WebSphere. software

IBM

Draft Document for Review May 4, 2006 3:20 pm

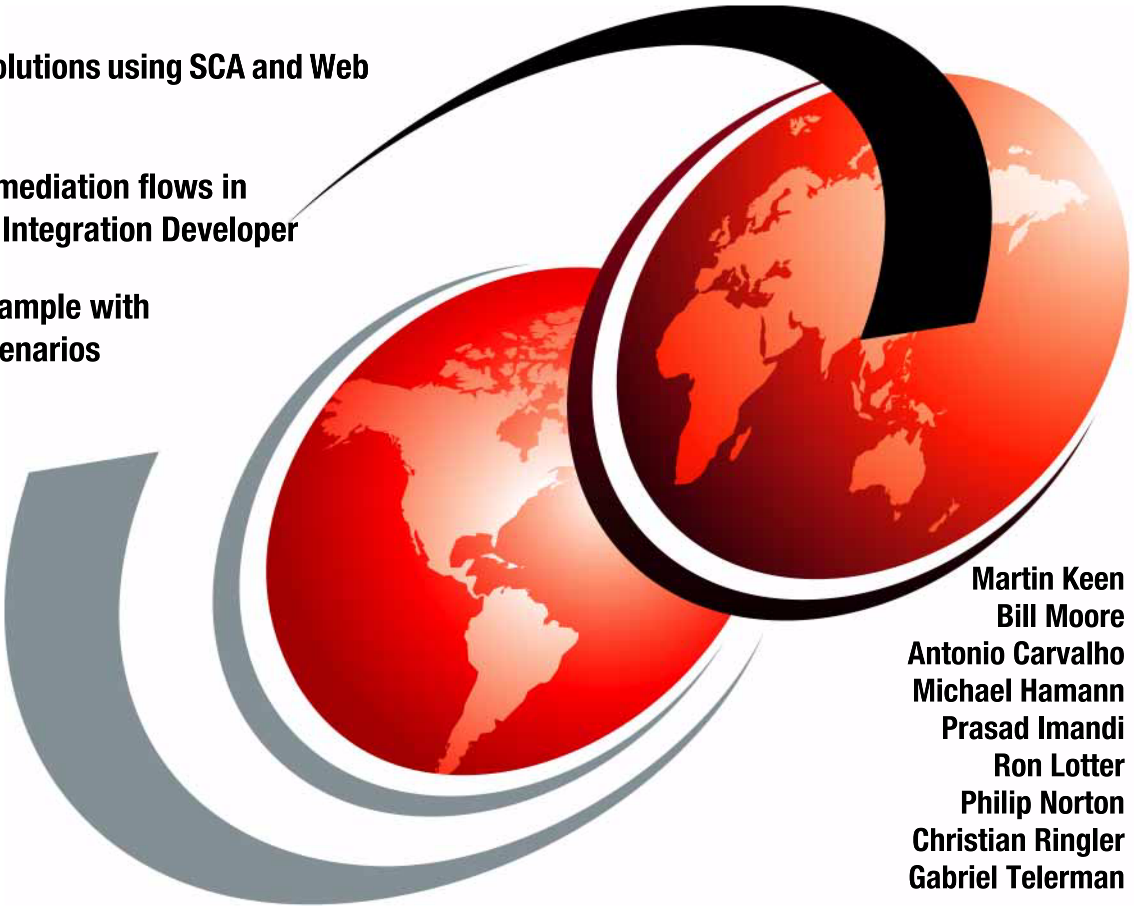
SG24-7212-00

Getting Started with WebSphere Enterprise Service Bus V6

Build ESB solutions using SCA and Web services

Implement mediation flows in WebSphere Integration Developer

Learn by example with practical scenarios



Martin Keen
Bill Moore
Antonio Carvalho
Michael Hamann
Prasad Imandi
Ron Lotter
Philip Norton
Christian Ringler
Gabriel Telerman

Redbooks

ibm.com/redbooks

International Technical Support Organization

Getting Started with WebSphere Enterprise Service Bus V6

April 2006

SG24-7212-00

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

First Edition (April 2006)

This edition applies to WebSphere Integration Developer V6.0.1 and WebSphere Enterprise Service Bus V6.0.1.

This document created or updated on May 4, 2006.

Contents

Notices

Trademarks

Preface

IBM® WebSphere® Enterprise Service Bus is a flexible connectivity infrastructure for integrating applications and services, designed to enable the development of a service-oriented architecture (SOA).

This IBM Redbook guides you through the capabilities and product features of WebSphere Enterprise Service Bus V6.0. It also contains many step-by-step examples of building resources for WebSphere Enterprise Service Bus using WebSphere Integration Developer.

Part 1 of this book introduces WebSphere Enterprise Service Bus and positions it among IBM's other SOA and Enterprise Service Bus product offerings.

Part 2 describes how to install and configure both WebSphere Enterprise Service Bus and WebSphere Integration Developer, and explains how to perform key concepts and tasks using these products.

Part 3 explains the administration and testing capabilities, including step-by-step examples.

Part 4 provides a wealth of development examples showing step-by-step how to develop solutions using mediation primitives, integrate with services, and deliver qualities of service.

The team that wrote this redbook

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

Figure 0-1 The redbook team (left-to-right): Martin, Prasad, Ron, Bill, Christian, Gabriel, Antonio, Phil, Michael

Martin Keen is a Senior IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products, SOA, and Patterns for e-business. He also teaches IBM classes worldwide about WebSphere, SOA, and business process management. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, UK. Martin holds a bachelor's degree in Computer Studies from Southampton Institute of Higher Education.

Bill Moore is a WebSphere specialist at the International Technical Support Organization, Raleigh Center. He writes extensively and teaches classes on WebSphere and related topics. Before joining the ITSO, Bill was a Senior AIM Consultant at the IBM Transarc laboratory in Sydney, Australia. He has 21 years of application development experience on a wide range of computing platforms and using many different coding languages. He holds a Master of Arts degree in English from the University of Waikato, in Hamilton, New Zealand. His current areas of expertise include application development tools, object-oriented programming and design, and e-business application development.

Antonio Carvalho is a System Specialist at IBM Information Technology Services Delivery in Brazil. He joined IBM in 1996 and has been working with Problem Management solutions, and Web developing. He holds a degree in Technology from (FASP) Faculdades Associadas de Sao Paulo in Sao Paulo, Brazil. His areas of expertise include experience in object-oriented programming, J2EE™, and application integration.

Michael Hamann is an IT Specialist at IBM Software Services for WebSphere, Software Education in Germany. He holds a degree in Geography from the University of Tuebingen, Germany. He has worked for IBM for 7 years. During this time he worked as an instructor, course developer, and consultant. His areas of expertise include WebSphere MQ, WebSphere Message Broker, WebSphere Adapters, and WebSphere Process Server.

Prasad Imandi is an Advisory Software Engineer in Level 2 support, AIM organization in RTP, NC. He has worked for IBM for 11 years working with WebSphere MQ, WebSphere Message Broker products, and lately been working with the Process Choreography component. He holds a Master degree in Computer Science and Engineering from Jadavpur University, India. His current area of expertise is WebSphere Message Broker.

Ron Lotter is a Senior Software Engineer in the Software Services for WebSphere organization in RTP, NC. He has worked for IBM for 23 years holding various management and technical positions and has 7 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 J2EE development.

Philip Norton is a Software Engineer in the WebSphere organization in Hursley, UK. He has 4 years of experience servicing WebSphere MQ JMS and developing WebSphere Enterprise Service Bus. He holds a degree in Computer Science from the University of Kent at Canterbury. His areas of expertise include WebSphere MQ, WebSphere ESB, Java™ programming, including JMS and J2EE development and Application Integration.

Christian Ringler is a certified Senior IT architect and Senior Consultant at IBM Business Consulting Services in Germany. He has worked for five years at IBM Global Services. Before joining IBM he worked six years for Oracle Consulting. He holds a masters degree in Computer Science from the Friedrich-Alexander University of Erlangen-Nuremberg. His areas of expertise include J2EE architectures, business process integration and the introduction of service-oriented architecture concepts in client environments, particularly the automotive industry.

Gabriel Telerman is an IT Specialist with IBM Software Services for WebSphere in the UK. He has 7 years of experience with IBM. He holds a degree in Computer Studies from Glasgow Caledonian University. His areas of expertise include object-oriented programming, enterprise Java, Web development, Web services, service-oriented architecture and enterprise application integration.

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

Chris Tomkins, Jon Martin, Calum Byrom, James Hodgson, Nigel Daniels, and Amanda Watkinson
IBM Hursley Lab, UK

Simon Kapadia
IBM Software Group, UK

Sunita Chacko
IBM Toronto Lab, Canada

Geoffrey Beers
IBM Rochester Lab, USA

Wolfgang Berger
IBM Business Consulting Services, Germany

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Part 1

Product overview

1

Welcome to this redbook

This chapter introduces this redbook to you, and provides guidelines for how to read it. It contains the following sections:

- An introduction to this document
- How to read this redbook

1.1 An introduction to this document

A warm welcome to this redbook, from the IBM Redbook team. We all assembled for five intense weeks in Raleigh, North Carolina to put together this resource. We hope you find it a useful read.

This redbook is aimed at integration developers, IT architects, and system administrators. It discusses how WebSphere Enterprise Service Bus fits into IBM's SOA and Enterprise Service Bus strategy, and gives an in-depth overview of the WebSphere Enterprise Service Bus product features.

We have also spent considerable time constructing a wealth of development examples which provide you with step-by-step instructions for building almost everything that WebSphere Enterprise Service Bus has to offer. You can follow along with these examples, or import completed solutions from the additional material supplied with this redbook.

1.2 How to read this redbook

As much as we would love you to read every page of this book cover-to-cover, we anticipate you might not quite have the time! To help you locate the information you need, and to provide guidance on which chapters are of most interest to you, this section provides a short description of each chapter.

Part 1. Product overview

This part introduces readers to the WebSphere Enterprise Service Bus offerings, and SOA related technologies in general.

- Chapter 1. Welcome to this redbook
- Chapter 2. Key technologies and concepts

  Provides an overview of SOA, Web services, Enterprise Service Bus, Service Component Architecture, and Service Data Objects.
- Chapter 3. WebSphere Enterprise Service Bus overview and product positioning

  Introduces the product features of WebSphere Enterprise Service Bus and related products, and positions these product features alongside WebSphere Message Broker.

Part 2. Configuration and usage

This part describes how to install and configure a development and runtime environment and the key concepts, terminology, and tasks in using WebSphere

Enterprise Service Bus and WebSphere Integration Developer. This part is aimed primarily at integration developers and system administrators.

- Chapter 4. Setting up the development environment

  Describes the installation options for WebSphere Integration Developer, and discusses configuration issues including considerations for building an environment for team development.

- Chapter 5. Setting up the runtime environment

  Describes the installation options for WebSphere Enterprise Service Bus and how to plan for test and production environments.

- Chapter 6. WebSphere Enterprise Service Bus key concepts and related technologies

  Explains the key terminology used in WebSphere Enterprise Service Bus, focusing on the mediation capabilities. Defines terms such as mediation flow, and mediation primitive.

- Chapter 7. WebSphere Integration Developer key concepts and common tasks

  Describes step-by-step how to use WebSphere Integration Developer to complete common tasks, ranging from navigating the Business Integration perspective, to building mediation flows.

Part 3. Administration and testing

This part explains many of the key system administration capabilities of WebSphere Enterprise Service Bus, and the testing and debugging capabilities of WebSphere Integration Developer.

- Chapter 8. Testing, debugging, and problem determination

  Describes how to test and debug mediation flows, and solve runtime problems.

- Chapter 9. Administering WebSphere Enterprise Service Bus

  Describes the administration capabilities and tasks specific to WebSphere Enterprise Service Bus, including the deployment and management of mediation modules.

Part 4. Development examples

This part of the redbook provides step-by-step instructions for building solutions in WebSphere Integration Developer for WebSphere Enterprise Service Bus. Each WebSphere Enterprise Service Bus product feature is explained separately in its own development example. This section is aimed at integration developers who want in-depth hands-on instructions on how to build solutions for WebSphere Enterprise Service Bus.

- Chapter 10. Preparing for the development examples

  The development examples all use a common set of services. You must complete the steps in this chapter to prepare your WebSphere Integration Developer workspace before attempting any of the development examples in this redbook.

