is often created by a CICS application programmer who is familiar with the application or an application analyst or developer who specializes in creating CICS event bindings. To perform its function, an Event Developer performs these tasks:

1. Be informed of what events CICS needs to emit
2. Find the appropriate place in the application program where events must be emitted
3. Download appropriate copybooks that describe the data being used for filters and enrichment
4. Create an event binding in an event binding project
5. Export the event binding to zFS
6. Create a BUNDLE resource in CICS
7. Install the BUNDLE resource in CICS

Security can be applied to each of these steps.

Depending on the sensitivity of the data in the event, you might want to secure access to the location where events are placed by CICS or other products that emit events (for example, ensure that CICS is allowed to place messages on a given WebSphere MQ queue). If events are directed to an Event Bus (WebSphere Message Broker, WebSphere Enterprise Services Bus, or WebSphere DataPower), sensitive messages can be encrypted as they flow between buses or queue managers. In addition to securing the individuals who can place events on the Event Bus and how events are transferred between entities within or between buses, you might also need to secure the individuals who are allowed to access or remove events from the Event Bus.

From a security perspective, note that CICS's event capture mechanism does not change any application information. CICS monitors whether the conditions are correct for an event to be emitted and then emits the event.

In addition to placing an event on an Event Bus, CICS can start a CICS transaction and provide the event data to that transaction, or CICS can invoke a user-written event processing adapter to process the event data. Your production turnover procedures need to ensure that a rogue programmer does not trigger a transaction, or event processing adapter that the rogue programmer writes, that might alter corporate data in a malicious way.

You can secure all the items that were discussed in the previous paragraphs, but for the purposes of this chapter, we only discuss the following items:

- ▶ Development security:
 - Determining the individuals that can create event definitions
 - Creating event bindings can be done by anyone who is given access to the CICS Explorer
 - Exporting the bundle to zFS is under Time Sharing Option (TSO) user ID and password control
- ▶ Resource security
- ▶ Deployment security:
 - Securing the deployment of events to zFS
 - Securing installation of events into CICS (normal CICS security)
 - Securing who can see the events that are installed
- ▶ Runtime security:
 - Securing emission of events to consumers

If your event data is sensitive, you also need to control access and authority for events being added and removed from your enterprise service bus (ESB), the Event Bus, as well as privacy and non-repudiation of event contents. A

discussion of the protection of emitted events containing sensitive data is beyond the scope of this book. Consult the documentation about your ESB or event transport for discussions about how to protect and secure data.

12.3.1 Development security

CICS allows you to specify that events must be emitted in a non-invasive or invasive manner. Non-invasive means that no application changes are required. From a logical perspective, when an application runs, CICS detects whether certain conditions are met and emits an event. The information that CICS needs to monitor the application and determine whether an event must be emitted is stored in an event binding file.

You can create event binding files using the Event Binding editor that is supplied with both the CICS Explorer and IBM Rational Developer for System z. You can download the CICS Explorer from the CICS Explorer website and install it by expanding a compressed file.

When creating an event binding file, you will likely want to filter information (for example, to test if the “in stock” amount is less than a specified reorder point). The location where the in stock amount resides within a record or data structure is part of a data layout. Although you can manually provide the displacement, size, and type of the field, you can also have the Event Binding editor reference a copybook (data layout) so that the Event Binding editor can simply insert the appropriate displacement, size, and field type of the data element.

When using the copybook approach, you will likely need to access a copybook that currently resides on z/OS. The copybooks might be accessible on a partitioned data set (PDS), or they might possibly reside in a source control system, such as Software Configuration and Library Management (SCLM) productivity tools. You will need a local copy of the data layout, but you can use normal z/OS or source control system security to protect the access to the data layouts (copybooks).