- Chapter 11. Developing integration logic using mediation modules

  Step-by-step development examples for importing Web services, SCA services, and Enterprise Information Systems into mediation modules. Instructions for creating Web service, SCA, and JMS clients to mediation modules. Examples of how to manipulate service calls in a mediation module including changing bindings, request and response flows, and fault handling.

- Chapter 12. Developing mediation logic using mediation primitives

  Step-by-step development examples for each of the mediation primitives shipped with WebSphere Enterprise Service Bus. Includes development examples for the XSL Transformation, Database Lookup, Message Filter, Message Logging, Stop, Fail, and Custom mediation primitives.

- Chapter 13. Configuring modules to provide quality of service

  Step-by-step development examples of how to configure and browse CEI events, apply security, and use transactions.

Part 5. Appendixes

- Appendix A. Additional materials

  Solutions are provided for each of the development examples in Part 4. Additionally, many of the development examples require resources supplied with this redbook. This appendix describes how to locate the redbook additional material.

- Appendix B. Hints and tips

  Lists a few obstacles the team ran into while creating the development examples for this redbook and workarounds for them.

2 Key technologies and concepts

This chapter describes the key technologies and concepts that apply to architecting and building solutions in WebSphere Enterprise Service Bus.

This chapter contains the following sections:

- Service-oriented architecture (SOA)
- Web services
- Enterprise Service Bus (ESB)
- Service Component Architecture (SCA)
- Service Data Objects (SDO)

2.1 Service-oriented architecture

Service-oriented architecture (SOA) is an approach to defining integration architectures based on the concept of a *service*. Applications collaborate by invoking each others services and services can be composed into larger sequences to implement business processes.

Drivers for SOA

The main driver for SOA is to define an architectural approach that assists in the flexible integration of IT systems. Organizations spend a considerable amount of time and money trying to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The drivers behind this objective include:

- Increasing the speed at which businesses can implement new products and processes, can change existing ones, or can recombine them in new ways
- Reducing implementation and ownership costs of IT systems and the integration between them
- Enabling flexible pricing models by outsourcing more fine-grained elements of the business than were previously possible or by moving from fixed to variable pricing, based on transaction volumes
- Simplifying the integration work that is required by mergers and acquisitions
- Achieving better IT utilization and return on investment
- Achieving implementation of business processes at a level that is independent from the applications and platforms that are used to support the processes

SOA prescribes a set of design principles and an architectural approach to achieve this rapid flexible integration. In the following sections we provide an overview of some of the elements in SOA that achieve this aim.

Definition of SOA

SOA is an integration architecture approach based on the concept of a service. The business and infrastructure functions that are required to build distributed systems are provided as services that collectively, or individually, deliver application functionality to either end-user applications or other services.

SOA specifies that within any given architecture, there should be a consistent mechanism for services to communicate. That mechanism should be loosely coupled and support the use of explicit interfaces.

SOA brings the benefits of loose coupling and encapsulation to integration at an enterprise level. It applies successful concepts proved by Object Oriented development, Component Based Design, and Enterprise Application Integration technology to an architectural approach for IT system integration.

Services are the building blocks to SOA, providing function out of which distributed systems can be built. Services can be invoked independently by either external or internal service consumers to process simple functions, or can be chained together to form more complex functionality and so to quickly devise new functionality.

By adopting an SOA approach and implementing it using supporting technologies, companies can build flexible systems that implement changing business processes quickly, and make extensive use of reusable components (Figure 2-1).

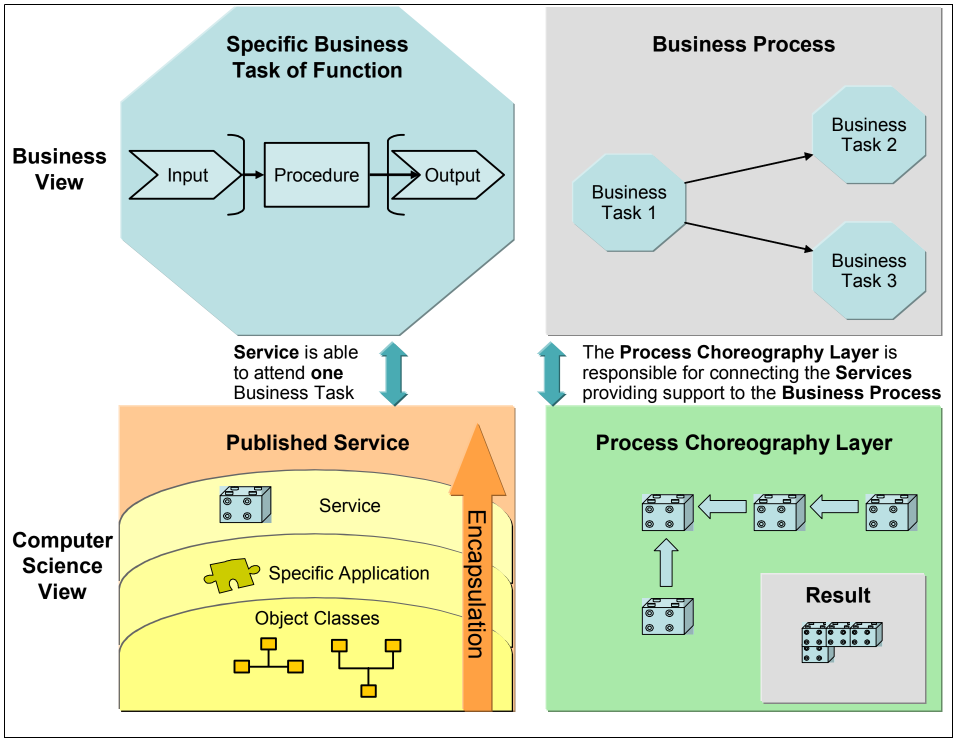

Business View
Specific Business Task of Function
Input
Procedure
Output
Business Process
Business Task 1
Business Task 2
Business Task 3
Service is able to attend one Business Task
The Process Choreography Layer is responsible for connecting the Services providing support to the Business Process
Computer Science View
Published Service
Service
Specific Application
Object Classes
Encapsulation
Process Choreography Layer
Result

Figure 2-1 Mapping services with business tasks or functions

2.1.1 What is a service?

Having outlined SOA as being an architectural approach to defining integration architectures based on services, it is important to define what is meant by a "service" in this context in order to fully describe SOA and understand what can be achieved by using it

A service can be defined as any discrete function that can be offered to an external consumer. This can be an individual business function, or a collection of functions that together form a process.

There are many additional aspects to a service that must also be considered in the definition of a service within a SOA. The most commonly agreed-on aspects are:

- Services encapsulate reusable business function
- Services are defined by explicit, implementation-independent interfaces
- Services are invoked through communication protocols that stress location transparency and inter operability

Reusable function

A service can be any business function. In an SOA however it is preferable that the function is genuinely reusable. The goal of a service in service-oriented architecture is that it can be used and reused by one or more systems that participate in the architecture. For example, while the reuse of a Java logging API could be described as “design time” (when a decision is made to reuse an available package and bind it into application code), the intention of service-oriented architecture is to achieve the reuse of services at:

- Runtime

  Each service is deployed in one place and one place only, and remotely invoked by anything that must use it. The advantage of this approach is that changes to the service (for example, to the calculation algorithm or the reference data it depends on) need only be applied in a single place.

- Deployment time

  Each service is built once but redeployed locally to each system or set of systems that must use it. The advantage of this approach is increased flexibility to achieve performance targets or to customize the service (perhaps according to geography).

Explicit implementation independent interfaces

The use of explicit interfaces to define and encapsulate service function is of particular importance to making services genuinely reusable. The interface should encapsulate only those aspects of process and behavior that are used in the interaction between the service consumer and the service provider. An explicit interface definition, or contract, is used to bind a service consumer and a service provider. It should specify only the mutual behavior required for the interaction, and nothing about the implementation of the consumer or the provider.

By explicitly defining the interaction in this way, those aspects of either system (for example the platform they are based on) that are not part of the interaction are free to change without affecting the other system. This allows either system to change implementation or identity freely.

Communication protocols that stress location transparency

SOA does not specify that any specific protocol should be used to provide access to a service. A key principle in SOA is that a service is not defined by the communication protocol that it uses, but instead should be defined in a protocol independent way that could allow different protocols to be used to access the same service.

Ideally a service should only be defined once, through a service interface, and have many implementations with different access protocols. This helps to increase the reusability of any service definition.

2.2 Web services

This section describes the core technologies of Web services, as well as how Web services are used in a service-oriented architecture.

2.2.1 Core technologies of Web services

Web services are self-contained, modular applications, that can be described, published, located, and invoked over networks. Web services encapsulate business functions, ranging from a simple request-reply to full business process interactions. The services can be new or wrap around existing applications.

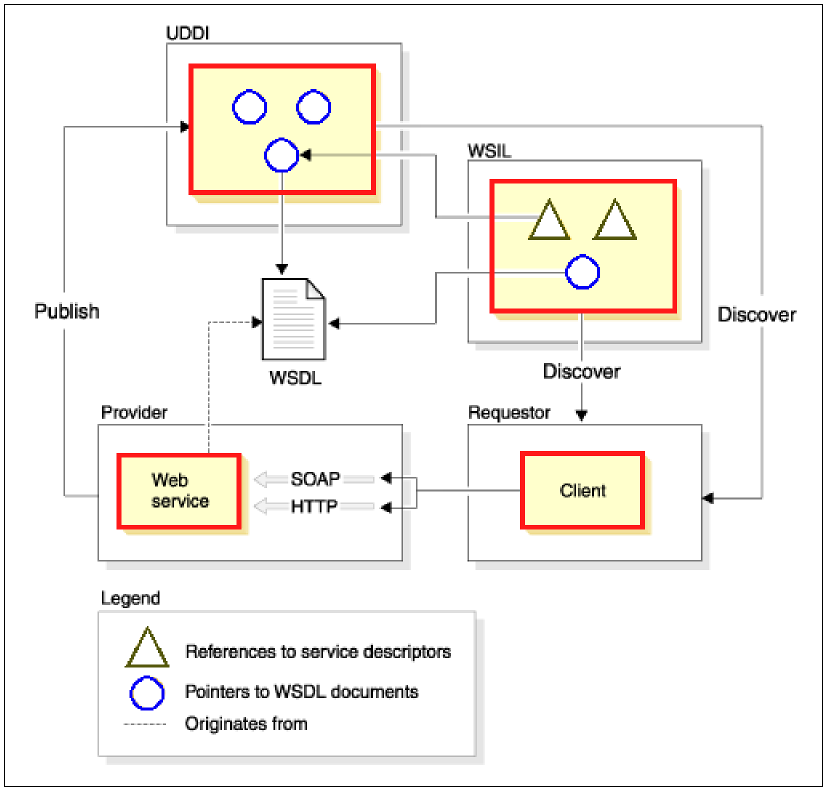


Figure 2-2 Main building blocks in an SOA approach based on Web services

Figure 2-2 shows the relationship between the core elements of Web services in an SOA.

The following are the core technologies used for Web services.

- **XML:** Extensible Markup Language) is the markup language that underlies most of the specifications used for Web services. XML is a generic language that can be used to describe any kind of content in a structured way, separated from its presentation to a specific device.
- **SOAP**: (Simple Object Access Protocol) is a network, transport, and programming language and platform-neutral protocol that allows a client to call a remote service. The message format is XML.
- **WSDL**: (Web Services Description Language) is an XML-based interface and implementation description language. The service provider uses a WSDL document in order to specify the operations a Web service provides and the parameters and data types of these operations. A WSDL document also contains the service access information.

- **WSIL:** (Web Services Inspection Language) is an XML-based specification about how to locate Web services without the necessity of using UDDI. However, WSIL can be also used together with UDDI, that is, it is orthogonal to UDDI and does not replace it.
- **UDDI:** (Universal Description, Discovery, and Integration) is both a client-side API and a SOAP-based server implementation that can be used to store and retrieve information on service providers and Web services.

2.2.2 Properties of Web services

All Web services share the following properties:

- **Web services are self-contained:** On the client side, no additional software is required. A programming language with XML and HTTP client support is enough to get you started. On the server side, merely an HTTP server and a SOAP server are required. It is possible to enable an existing application for Web services without writing a single line of code.
- **Web services are self-describing:** The definition of the message format travels with the message; no external metadata repositories or code generation tools are required.
- **Web services can be published, located, and invoked across the Web:** This technology uses established lightweight Internet standards such as HTTP. It leverages the existing infrastructure. Some additional standards that are required to do so include SOAP, WSDL, and UDDI.
- **Web services are modular:** Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation. Web services can be chained together to perform higher-level business functions. This shortens development time and enables best-of-breed implementations.
- **Web services are language-independent and interoperable:** The client and server can be implemented in different environments. Existing code does not have to be changed in order to be Web service enabled. Basically, any language can be used to implement Web service clients and servers. In this redbook we will only cover the use of Java for Web services.
- **Web services are inherently open and standard-based:** XML and HTTP are the major technical foundation for Web services. A large part of the Web service technology has been built using open-source projects. Therefore, vendor independence and interoperability are realistic goals.
- **Web services are loosely coupled:** Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that allows a more flexible reconfiguration for an integration of the services in question.

- **Web services are dynamic:** Dynamic e-business can become reality using Web services, because with UDDI and WSDL, the Web service description and discovery can be automated. In addition, Web services can be implemented and deployed without disturbing clients that use them.
- **Web services provide programmatic access:** The approach provides no graphical user interface; it operates at the code level. Service consumers have to know the interfaces to Web services but do not have to know the implementation details of services.
- **Web services provide the ability to wrap existing applications:** Already existing stand-alone applications can easily be integrated into the service-oriented architecture by implementing a Web service as an interface.
- **Web services build on proven, mature technology:** There are a lot of commonalities, as well as a few fundamental differences, with other distributed computing frameworks.

2.2.3 Web services and SOA

SOA represents a conceptual architecture of how to integrate applications. Web services are a specific set of standards and specifications that are one method of enabling SOA.

There are many logical links between Web services and SOA that suggest they are complementary:

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

Working together, Web services and SOA have the potential to address many of the technical issues that are faced when trying to build an on demand environment (Figure 2-3 on page 13).

.

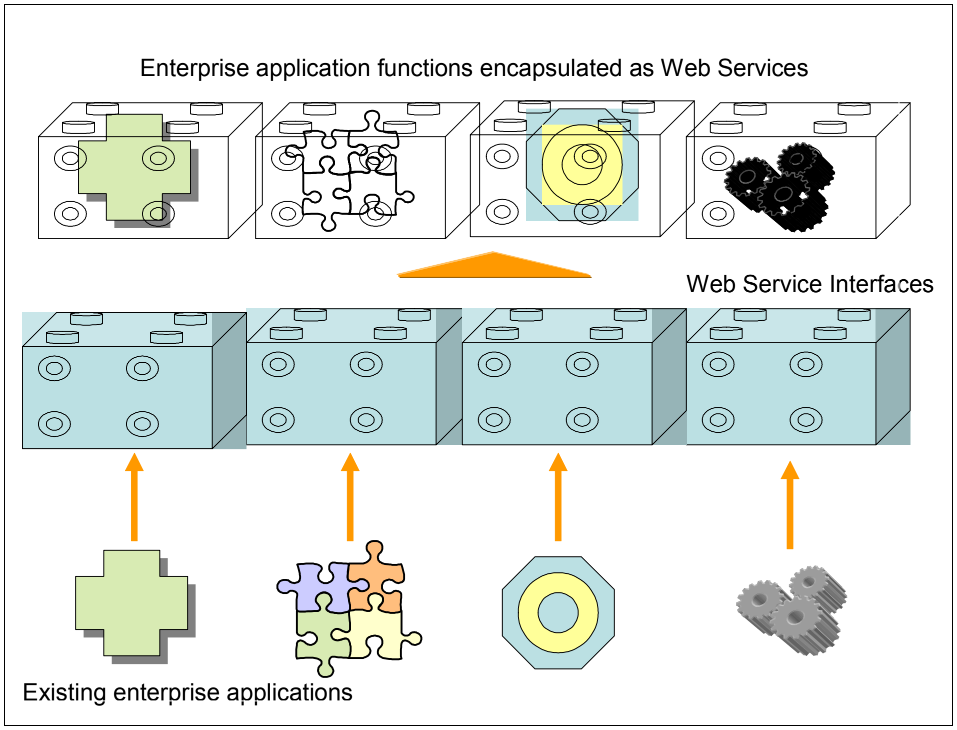

Enterprise application functions encapsulated as Web Services
Web Service Interfaces
Existing enterprise applications

Figure 2-3 Enterprise applications encapsulated as Web services

2.3 Enterprise Service Bus

The Enterprise Service Bus (ESB) is emerging as a middleware infrastructure component that supports the implementation of SOA within an enterprise. The need for an ESB can be seen by considering how it can support the concepts of SOA implementation by:

- Decoupling the consumer's view of a service from the actual implementation of the service
- Decoupling technical aspects of service interactions
- Integrating and managing services in the enterprise

This is achieved by replacing direct connections between service consumers and providers, with a hub and spoke architecture (Figure 2-4 on page 14).

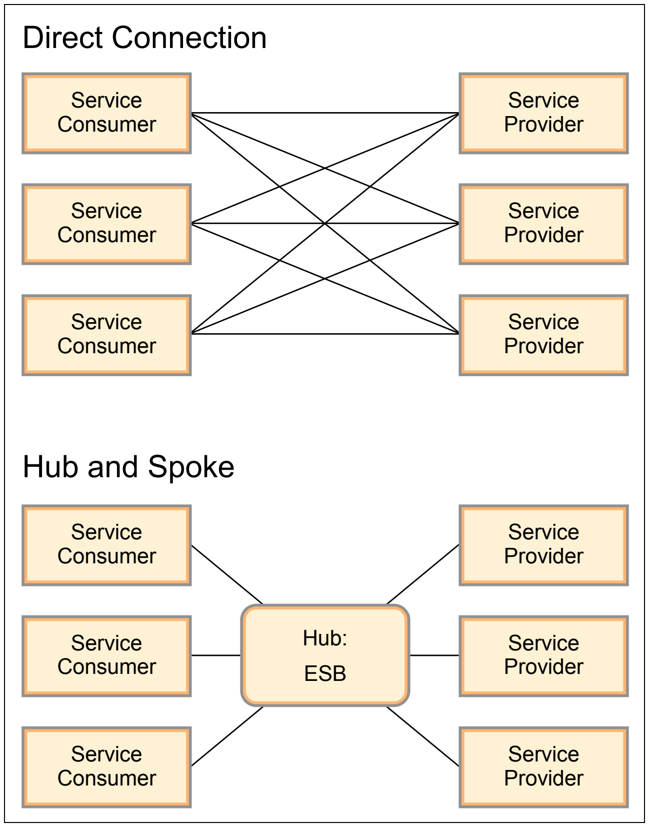


Figure 2-4 Direct connection and central hub integration styles

The ESB can be used to perform some of the following middleware functions:

- Map service requests from one protocol and address to another
- Transform data formats
- Support a variety of security and transactional models between service consumers and service providers and recognize that consumers and providers may support or require different models
- Aggregate or disaggregate service requests and responses
- Support communication protocols between multiple platforms with appropriate qualities of service
- Provide messaging capabilities such as message correlation and publish / subscribe, to support different messaging models such as events and asynchronous request/response

2.3.1 Enterprise requirements for an ESB

Using an ESB to implement an SOA has a number of advantages. In an SOA services should, by definition, be reusable by a number of different consumers, so the benefits of reduced connections are achieved. In addition the ESB:

- Supports high volumes of individual interactions.
- Support more established integration styles, such as message-oriented and event-driven integration, to extend the reach of the SOA. The ESB should allow applications to be SOA enabled either directly or through the use of adapters.
- Support centralization of enterprise-level qualities of service and manageability requirements into the hub.

Figure 2-5 shows a high-level view of the ESB.

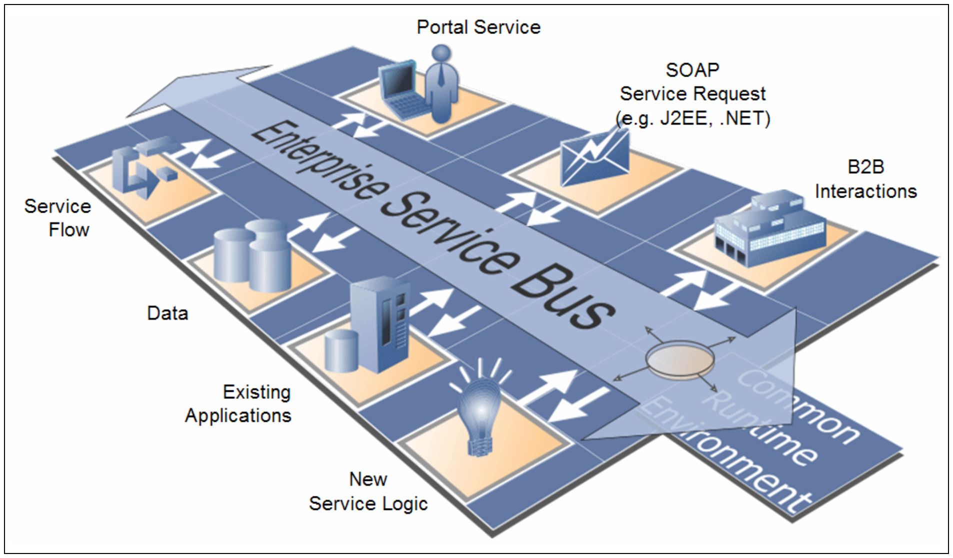


Figure 2-5 The Enterprise Service Bus

Mediation support

The ESB is more than just a transport layer. It must provide mediation support to facilitate service interactions (for example, to find services that provide capabilities that a consumer is asking for, or to take care of interface mismatches between consumers and providers that are compatible in terms of their capabilities). It must support a variety of ways to get on and off the bus, such as adapter support for legacy applications or business connections that enable external partners in business to business interaction scenarios. To do this it must support service interaction with a wide variety of service endpoints. It is likely that each endpoint will have its own integration techniques, protocols, security models and so on. This level of complexity should be hidden from service consumers.They need to be offered a simpler model. In order to achieve this, the ESB is required to mediate between the multiple interaction models understood by service providers and the simplified view provided to consumers.

Protocol independence

Services can be offered by a variety of sources. Without an ESB infrastructure any service consumer that needed to invoke a service would need to connect directly to a service provider using the protocol, transport and interaction pattern used by the provider. With an ESB the infrastructure shields the consumer from the details of how to connect to the provider.

In an ESB there is no direct connection between the consumer and provider. Consumers access the ESB to invoke services and the ESB acts as an intermediary, passing the request to the provider using the appropriate protocol, transport and interaction pattern for the provider. This enables the ESB to shield the consumer from the infrastructure details of how to connect to the provider. The ESB should support several integration mechanisms all of which could be described as invoking services through specific addresses and protocols, even if in some cases the address is the name of a CICS® transaction and the protocol is a J2EE resource adapter integrating with the CICS Transaction Gateway. By using the ESB the consumers are unaware of how the service is invoked on the provider.

As the ESB removes the direct connection between service consumer and providers, an ESB enables the substitution of one service implementation by another with no effect to the consumers of that service. This means an ESB allows the reach of an SOA to extend to non-SOA enabled service providers. It can also be used to support migration of the non-SOA providers to using an SOA approach without impacting the consumers of the service.

Support for multiple interaction patterns

To fully support the variety of interaction patterns that are required in a comprehensive service-oriented architecture (for example, request / response, publish / subscribe and events), the ESB must support in one infrastructure the three major styles of Enterprise Integration:

- Service-oriented architectures in which applications communicate through reusable services with well-defined, explicit interfaces. Service-oriented interactions leverage underlying messaging and event communication models.
- Message-driven architectures in which applications send messages through the ESB to receiving applications.

- Event-driven architectures in which applications generate and consume messages independently of one another.

The ESB does this while providing additional capabilities to mediate or transform service messages and interactions, enabling a wide variety of behaviors and supporting the various models of coupling interaction.

2.3.2 Minimum ESB capabilities

In this section we discuss the minimum capabilities an ESB must have to support the requirements of an SOA enabling infrastructure component. This will allow us to assess the suitability of individual technologies or products for implementing an ESB by analyzing the functionality they offer to support the minimum ESB capabilities.