After you have completed the development of the event binding, you will have a project containing artifacts that need to be transferred to a z/OS zFS file system. The event binding project is a directory with one or more event binding files along with a META-INF directory containing a manifest file (named `cics.xml`). You need to transfer the entire event binding directory (the project) and its contents to the z/OS zFS file system. There are several ways to transfer the project to your zFS directory, including these methods:

- ▶ The “Export to System z HFS” function in the CICS Explorer, which uses File Transfer Protocol (FTP)

- ▶ IBM Rational Developer for System z, as part of the event binding development process
- ▶ FTP
- ▶ IND\$FILE

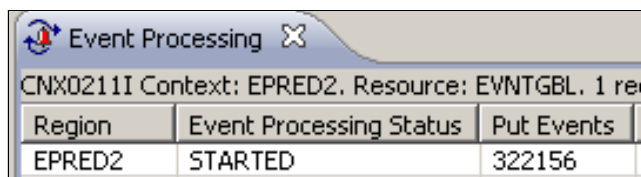
Alternative: You can export the bundle from the Event Binding editor and store it in a software configuration management (SCM) system, and someone other than a developer can deploy it to zFS.

Securing the transfer of the event-related artifacts to the z/OS zFS file system is performed using the normal security procedures that are associated with the type of file transfer being used. All these transfer techniques use normal user ID-based access security on z/OS.

If desired, you can configure IBM Rational Developer for System z with an SSL connection to the IBM Rational Developer for System z z/OS components to encrypt data as it is transferred to and from z/OS. You can configure the CICS Explorer to use SSL to CICS for resource display, definition, and administration.

12.3.2 Resource security

To see if event processing is enabled in CICS, you can inquire on the EVENTPROCESS resource. If event processing is enabled, the EVENTPROCESS resource shows a status of *Started*. Using CEMT, you can only see the status; however, if using the CICS Explorer, additional information is available (Figure 12-2).



Region	Event Processing Status	Put Events
EPRED2	STARTED	322156

Figure 12-2 Event processing status

CICS is informed about event binding files and the CICS manifest file by using a BUNDLE resource. The BUNDLE resource references the directory containing the CICS manifest and the event binding files. After the BUNDLE containing the event binding file is installed, you can display BUNDLES and see your BUNDLE in an Enabled status (Figure 12-3 on page 393).

Region	Name	Status	Install Time
EPRED2	LOADTEST	✓ ENABLED	Mar 11, 2010 4:55:59 PM

Figure 12-3 Bundle display

A detailed display of your BUNDLE shows the number of BUNDLE parts. When using BUNDLES for events, the part count reflects the number of event bindings in the BUNDLE (assuming the bundle is being used exclusively for events and nothing else). When using the CICS Explorer, you can see the individual BUNDLE parts (Figure 12-4). CEMT does not provide a way to display BUNDLE parts; however, you can use the CICS System Programmer Interface (CICS SPI) command to browse through the BUNDLE parts.

Region	Bundle	Bundle Part	Enable Status	Meta Data File	Part Class	Part Type
EPRED2	LOADTEST	InsufficientStock	✓ ENABLED	InsufficientStock.evbind	DEFINITION	http://www
EPRED2	LOADTEST	SuccessfulOrder	✓ ENABLED	SuccessfulOrder.evbind	DEFINITION	http://www

Figure 12-4 Bundle parts

For each event specification in the BUNDLE, you see an EVENTBINDING resource. A display of the EVENTBINDING shows its associated BUNDLE (Figure 12-5).

Region	Name	Status	Bundle	Install Time
EPRED2	InsufficientStock	✓ ENABLED	LOADTEST	Mar 11, 2010 4:55:59 PM
EPRED2	SuccessfulOrder	✓ ENABLED	LOADTEST	Mar 11, 2010 4:55:59 PM

Figure 12-5 Event binding

For each capture specification in an event specification, there is a CAPTURESPEC resource. When using the CICS Explorer, you can display capture specifications (Figure 12-6 on page 394). CEMT does not provide a way to display capture specifications; however, you can use SPI to browse through the capture specifications in an event binding and to display the details of a capture specification (similar to the details that you can display with the CICS Explorer).