In discussions on ESB the most commonly agreed elements for defining an ESB are:

- The ESB is a logical architectural component that provides an integration infrastructure consistent with the principles of service-oriented architecture
- The ESB may be implemented as a distributed, heterogeneous infrastructure
- The ESB provides the means to manage the service infrastructure and the capability to operate in a distributed, heterogeneous environment

The minimum capabilities that an ESB should have in order to provide an infrastructure consistent with these elements, and so consistent with the benefits of service-oriented architecture, are summarized in Table 2-1 and discussed in more detail in subsections below.

Table 2-1 Minimum capabilities of a ESB

Category	Capabilities	Reasons
Communications	► Routing ► Addressing ► At least one messaging style (request / response, pub/sub) ► At least one transport protocol that is or can be made widely available	Provide location transparency and support service substitution

Category	Capabilities	Reasons
Integration	► Several integration styles or adapters ► Protocol transformation	Support integration in heterogeneous environments and support service substitution
Service interaction	► Service interface definition ► Service messaging model ► Substitution of service implementation	Support service-oriented architecture principles, separating application code from specific service protocols and implementations
Management	► Administration capability	► A point of control over service addressing and naming

Communication

The ESB needs to supply a communication layer to support service interactions. It should support communication through a variety of protocols. It should provide underlying support for message and event oriented middleware and integrate with existing HTTP infrastructure and other enterprise application integration (EAI) technologies. As a minimum capability the ESB should support at least the protocols that make sense given the requirements of a specific situation.

The ESB should be able to route between all these communication technologies through a consistent naming and administration model.

Service interaction

The ESB needs to support SOA concepts for the use of interfaces and support declaration service operations and quality of service requirements.

The ESB should also support service messaging models consistent with those interfaces, and be capable of transmitting the required interaction context, such as security, transaction or message correlation information.

Integration

The ESB should support linking to a variety of systems that do not directly support service-style interactions so that a variety of services can be offered in a heterogeneous environment.

This includes legacy systems, packaged applications and other EAI technologies. Integration technologies might be protocols (for example JDBC™, FTP, EDI) or adapters such as the J2EE Connector Architecture resource adapters or WebSphere Business Integration Adapters. It also includes service client invocation through client APIs for various languages (Java, C++, C#) and platforms (J2EE, .Net), CORBA and RMI.

Management

As with any other infrastructure component the ESB needs to have administration capabilities to allow it to be managed and monitored and so to provide a point of control over service addressing and naming.

In addition it should be capable of integration into systems management software.

2.3.3 Extended ESB capabilities

The minimum capabilities described in 2.3.2, “Minimum ESB capabilities” on page 17 can help assess the suitability of individual technologies or products for implementing an ESB, however it will only establish which technologies are candidates. The detailed requirements of any particular scenario drive additional ESB capabilities that can then be used to select specific appropriate products.

In particular, the following types of requirements are likely to lead to the use of more sophisticated technologies, either now or over time:

- Non-functional requirements such quality of service demands and service-level capabilities
- Higher-level service-oriented architecture concepts, such as a service directory, and transformations
- Advanced management capabilities such as system management and autonomic capabilities and intelligent capabilities
- Truly heterogeneous operation across multiple networks, multiple protocols, and multiple domains of disparate ownership

Table 2-2 on page 20 extends the ESB capabilities described in 2.3.2, “Minimum ESB capabilities” on page 17 to include additional ESB capabilities.

Table 2-2 Categorized ESB capabilities

Communication	**Service interaction**
► Routing ► Addressing ► Protocols and standards (HTTP, HTTPS) ► Publish / subscribe ► Response / request ► Fire & forget, events ► Synchronous and asynchronous messaging	► Service interface definition (WSDL) ► Substitution of service implementation ► Service messaging models required for communication and integration (SOAP, XML, or proprietary Enterprise Application Integration models) ► Service directory and discovery
Integration	**Quality of service**
► Database ► Legacy and application adapters ► Connectivity to enterprise application integration middleware ► Service mapping ► Protocol transformation ► Data enrichment ► Application server environments (J2EE and .Net) ► Language interfaces for service invocation (Java, C/C++/C#)	► Transactions (atomic transactions, compensation, WS-Transaction) ► Various assured delivery paradigms (WS-ReliableMessaging or support for Enterprise Application Integration middleware)
Security	**Service level**
► Authentication ► Authorization ► Non-repudiation ► Confidentiality ► Security standards (Kerberos, WS-Security)	► Performance (response time, throughput and capacity) ► Availability ► Other continuous measures that might form the basis of contracts or agreements
Message processing	**Management and autonomic**
► Encoded logic ► Content-based logic ► Message and data transformations ► Message / service aggregation and correlation ► Validation ► Intermediaries ► Object identity mapping ► Service / message aggregation ► Store and forward	► Administration capability ► Service provisioning and registration ► Logging ► Metering ► Monitoring ► Integration to systems management and administration tooling ► Self-monitoring and self-management

Modeling	Infrastructure Intelligence
► Object modeling ► Common business object models ► Data format libraries ► Public versus private models for business-to-business integration ► Development and deployment tooling	► Business rules ► Policy-driven behavior, particularly for service level, security and quality of service capabilities (WS-Policy) ► Pattern recognition

2.4 Service Component Architecture

Service Component Architecture (SCA) was developed to simplify the integration between business applications and the development of new services. SOA is an abstract way to interpret the services and his correlations, SCA is defined as the implementation of the SOA architecture. Its standards allow the creation of services and the integration between them.

SCA separates application business logic and the implementation details. It provides a model that defines interfaces, implementations, and references in a technology neutral way, letting you then bind these elements to any technology specific implementation. The ability to separate business logic from infrastructure logic reduces the IT resources needed to build an enterprise application, and gives developers more time to work on solving a particular business problem rather than focusing on the details of which implementation technology to use.

2.4.1 Anatomy of SCA

SCA provides an abstraction that covers stateless session EJBs, Web services, POJOs, WS-BPEL processes, database access, Enterprise Information System (EIS) access, and so on. SCA separates business logic from infrastructure logic so that application programmers can focus on the business problem. SCA covers both the usage of services and the development of services. It provides a uniform model for application programmers and for tools.

SCA is a universal model for business services that publish or operate on business data. Service Data Objects (SDO) provide the universal model for business data.

Figure 2-6 shows the main terms of an SCA component:

- Interface
- Implementation
- Reference

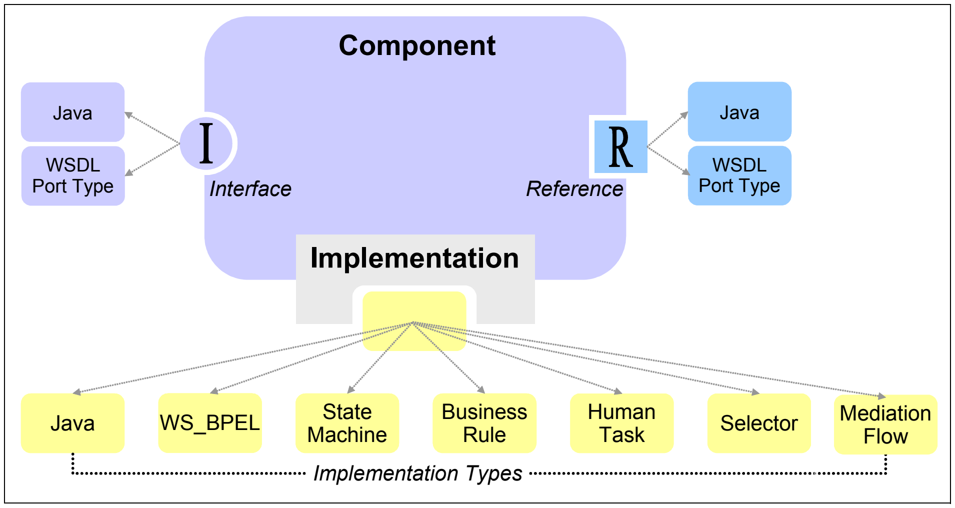


Figure 2-6 Service component: overview

A service interface is defined by a Java interface or WSDL Port Type. Arguments and return values are described with Java classes, simple Java types, or XML schema. SDO generated Java classes are the preferred form of Java class because of their integration with XML technologies. Arguments described in XML schema are exposed to programmers as SDOs.

A component exposes business-level interfaces to its application business logic so that the service can be used or invoked. The interface of a component defines the operations that can be called and the data that is passed, such as input arguments, returned values, and exceptions. An import and export also has interfaces so that the published service can be invoked.

All components have interfaces of the WSDL type. Only Java components support Java-type interfaces. If a component, import or export, has more than one interface, all interfaces must be the same type.

A component can be called synchronously or asynchronously; this is independent of whether the implementation is synchronous or asynchronous. The component interfaces are defined in the synchronous form and asynchronous support is also generated for them. You can specify a preferred interaction style as synchronous or asynchronous. The asynchronous type advertises to users of the interface that it contains at least one operation that can take a significant amount of time to complete. As a consequence, the calling service must avoid keeping a transaction open while waiting for the operation to complete and send its response. The interaction style applies to all the operations in the interface.

You can also apply a role-based permission qualifier to an interface so that only authorized applications can invoke the service with that interface. If the operations require different levels of permission for their use, you must define separate interfaces to control their access.

A service can be implemented in a range of languages (for example Java, WS-BPEL, state-machine definitions, and so on). When implementing a service, the focus is on the business purpose and less on infrastructure technology.

SCA and non-SCA services can use other service components in their implementations. They do not hard code the other services they use. They declare soft links called service references. Service wires resolve service references. You can use SCA wiring to create SCA applications by component assembly.

Figure 2-7 on page 24 shows a service component and a number of references. When a component wants to use the services of another component, it must have a partner reference or simply a reference. We can consider an in-line reference, which means that the referenced service component is defined within the same scope of the referencing component. In other words, both components are defined within the same module.

Applications that are not defined as SCA components (for example, JavaServer™ Pages™ (JSPs)) can still invoke SCA components; they do so through the use of stand-alone references. Stand-alone references contain partner references that identify the components to call. Alone, stand-alone references do not have any implementation or interface.

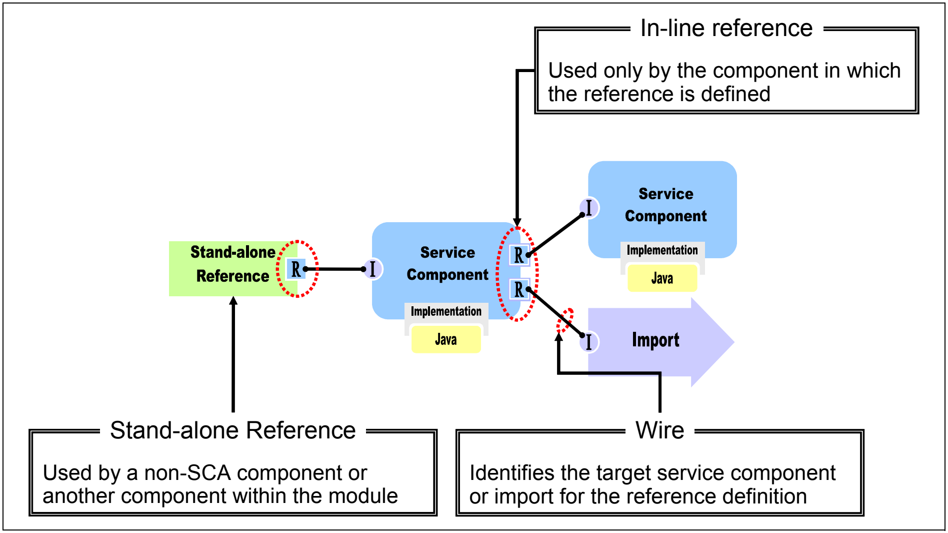

In-line reference
Used only by the component in which the reference is defined
Service Component
Implementation
Java
Stand-alone Reference
R
I
Service Component
Implementation
Java
R
R
I
Import
Stand-alone Reference
Used by a non-SCA component or another component within the module
Wire
Identifies the target service component or import for the reference definition

Figure 2-7 Service component and references

Components are assembled in a module (Figure 2-8): either a service module or a mediation module (which is specific to WebSphere Enterprise Service Bus).

.