Region	Capture Specification	Event Binding	Capture Type	Capture Point	Event Name	Events Captured
EPRED2	InsufficientStock	InsufficientStock	POSTCOMMAND	LINK_PROGRAM	FailedOrder	4
EPRED2	OrderSuccess	SuccessfulOrder	POSTCOMMAND	LINK_PROGRAM	ItemOrder	322152

Figure 12-6 Capture specifications

CICS event processing-related transactions

The CEPD transaction is the event processing dispatcher task. The CEPD transaction is a category 1 transaction that is implemented by the DFHEPDS program. The CEPM transaction handles the captured event queue. It distributes events to be formatted and emitted. The CEPM transaction is a category 1 transaction that is implemented by program DFHEPSY. Both CEPD and CEPM are defined internally by the event processing domain. The CRLR transaction is a category 1 transaction and is used for bundle resource resolution. See the CICS Transaction Server V4.1 Information Center for details about how to protect category 1 transactions:

<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>

Resource security

Resource security for EVENTBINDING resources uses access resource profiles in the RCICSRES class or the WCICSRES grouping class, or equivalent customer-defined classes specified in the XRES system initialization parameter. You must supply a prefix of EVENTBINDING to the name of the EVENTBINDING resource definition. When you start to browse for CAPTURESPEC, CICS checks whether you have authority to access the EVENTBINDING that contains the CAPTURESPEC.

Command security

Command security for CAPTURESPEC resources uses the CAPTURESPEC resource in the CCICSCMD class or the VCICSCMD grouping class. Command security for EVENTBINDING resources uses the EVENTBINDING resource in the CCICSCMD class or the VCICSCMD grouping class. Command security for EVENTPROCESS resources uses the EVENTPROCESS resource in the CCICSCMD class or the VCICSCMD grouping class.

Security using the XRES resource security parameter

You use the XRES system initialization parameter to perform a security check on the following CICS resources: ATOMSERVICE, BUNDLE, DOCTEMPLATE, EVENTBINDING, JVMSERVER, and XMLTRANSFORM. The BUNDLE and EVENTBINDING resources relate to event processing.

12.3.3 Deployment security

The BUNDLE resource that references the directory containing the event binding files and CICS manifest file is the unit of deployment for CICS events. The application programmer can be denied access to creating BUNDLE resources or denied access to create BUNDLE resources outside the development environment. You can define the BUNDLE resource to CICS using the CICS Explorer, CICSplex® SM (CPSM), the CEDA transaction, and Configuration Manager for z/OS.

When a BUNDLE resource is installed that includes an event binding for which a user ID was specified in the Adapter tab of the CICS Event Binding Editor, CICS checks that the user ID performing the installation operation is authorized as a surrogate user of the user ID specified in the CICS Event Binding Editor. This check also applies to the CICS region user ID during group list installation on a CICS cold or initial start.

CICS Transaction Server for z/OS V4.1 introduces surrogate user security for event binding, *userid.DFHINSTL*.

For more information about user IDs, see the next few paragraphs about runtime security.

12.3.4 Runtime security

Next, we describe runtime security.

User ID associated with event processing adapter

You need to ensure that the user ID that is used to run the EP adapter has sufficient authority to place the event data on the desired destination (such as the Event Bus). By default, the EP adapter runs under the CICS region user ID. If you specify a user ID in the Event Binding Editor, this user ID is used for EP adapter processing. If you select “Use Context User ID” in the Event Binding Editor, the EP adapter will run with the user ID under which the event was captured. You select the “Use Context User ID” option in either the Adapter section or the Advanced Options section of the Adapter tab in the Event Binding Editor, depending on the type of adapter that is chosen.

Note that there is overhead associated with specifying a user ID or transaction ID other than the default, because then the EP adapter runs under a separately attached task.