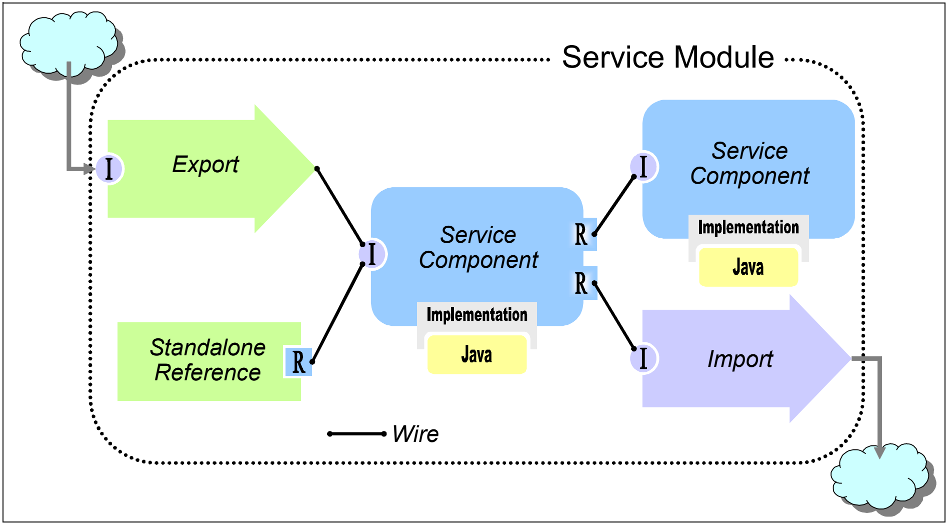


Figure 2-8 Service module: overview

The implementations of components that are used in a module assembly might reside within the module. Components that belong to other modules can be used through imports. Components in different modules can be wired together by publishing the services as exports that have their interfaces and dragging the exports into the required assembly diagram to create imports.

When wiring components, you can also specify quality of service qualifiers on the implementations, partner references, and interfaces of the component.

An *import* allows you to use functions that are not part of the module that you are assembling. Imports can be from components in other modules or non-SCA components such as stateless session EJBs and Web services. Available function (or business logic) that is implemented in remote systems (such as Web services, EIS functions, EJBs, or remote SCA components) is modeled as an *imported service* (Figure 2-9).

Imports have interfaces that are the same as or a subset of the interfaces of the remote service that they are associated with so that those remote services can be called. Imports are used in an application in exactly the same way as local components. This provides a uniform assembly model for all functions, regardless of their locations or implementations. The import binding does not have to be defined at development time; it can be done at deployment time.

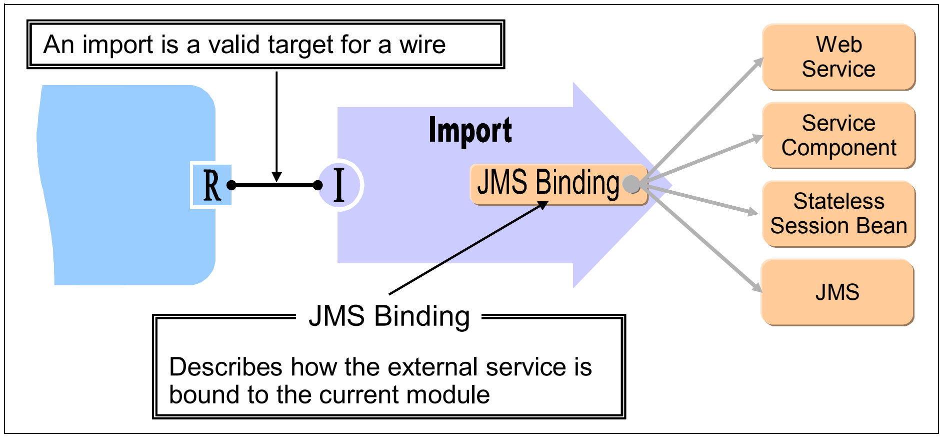

An import is a valid target for a wire
R
I
Import
JMS Binding
Web Service
Service Component
Stateless Session Bean
JMS
JMS Binding
Describes how the external service is bound to the current module

Figure 2-9 Service component and import

An *export* is a published interface from a component (Figure 2-10) that offers the component business service to the outside world, for example, as a Web service. Exports have interfaces that are the same as or a subset of the interfaces of the component that they are associated with so that the published service can be called. An export dragged from another module into an assembly diagram automatically creates an import.

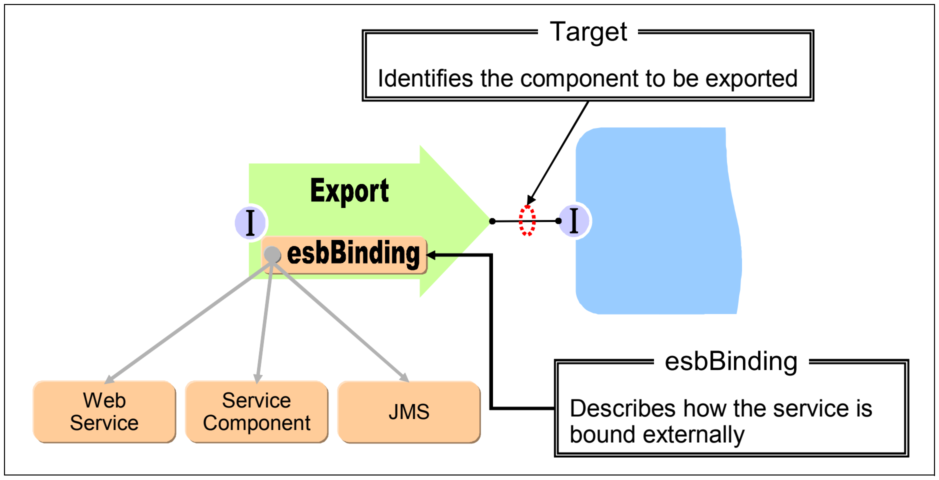

Target
Identifies the component to be exported
Export
I
esbBinding
I
Web Service
Service Component
JMS
esbBinding
Describes how the service is bound externally

Figure 2-10 Service component and export

2.5 Service Data Objects

Business data that is exchanged in an integrated application in WebSphere Enterprise Service Bus is represented by business objects. The objects are based on Service Data Objects (SDO), which is a new data access technology.

2.5.1 SDO concepts

There are a a few key SDO concepts that can provide a framework for understanding business object architecture, including the design and use of business objects in WebSphere Enterprise Service Bus.

The fundamental concept in the SDO architecture is the data object. In fact, the term SDO is often used interchangeably with the term data object. A data object is a data structure that holds primitive data, multi-valued fields (other data objects), or both. The data object also has references to metadata that provide information about the data found in the data object. In the SDO programming model, data objects are represented by the commonj.sdo.DataObject Java interface definition. This interface includes method definitions that allow clients to obtain and set the properties associated with DataObject.

As an example, consider modeling customer data with an SDO data object. The properties associated with the customer might be firstName (String), lastName (String), and customerID (long). The following pseudo code shows how you might use the DataObject API to obtain and set properties for the customer data object:

```
DataObject customer = …
customer.setString("firstName","John");
customer.setString("lastName","Doe");
customer.setInt("customerID", 123);
int id = customer.getInt(“customerID”);
```

Another important concept in the SDO architecture is the data graph. A data graph is a structure that encapsulates a set of data objects. From the top level data object in the graph, all other data objects can be reached by traversing the references from the root data object. In the SDO programming model, data graphs are represented by the commonj.sdo.DataGraph Java interface definition.

2.5.2 Applying SDO to SCA

Both SCA and SDO (the basis of business objects) have been designed to be complimentary service oriented technologies. Figure 2-11 illustrates how SCA provides the framework to define service components and to compose these services into integrated applications, and it further shows that business objects represent the data that flows between each service. Whether the interface associated with a particular service component is defined as a Java interface or a WSDL port type, the input and output parameters are represented by business objects.

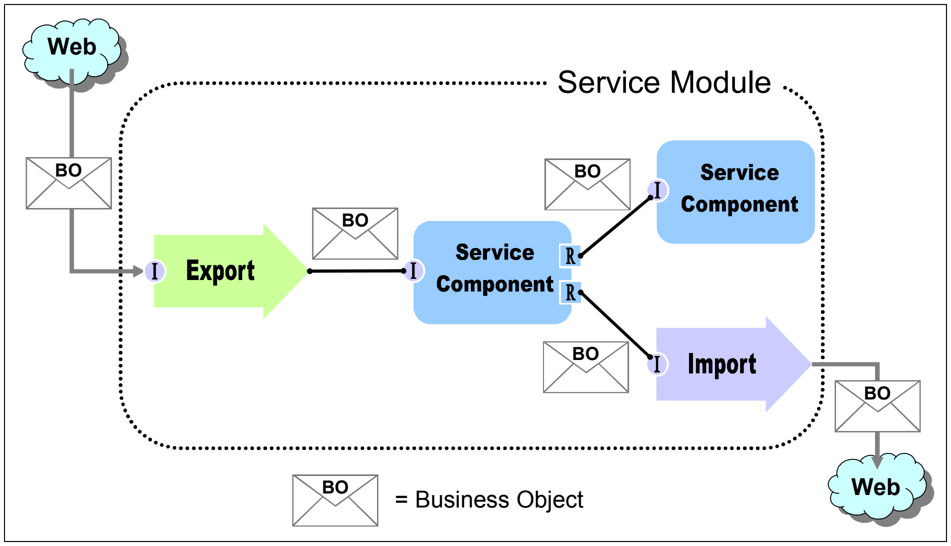


Figure 2-11 Exchanging data in an SCA runtime

3

WebSphere Enterprise Service Bus overview and product positioning

This chapter gives a general product overview of WebSphere Enterprise Service Bus and related products. The products discussed here are:

- WebSphere Application Server V6.0
- WebSphere Enterprise Service Bus V6.0
- WebSphere Process Server V6.0
- WebSphere MQ V6.0
- WebSphere Message Broker V6.0
- WebSphere Adapters V6.0
- Rational Application Developer V6.0
- WebSphere Integration Developer V6.0

Additionally, this chapter compares the ESB capabilities of WebSphere Enterprise Service Bus and WebSphere Message Broker to help you to determine which product may be appropriate in your situation.

3.1 Product overview

This section provides an overview of the functions of WebSphere Enterprise Service Bus and other related WebSphere products.

The IBM SOA Reference Architecture defines the IT services required to support an SOA. It includes development environment, services management, application integration, and runtime process services. The capabilities of the architecture can be implemented on a build-as-you-go basis as new requirements are addressed over time.

Figure 3-1 shows the IBM SOA reference architecture and the supporting software.

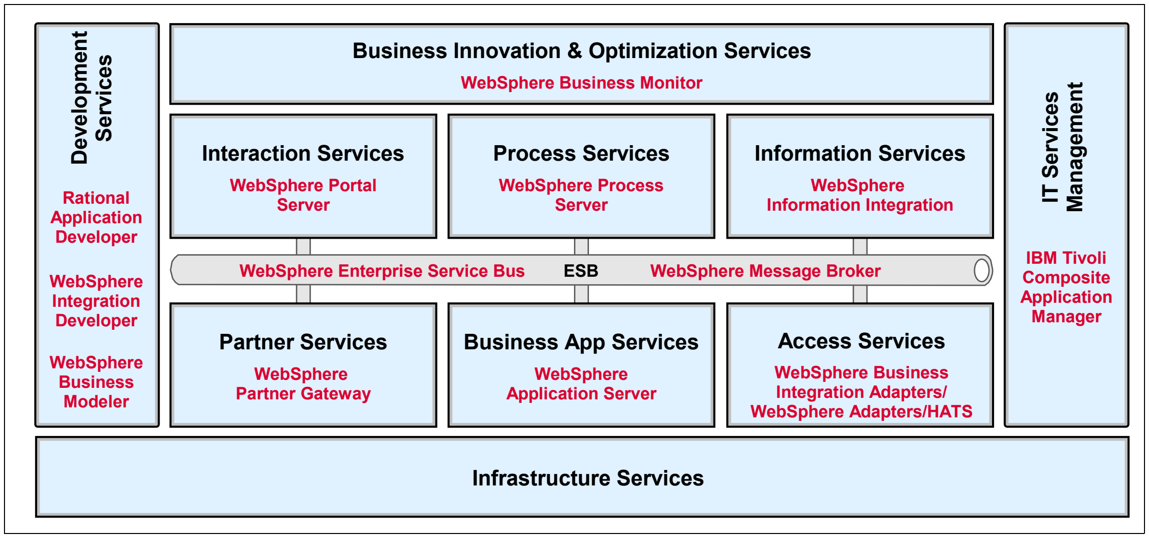

Development Services
Rational Application Developer
WebSphere Integration Developer
WebSphere Business Modeler
Business Innovation & Optimization Services
WebSphere Business Monitor
Interaction Services
WebSphere Portal Server
Process Services
WebSphere Process Server
Information Services
WebSphere Information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

Figure 3-1 IBM SOA Reference Architecture with product mapping

In this chapter we concentrate on the two products providing the ESB capabilities for an SOA and their related products:

1. **WebSphere Enterprise Service Bus** provides ESB functions for SOAs built on open standards. It is based on WebSphere Application Server Network Deployment and inherits its build-in messaging provider and quality of services. WebSphere Process Server is built on top of WebSphere Enterprise Service Bus and adds a business process runtime.
2. **WebSphere Message Broker** provides advanced ESB functionality for universal support of messaging applications. It is based on WebSphere MQ and takes advantage of the services provided by its messaging infrastructure.

Figure 3-2 shows the two main ESB products, their foundation and relationship.

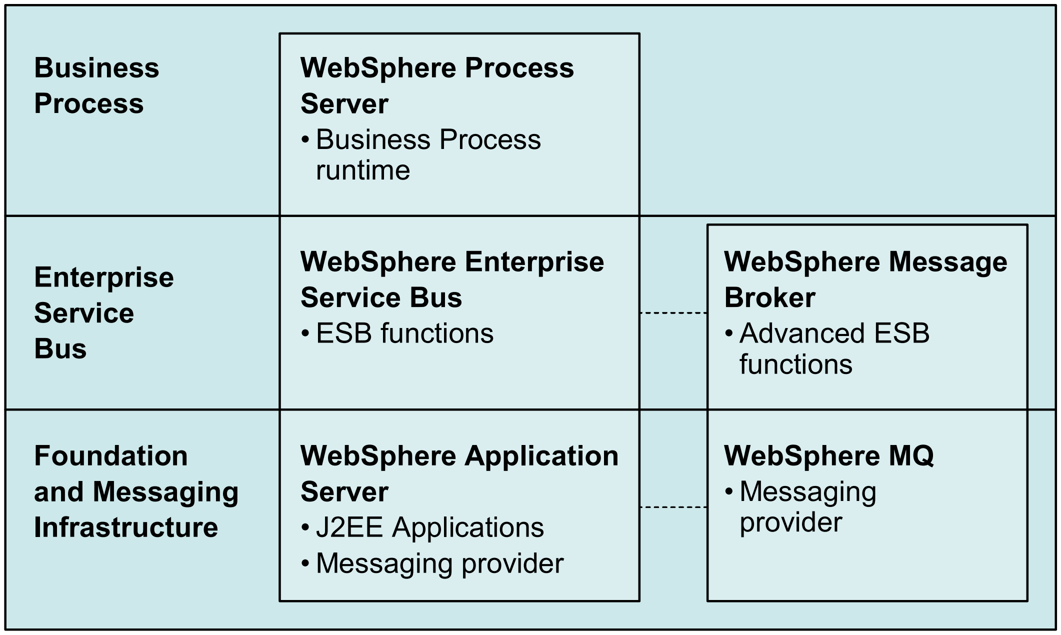


Figure 3-2 IBM ESB offerings

3.1.1 IBM WebSphere Application Server V6.0

The foundation of the WebSphere brand is the application server, which provides the runtime environment and management tools for J2EE and Web services based applications. WebSphere Application Server provides qualities of service such as clustering, failover, scalability, and security. It also includes a built-in messaging provider which can be configured to connect to an existing WebSphere MQ network.

WebSphere Application Server is available in three packages:

- WebSphere Application Server - Express

  The Express package is geared to those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation, and is focused on providing ease of use and ease of application development. It contains full J2EE 1.4 support but is limited to a single-server environment. WebSphere Application Server - Express is bundled with the Rational Web Developer application development tool.

- WebSphere Application Server

  The WebSphere Application Server package provides the next level of server infrastructure. Though the server is functionally equivalent to the server shipped with Express, this package differs slightly in packaging and licensing. The development tool included is a trial version of Rational Application Developer, a full J2EE 1.4 compliant development tool.

- WebSphere Application Server Network Deployment

  WebSphere Application Server Network Deployment is an even higher level of server infrastructure in the WebSphere Application Server family. It extends the WebSphere Application Server base package to include clustering capabilities, Edge components, and high availability for distributed configurations. These features become more important at larger enterprises, where applications tend to service a larger customer base, and more elaborate performance and availability requirements are in place.

WebSphere Application Server V6 provides full support for the J2EE 1.4 specification. The J2EE specification defines the concept of containers to provide runtime support for applications. There are three types of containers in the application server implementation:

- Web container

  The Web container processes HTML, servlets, JSP™ files and other types of server-side includes. It provides infrastructure support like Web container transport chains, session management and Web services engine.

- EJB™ container

  The EJB container provides all the runtime services that are needed to deploy and manage enterprise beans. Is is a server process that handles requests for both session and entity beans. The container provides low-level services including threading and transaction support.

- Client container

  The client container is a separately installed component on the client's machine. It allows the client to run applications in an environment that is compatible with J2EE.

In addition to the definition of containers as a runtime environment for application components, the application server supports the following features, prescribed by the J2EE:

- J2EE Connector Architecture
- Java Naming and Directory Interface™ (JNDI) name space
- Security: J2EE security, Java 2 security, JAAS

- JMS provider: WebSphere messaging (default messaging provider), WebSphere MQ JMS provider, generic JMS providers.
- Web services engine: WS-I Basic Profile, WS-Security, JAX-RPC, JAXR, SAAJ, UDDI.

With Network Deployment, clustering application servers automatically enables plug-in workload management for the application servers and the servlets they host. The routing is based on weights associated with the cluster members. If all cluster members have identical weights, the plug-in sends equal requests to all members of the cluster. Workload management for EJB containers can be performed by configuring the Web container and EJB containers on separate application servers. Multiple application servers with the EJB containers can be clustered, enabling the distribution of EJB requests between the EJB containers.

WebSphere Application Server Network Deployment also provides high availability features. The following is a quick overview of the failover capabilities:

- Web container failover

  The Web server plug-in in the Web server is aware of the configuration of all Web containers and can route around a failed Web container in a cluster. Sessions can be persisted to a database or in-memory using data replication services.

- EJB container failover

  Client code and the ORB plug-in can route to the next EJB container in the cluster.

- Critical services failover

  Hot standby and peer failover for critical services (such as workload management routing, PMI aggregation, JMS messaging, transaction manager, and so on) is provided through the use of high availability domains. A high availability domain defines a set of WebSphere processes (core group) that provides high availability function to each other.

  One or more members of the core group can act as a high availability coordinator, managing the high availability activities within the core group processes. If a high availability coordinator server fails, another server in the core group takes over the duties of that coordinator. High availability policies define how the failover occurs. Workload management information is shared between core members and failover of critical services is done among them in a peer-to-peer fashion. Little configuration is necessary, and in many cases, this function works with the defaults that are created automatically as you create the processes.

- JMS messaging failover

The messaging engine keeps messages in a remote database. When a server in a cluster fails, WebSphere selects an online server to run the messaging engine and the workload manager routes JMS connections to that server.

WebSphere Application Server provides a browser-based administrative console for administration. Command-line and scripting administration is also provided.

You can find more information about the WebSphere Application Server at:

- WebSphere Application Server home page:

  `http://www.ibm.com/software/webservers/appserv/was/`

3.1.2 IBM WebSphere Enterprise Service Bus V6.0

WebSphere Enterprise Service Bus is designed to provide an Enterprise Service Bus (ESB) for IT environments built around open standards and SOA. It delivers easy to use functionality built on the messaging and Web services technologies of WebSphere Application Server.

WebSphere Application Server is the foundation for WebSphere Enterprise Service Bus, providing not only the required quality of service, the J2EE runtime environment and the messaging engine but also by providing broad support regarding open standards and Web services. WebSphere Enterprise Service Bus is built on the Network Deployment package, providing a wide range of capabilities for large enterprise networks, including clustering, failover, and scalability features.

The development tool for WebSphere Enterprise Service Bus is WebSphere Integration Developer.

Architecture

WebSphere Enterprise Service Bus provides uniform invocation and data-representation programming models and monitoring capabilities for components running on WebSphere Enterprise Service Bus.

Service Component Architecture (SCA)

On top of the infrastructure provided by WebSphere Application Server, WebSphere Enterprise Service Bus implements a mediation layer consisting of a mediation base and mediation functions. The newly provided mediation framework is different from the one implemented by WebSphere Application Server as it is based on the Service Component Architecture (SCA). It allows enhanced flexibility, encapsulation and reuse. Mediations implemented for WebSphere Application Server can still be used together with WebSphere

Enterprise Service Bus but the new tooling provided for WebSphere Enterprise Service Bus does not support the modification of these mediations.

Service Data Objects (SDO)

Mediation components are typically concerned with the flow of messages through the infrastructure and not just with the business content of the message. The information that governs their behavior is often held in headers flowing with the business message. Therefore the Service Message Object (SMO) pattern for Service Data Objects is introduced to support this pattern. Service Message Objects are enhanced Service Data Objects, providing an abstraction layer for processing and manipulating messages exchanged between services.

Common Event Infrastructure (CEI)

WebSphere Enterprise Service Bus uses the Common Event Infrastructure (CEI) to provide event management services, such as event generation, transmission, persistence, and consumption. The format of those events is defined by the Common Base Event (CBE) specification.

Mediations

Mediations are provided by SCA and Service Message Objects (SMO). SCA supports the description of every mediation module through a technology-neutral interface. SMO is based on SDO and supports the representation of a binding-specific data format in a common, neutral way. The application of this SCA/SMO based programming model allows for the configurable assembly of different mediation modules containing the mediation flow, thus enabling a very flexible and encapsulated solution.

Mediation functions are built upon the mediation base and consist of one or more mediation modules. An SCA/SMO based mediation module is composed of different parts such as imports representing providers, exports representing service consumers and a mediation flow component representing integration and mediation functionality.

Figure 3-3 shows a mediation module acting on the flow of services requests between service consumers and providers.

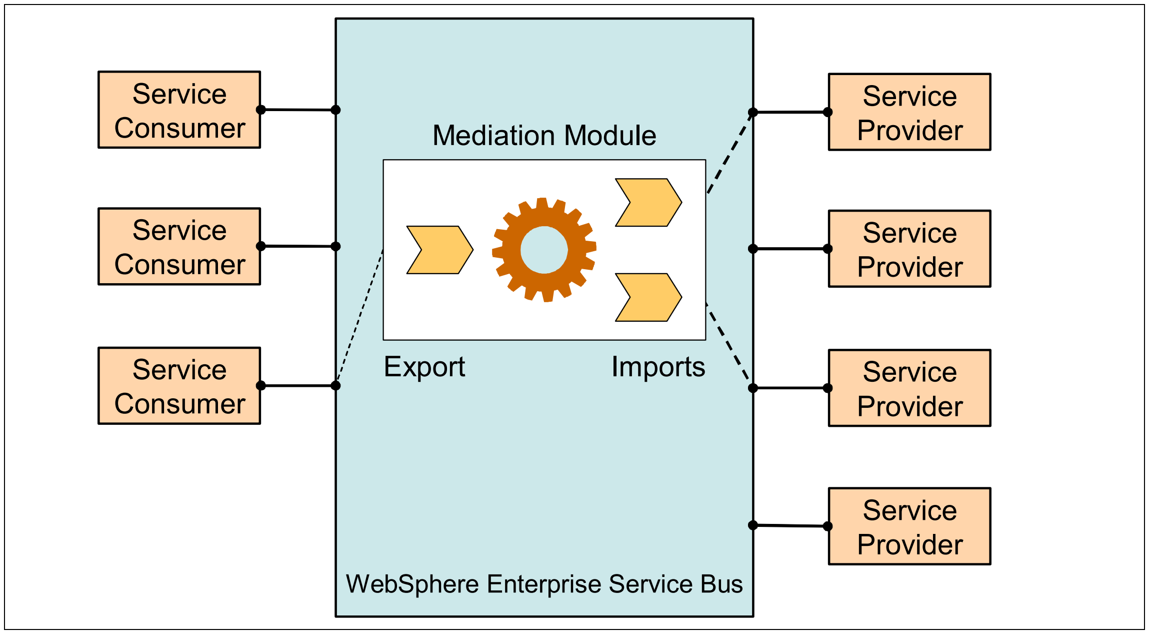


Figure 3-3 WebSphere Enterprise Service Bus mediation module

WebSphere Enterprise Service Bus provides prebuilt components called *mediation primitives* that can be used in mediation flows to perform XSLT message transformation, logging, routing, and database lookup. It also supports the implementation of custom mediation primitives.