Transaction ID associated with EP adapter processing

There are three types of EP adapters supplied with CICS: the WebSphere MQ adapter, the Temporary Storage (TS) Queue adapter, and the transaction start EP adapter. You can also write a custom EP adapter. Although similar, establishing transaction IDs for the adapters varies slightly.

HTTP EP adapter: The HTTP EP adapter has been added to the EP adapters supplied with CICS, but it was not available at the time that we wrote this book.

WebSphere MQ Queue EP adapter or the TS EP adapter

If you specify a user ID, but you do not specify a transaction ID, the EP adapter runs under the default transaction for the EP adapter type:

- ▶ The WebSphere MQ Queue EP adapter runs under the CEPQ transaction.
- ▶ The transaction start (TS) EP adapter runs under the CEPT transaction.

If you specify a user ID and you want the EP adapter processing to run under a transaction other than CEPQ or CEPT (depending on whether you are using the WebSphere MQ Queue EP adapter or the TS Queue EP adapter), you need to make a copy of the CICS-supplied adapter transaction ID and name the transaction under which you intend the EP adapter to run. Additionally, you need to specify the name of the transaction ID as the Transaction ID in the advanced Options section of the Adapter tab in the Event Processing editor.

When using the WebSphere MQ Queue EP or the TS Queue EP adapter, if you specify neither a user ID nor a transaction ID in the Advanced Options section of the Adapter tab in the Event Binding Editor, the adapter is linked to under the dispatcher transaction CEPD.

The Custom EP adapter

When using the Custom EP adapter, you can optionally specify a transaction ID in the Adapter section of the Adapter tab of the Event Binding Editor. You can optionally specify the user ID in the Advanced Options section of the Adapter tab of the Event Binding Editor. If neither a user ID or a transaction ID is specified, the Custom adapter is linked to under the dispatcher transaction CEPD.

The transaction start EP adapter

When using the transaction start (TS) EP adapter, the transaction ID, which is specified in the Adapter section of the Adapter tab in the Event Binding Editor, is required. The transaction start EP adapter always runs under the dispatcher task with the default transaction ID and user ID, because its only action is to start the transaction that will consume the event.

12.4 Troubleshooting

In this section, we describe several of the problems that you might encounter when using CICS events and how to diagnose these problems. For detailed troubleshooting and problem determination information, see these resources:

- ▶ Event processing problem determination in the CICS Transaction Server for z/OS V4.1 Information Center:
<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>
- ▶ Problem determination in the Troubleshooting chapter of *Implementing Event Processing with CICS*, SG24-7792:
<http://www.redbooks.ibm.com/abstracts/sg247792.html?Open&pdfbookmark>

12.4.1 Problems we encountered

There are a few problems, or potential points of confusion, that we ran into when configuring our events in CICS. You can address them easily when you know what the messages are trying to describe. Development might improve the messages in the future, but we document them for reference.

Copybook filename too long

When selecting fields from a copybook, if the filename on your personal computer is longer than eight characters, the operation fails with a DFHPI9506E message stating that the PDSMEM is greater than 8 (Figure 12-7). The term *PDSMEM* refers to the filename, which might not be obvious at first. The Event Binding editor uses an embedded DFHLS2WS, and the term PDSMEM makes sense on a z/OS platform. The solution is to shorten the filename that contains your copybook to eight or fewer characters.