WebSphere Enterprise Service Bus supports different binding types for imports and exports, thus allowing the connection of different kinds of service consumers and providers. Supported binding types are JMS binding, Web services binding, WebSphere adapter binding, EJB binding, as well as SCA binding used for module to module communication.

The mediation framework and its mediation modules separate the processing of requests from the processing of replies. They allow the mediation flow components to pass a potentially modified request from a service consumer to a service provider and to pass a potentially modified reply from a service provider to a service consumer. The request processing within a mediation flow component can send a reply back to the consumer without necessarily needing to contact a service provider.

Figure 3-4 shows a service consumer sending a request over the ESB. The ESB passes the request to the service provider. The service provider runs the service, then, optionally sends a reply to the consumer.

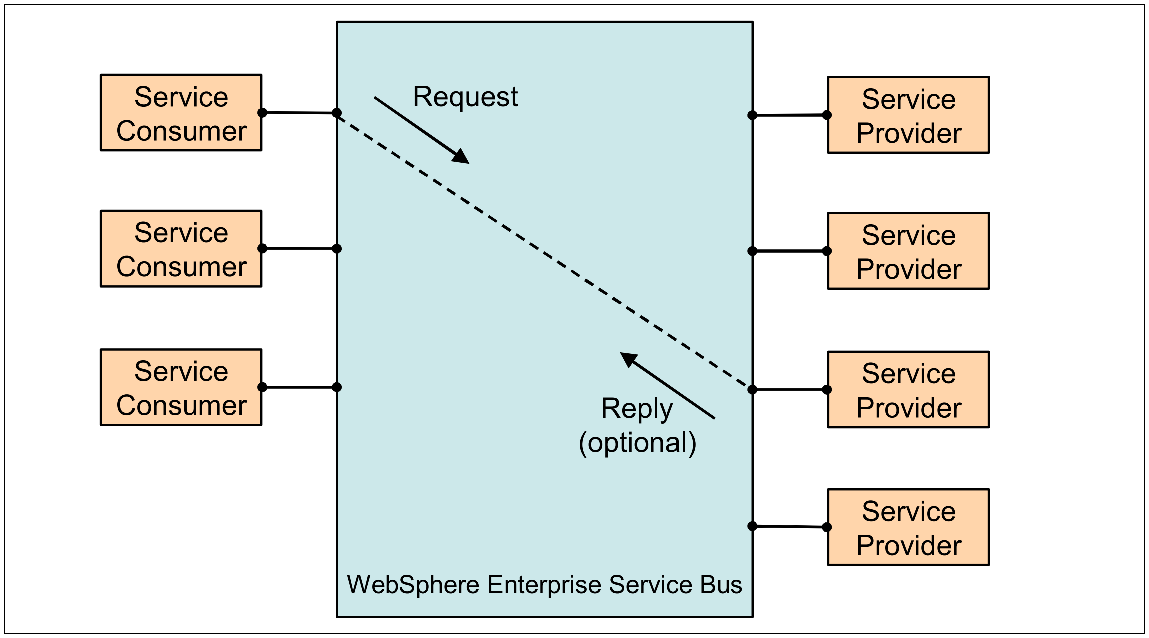


Figure 3-4 ESB passing request from consumer to provider

Clients

J2EE clients from WebSphere Application Server Network Deployment, including Web services clients, EJB clients, and JMS clients can be used to extend connectivity of the ESB. Additionally WebSphere Enterprise Service Bus provides Message Service Clients for C/C++ and .NET, Web services clients for C++, and SCA clients for Java.

Mediation functions in WebSphere Enterprise Service Bus versus WebSphere Application Server

Before the announcement of WebSphere Enterprise Service Bus, the service integration bus in WebSphere Application Server was often positioned as a basic ESB. Though this is still a useful strategy for development environments, WebSphere Enterprise Service Bus is now the recommended solution for environments where the service integration bus was used.

WebSphere Enterprise Service Bus adds the following functionality to the service integration bus.

- Easy to build mediation layer
- Simplified administration
- Pre-built mediation functions
- Broad connectivity

Mediation functions in WebSphere Enterprise Service Bus are service intermediaries that:

- Operate on interactions between service endpoints (consumer and provider)
- Are administered as part of WebSphere Enterprise Service Bus
- Are created using visual tooling exploiting supplied and custom mediation functions
- Have access to binding specific header data like SOAP and JMS headers

Mediation handlers in the WebSphere Application Server service integration bus are message handlers that:

- Operate on messages traversing the bus
- Are administered as part of the bus
- Are created by implementing Java programs
- Allow access to the full WebSphere messaging header information

You can find more information about the WebSphere Enterprise Service Bus at:

- WebSphere Enterprise Service Bus home page:

  `http://www.ibm.com/software/integration/wsesb/`

3.1.3 IBM WebSphere Process Server V6.0

WebSphere Process Server is built on WebSphere Enterprise Service Bus, thus providing it with the mediation functionality of WebSphere Enterprise Service Bus and the qualities of service that WebSphere Application Server provides, for example clustering, failover, scalability, and security. To this, WebSphere Process Server adds the ability to build business processes that orchestrate multiple services to achieve a business goal.

The WebSphere Process Server architectural model consists of the three layers, as shown in Figure 3-5.

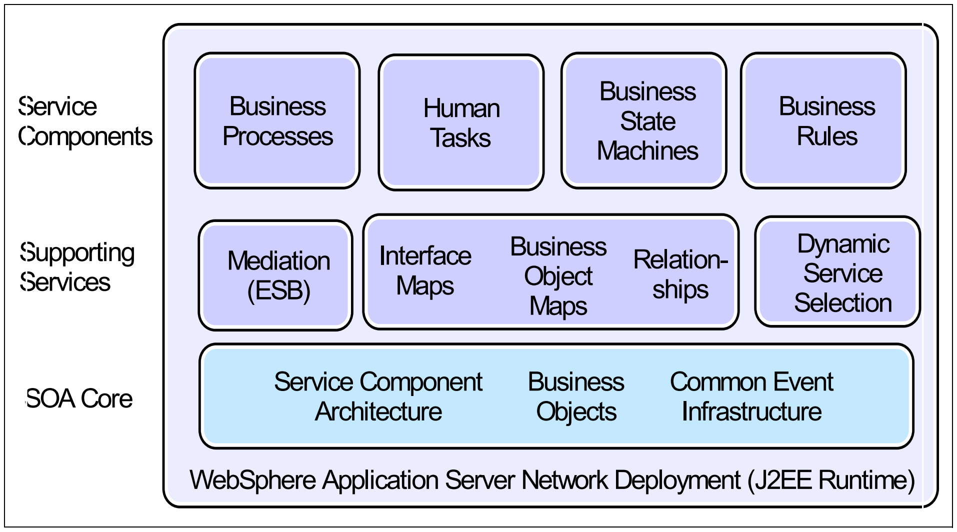


Figure 3-5 Architectural model of WebSphere Process Server

Above the infrastructure provided by WebSphere Application Server, WebSphere Process Server implements the SOA core layer, also used by WebSphere Enterprise Service Bus, that includes the following:

- Service Component Architecture (SCA)
- Business objects
- Common Event Infrastructure

On top of this SOA Core layer lies the service components and supporting services layers. WebSphere Process Server implements a number of components and services that can be used in an integration solution. In the service components layer you will find the following:

- Business processes

The business process component in WebSphere Process Server implements a WS-BPEL compliant process engine.

- Human tasks

  Human tasks in WebSphere Process Server are standalone components which can be used to assign work to employees or to invoke any other service.

- Business state machines

  A business state machine provides another way of modeling a business process. This enables businesses to represent their business processes based on states and events/

- Business rules

  Business rules are a means of implementing and enforcing business policy through externalizing of business function. This enables dynamic changes of a business process.

These components can use the features of a number of supporting services in WebSphere Process Server. Most of these can be classified as some form of transformation. There are a number of transformation challenges when connecting components and external services, each of which is being addressed by a component of WebSphere Process Server:

- Interface maps

  Very often interfaces of existing components match semantically but not syntactically. Interface maps allow the invocation of these components by translating these calls. Additionally business object maps can be used to translate the actual business object parameters of a service invocation.

- Business object maps

  A business object map is used to translate one type of business object into another type of business object.

- Relationships

  In business integration scenarios it is often necessary to access the same data in various backend systems for example an ERP system and a CRM system. A common problem for keeping business objects in sync is that different backend systems use different keys to represent the same objects. The relationship service in WebSphere Process Server can be used to establish relationship instances between objects in these disparate backend systems. These relationships are accessed from a business object map when translating one business object format into another.

- Dynamic service selection

A selector component allows dynamic selection and invocation of different services, which all share the same interface.

- Mediation

  This component is inherited from WebSphere Enterprise Service Bus.

The primary development tool for WebSphere Process Server is WebSphere Integration Developer. This is the same tool used for WebSphere Enterprise Service Bus development tasks.

You can find more information about IBM WebSphere Process Server V6 at:

- WebSphere Process Server home page:

  `http://www.ibm.com/software/integration/wps/`

3.1.4 IBM WebSphere MQ V6.0

IBM WebSphere MQ is an established and reliable message queuing middleware platform. A message queuing infrastructure built upon WebSphere MQ technology can provide an available, reliable, scalable, secure and maintainable transport for messages with exactly once delivery assurance.

The Message Queuing Interface (MQI) is the core API provided by WebSphere MQ. It is a procedural API suitable for applications developed within procedural programming languages. Procedural languages like C and COBOL most likely utilize MQI directly whereas object oriented languages like Java and C++ are supported with object oriented APIs built upon MQI.

WebSphere MQ also supports Java Message Service (JMS) and WebSphere message client API (XMS), a programming API that allows access from C, C++, and .NET applications.

WebSphere MQ provides features to assure security of access, authentication of identity and security and integrity of communication. The Object Authority Manager (OAM) is the default authorization service for command and object management. All actions performed by an application connected to a queue manager, are authenticated by the OAM.

WebSphere MQ provides high availability through workload balancing and failover capabilities. Administration of WebSphere MQ is typically done using control commands or the Eclipse-based WebSphere MQ Explorer administration tool (Windows® or Linux® only).

You can find more information about IBM WebSphere MQ at:

- WebSphere MQ home page:

`http://www.ibm.com/software/integration/wmq/`

3.1.5 IBM WebSphere Message Broker V6.0

WebSphere Message Broker enhances the flow and distribution of information by enabling the transformation and intelligent routing of messages without the need to change either the applications that are generating the messages or the applications that are consuming them.

Figure 3-6 shows a high level architectural view of WebSphere Message Broker.

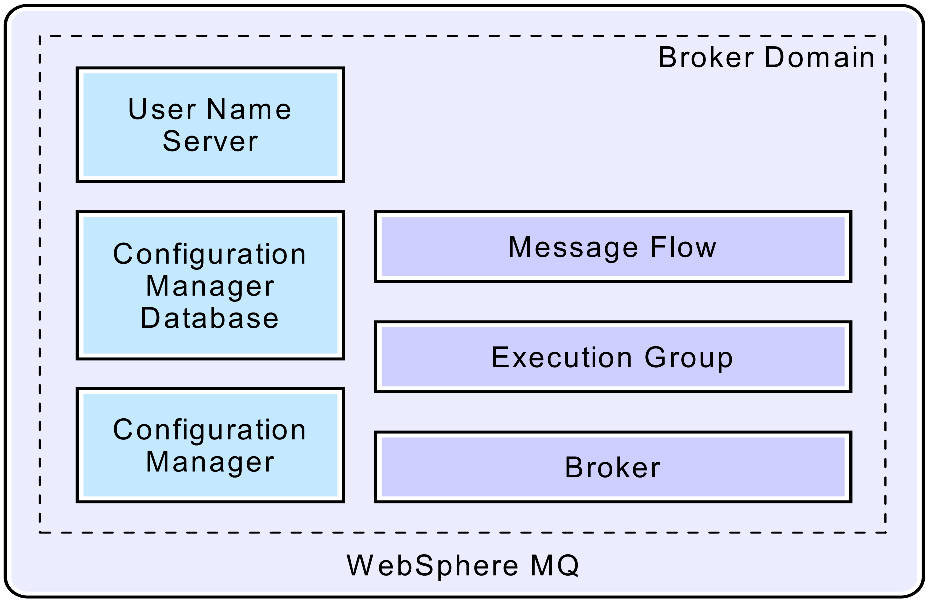


Figure 3-6 Architectural model of WebSphere Message Broker

The broker is a set of application processes that host and run message flows consisting of a graph of nodes that represent the processing needed for integrating applications. The broker also hosts message sets containing message models for predefined message formats.