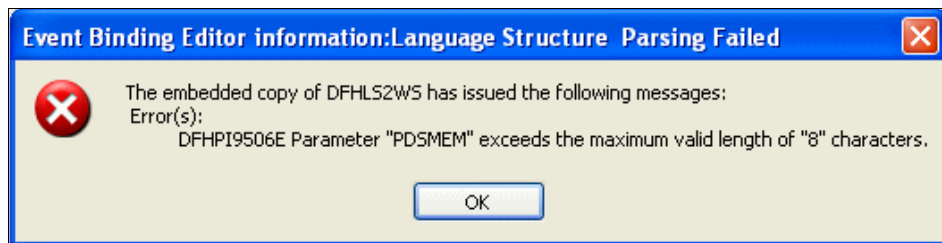


Figure 12-7 DFHPI9506E

Alignment of levels in COBOL copybook

If you cut and paste a copybook (or a portion of a copybook) into a file, the levels must start in the correct COBOL columns. If the alignment is incorrect, the import might fail with the DFHPI9552E message (Figure 12-8), because the embedded version of DFHLS2WS in the Event Binding editor finds text where it needs to find an integer.

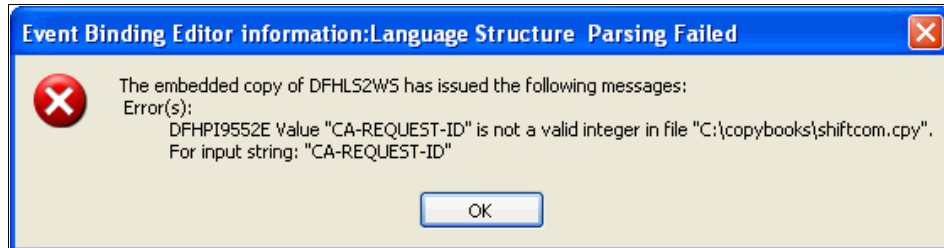


Figure 12-8 DFHPI9552E

Browse appears inactive on export of bundle to HFS

If you have an incorrect FTP port number specified and click Browse to navigate to your hierarchical file system (HFS) directory to export the bundle, selecting Browse does not appear to do anything and no errors are returned. However, if you specify the HFS path to which you want to export and click Finish, an error message appears (Figure 12-9 on page 399) as expected due to the incorrect FTP port number. Specify the correct port number for FTP; typically, the FTP port number is 21.

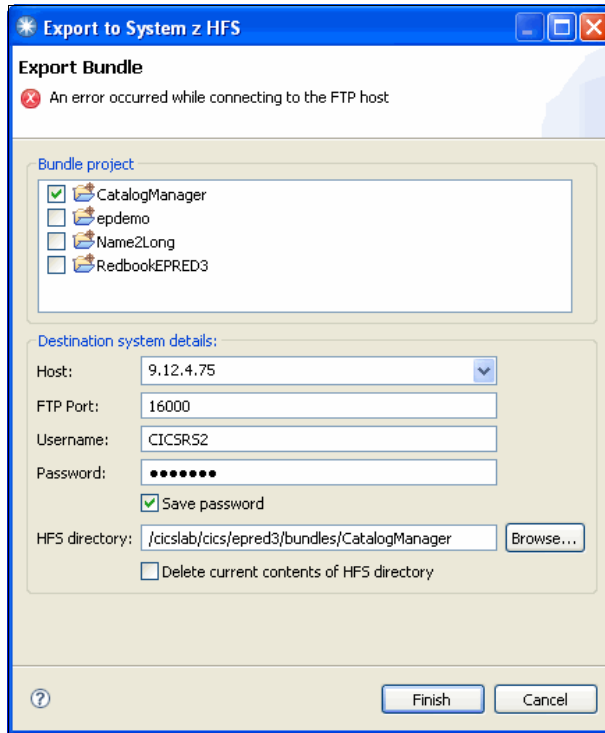


Figure 12-9 Incorrect FTP port number specified



Additional material

This book refers to additional material that you can download from the Internet as described in the following sections.

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 this site:

<ftp://www.redbooks.ibm.com/redbooks/CICCATLOGORDERS.zip>

Alternatively, you can go to the IBM Redbooks web site:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, which is `CICSCATALOGORDERS.zip`.

Using the web material

The additional web material that accompanies this book includes the following file:

<i>File name</i>	<i>Description</i>
CICSCATALOGORDERS.zip	CICS catalog orders

We supply the following files that are associated with Chapter 11, “WebSphere Process Server” on page 295 as part of the additional materials for this IBM Redbooks publication:

- ▶ The `multiLowStockEventApp.ear` file contains the process application.
- ▶ The `scenario-4-wid-process.zip` project interchange file contains the project for the process.
- ▶ The `callLowStockEventProcess.ear` file contains the application that can be used to test the process through a web service call.
- ▶ The `appToTestScenario-4-process.zip` project interchange file contains the project for the test application.
- ▶ The `handleOrderFailureApp.ear` file contains the WebSphere Business Events to process the mediation flow.
- ▶ The `wbe-to-process-mediation-flow.zip` project interchange file contains the project for WebSphere Business Events to process the mediation flow.

System requirements for downloading the web material

Follow these instructions.

How to use the web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the web material zip file into this folder.

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 publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 404. Note that the documents referenced here might be available in softcopy only.

- ▶ *Implementing Event Processing with CICS*, SG24-7792
- ▶ *CICS Explorer*, SG24-7778

Other publications

This publication is also relevant as a further information source:

- ▶ *IBM Tivoli OMEGAMON XE for CICS Transaction Gateway on z/OS: User's Guide*, SC23-5963

Online resources

These web sites are also relevant as further information sources:

- ▶ CICS Transaction Server for z/OS, Version 4 Release 1:
<http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp>
- ▶ WebSphere Business Events:
<http://publib.boulder.ibm.com/infocenter/wbevents/v6r2m1/index.jsp>
- ▶ WebSphere Business Monitor:
<http://www-01.ibm.com/software/integration/wbimonitor/>
- ▶ WebSphere Enterprise Service Bus:
<http://www-01.ibm.com/software/integration/wsesb/>

- ▶ WebSphere Message Broker:
<http://www-01.ibm.com/software/integration/wbimessagebroker/>
- ▶ CICS Explorer:
<http://www-01.ibm.com/software/http/cics/explorer/>
- ▶ WebSphere Business Events Design tool:
<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/Designmigration.html>
- ▶ WebSphere Business Events runtime and WebSphere Process Server connection:
<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.integrating.doc/doc/sibx.html>
- ▶ WebSphere Business Events Business Events Tester widget:
<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/testingeventlogic.html>
- ▶ WebSphere Business Events Event Capture and Replay widget:
<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.appdev.doc/doc/eventcapturereplay.html>
- ▶ WebSphere Business Events enhanced application logging and WebSphere Application Server trace:
<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/topic/com.ibm.wbe.admin.doc/doc/wbelogs.html>

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers, Web docs, draft publications and Additional materials, as well as order hardcopy IBM Redbooks publications, at this web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Leveraging CICS Events with an ESB

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Leveraging CICS Events with an ESB



Exploit CICS events using your enterprise service bus

Learn about CICS events and governance in depth

Transform and enrich CICS events with an ESB

There are many reasons to use an enterprise service bus (ESB) as the Event Bus:

- ▶ Transformation: Function that transforms the incoming event by translating or splitting it
- ▶ Enrichment: Function that enriches the content of events with reference data from multiple possible sources
- ▶ Validation: Function to provide validation against required criteria
- ▶ Pattern detection: Function that recognizes actual and retrospective patterns; a combination from possibly multiple events, characterizing a significant business situation
- ▶ Filtering: Stateless function that filters events based on their content, that is, the information that is carried by the message generated when the event happened
- ▶ Aggregation: Function that can group events as necessary
- ▶ Routing: Function that routes events to the destination based on various possible routing patterns, such as pre-established itinerary, calendar-based, or subscription or “intelligent” routing decisions

In this IBM Redbooks publication, we show examples of using an ESB to transform and enrich an event received from Customer Information Control System (CICS) Transaction Server. We also show an example of enriching an event.

This book is intended for anyone planning to use CICS events with an ESB.

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