When a message from a business application arrives at the broker, the broker processes the message before passing it on to one or more other business applications. The broker routes, transforms, and manipulates messages according to the logic that is defined in message flow applications. A broker uses WebSphere MQ as the transport mechanism both to communicate with the Configuration Manager, from which it receives configuration information, and to communicate with any other brokers to which it is associated. Each broker has a database in which it stores the information that it needs to process messages at run time.

Execution groups enable message flows within the broker to be grouped together. Each broker contains a default execution group. Additional execution groups can be created as long as they are given unique names within the broker. Each execution group is a separate operating system process and, therefore, the contents of an execution group remain separate from the contents of other execution groups within the same broker. This can be useful for isolating pieces of information for security because the message flows execute in separate address spaces or as unique processes. Message flow applications are deployed to a specific execution group. To enhance performance, the same message flows and message sets can be running in different execution groups.

The Configuration Manager is the interface between the Message Brokers Toolkit and the brokers in the broker domain. The Configuration Manager stores configuration details for the broker domain in an internal repository, providing a central store for resources in the broker domain. The Configuration Manager is responsible for deploying message flow applications to the brokers and delivering reports on the progress of the deployment and on the status of the broker. When the Message Brokers Toolkit connects to the Configuration Manager, the status of the brokers in the domain is derived from the configuration information stored in the Configuration Manager’s internal repository.

WebSphere Message Broker together with WebSphere MQ provide high availability features. This is quite important since WebSphere Message Broker acts as a hub and therefore needs to be eliminated as a single point of failure.

Load balancing and high availability can be achieved by providing multiple broker instances serving the same logical hub with each instance is mapped to its own WebSphere MQ queue manager. The different broker instances could reside on different machines.

WebSphere Message Broker provides the Message Broker Toolkit, a graphical environment for developing and deploying message flow applications.

You can find more information about IBM WebSphere Message Broker at:

- WebSphere Message Broker home page:

  `http://www.ibm.com/software/integration/wbimessagebroker/v6/`

3.1.6 IBM WebSphere Adapters V6.0

Not all applications provide a Web service or messaging interface. In these instances, adapters can be used to link applications to the ESB.

WebSphere Adapters are compliant to the Java 2 Platform, Enterprise Edition Connector Architecture version 1.5. They enable inbound and outbound connectivity between enterprise information systems and SCA based applications hosted by WebSphere Enterprise Service Bus or WebSphere Process Server. The WebSphere Adapters are deployed as part of an J2EE application, as an embedded resource adapter.

IBM also offers WebSphere Business Integration Adapters. They are not standard based and they reside outside of the application server. The server communicates with these adapters using a JMS transport layer.

3.2 ESB product positioning

The product you select to implement an ESB depends on the requirements of your solution. We have introduced two strategic products that provide ESB capabilities:

- WebSphere Enterprise Service Bus
- WebSphere Message Broker

Now, we give you a quick comparison of the two.

WebSphere Enterprise Service Bus is designed to provide the core functionality of an ESB for a predominantly Web services based environment. It is built on WebSphere Application Server, which provides the foundation for the transport layer. WebSphere Enterprise Service Bus adds a mediation layer based on the SCA programming model on top of this foundation to provide intelligent connectivity. If the customer has a lot of Web services in their environment, WebSphere Enterprise Service Bus is likely to be the better product to use.

WebSphere Message Broker provides a more advanced ESB solution with advanced integration capabilities such as universal connectivity and any-to-any transformation for data-centric deployments. It can handle services integration as well as integration with non-services applications. WebSphere MQ provides the transport backbone for messaging applications. Typically, customers who need a higher performance and throughput product in a message-centric environment would use WebSphere Message Broker.

Both products can also be used in combination. There are two main scenarios, where a combination can be used:

- Both ESB products are connected to provide a enterprise wide ESB combining the features to support all Web services technologies and integration for messaging applications. Figure 3-7 shows this setup.

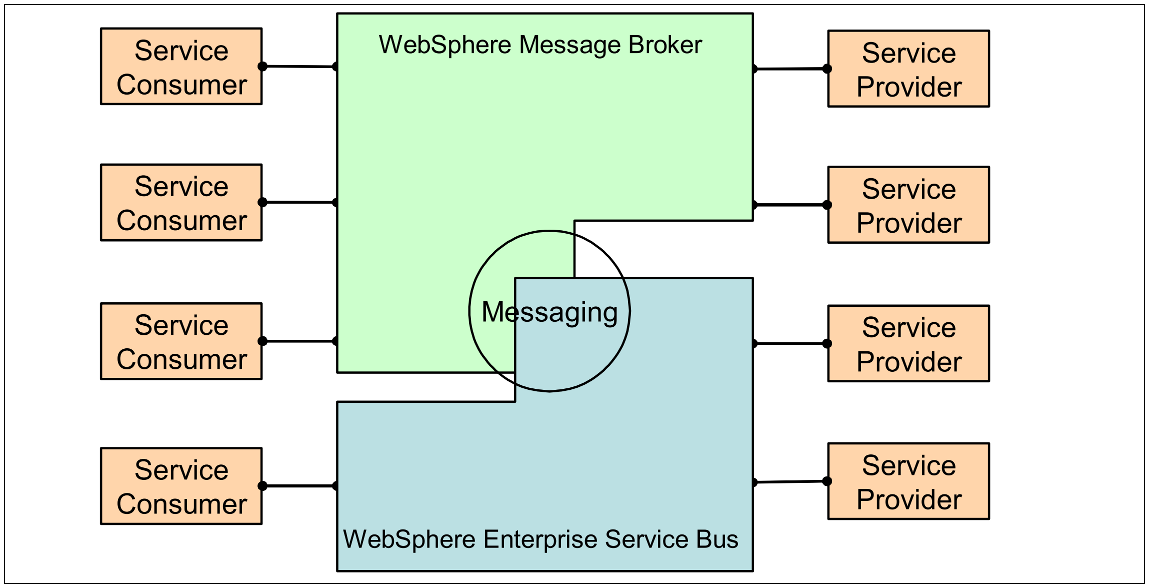


Figure 3-7 Combining WebSphere Message Broker and WebSphere Enterprise Service Bus

- WebSphere Message Broker acts as the central ESB, while WebSphere Enterprise Service Bus enables message processing to be efficiently deployed into branches, warehouses, stores, and so on. Having ESB functionally available locally adds flexibility, as branches can run independently from the central hub if connectivity is limited. Figure 3-8 shows this scenario.

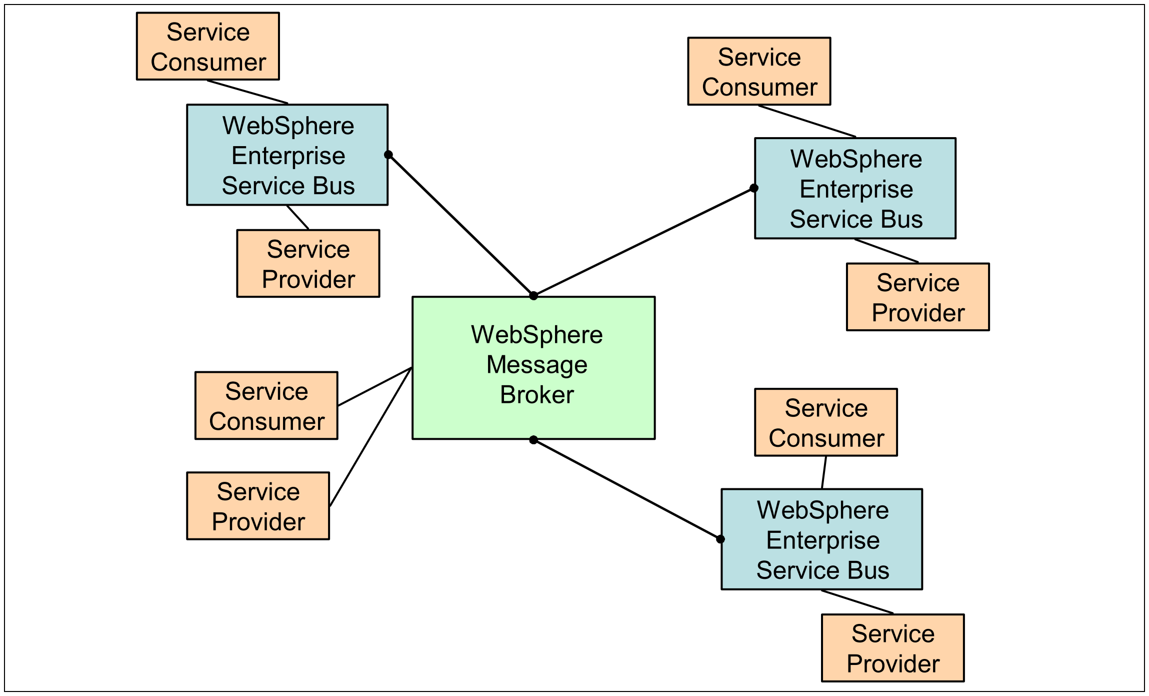


Figure 3-8 WebSphere Enterprise Service Bus complementing WebSphere Message Broker

3.2.1 Comparing WebSphere Enterprise Service Bus to WebSphere Message Broker

For a comparison of WebSphere Enterprise Service Bus and WebSphere Message Broker ESB capabilities see Table 3-1.

Table 3-1 WebSphere Enterprise Service Bus vs. WebSphere Message Broker

	WebSphere Enterprise Service Bus V6.0	**WebSphere Message Broker V6.0**
Connectivity	► TCP/IP, SSL, HTTP(S), IIOP ► JMS V1.1 (point-to-point, pub/sub) ► JMS/MQ (using MQLINK configuration)	► TCP/IP, SSL, HTTP(S) ► JMS V1.1 (point-to-point, pub/sub) ► Native WebSphere MQ ► Supports WebSphere MQ Transport, WebSphere MQ Everyplace® Transport, Multicast Transport, Real-time Transport, SCADA Transport, Web Services Transport, JMS Transport ► CICS, VSAM using SupportPacs ► Files using WebSphere Message Broker File Extender
Web services support	► SOAP/HTTP(S), SOAP/JMS, WSDL 1.1 ► Supports WS-I Basic Profile V1.1 ► UDDI V3.0 Service Registry ► WS-Security, WS-Atomic Transactions ► Client support: J2EE client, Message client for C/C++ and .NET, Web services client	► SOAP/HTTP(S), SOAP/JMS, WSDL 1.1 ► Supports WS-I Basic Profile V1.0 ► Client support: JMS client, Message client for C/C++ and .NET, Web services client, MQI client
Adapter support	► WebSphere Adapters and WebSphere Business Integration Adapters	► WebSphere Business Integration Adapters
Message logging	► Provides prebuilt mediation primitives for message logging	► Provides prebuilt message flow nodes for message logging

	WebSphere Enterprise Service Bus V6.0	WebSphere Message Broker V6.0
Message transformation	► Protocol transformation between HTTP, JMS, IIOP ► Custom transformation logic can be implemented in Java, XSLT ► Supports transformation of XML, SOAP, JMS message data format (many more if used with adapters)	► Protocol transformation between any protocols available as input or output nodes (HTTP, JMS, MQ, and more) ► Custom transformation logic can be implemented in Java, ESQL, or XSLT ► Supports transformation of self defined messages (XML), built-in predefined messages (SOAP, MIME, and more), and custom predefined messages (MRM)
Message routing	► Content and transport/protocol based routing ► Provides prebuilt mediation primitive for message routing, or custom build mediation using Java ► Supported through SCA	► Content and transport/protocol based routing ► Custom routing logic can be implemented in Java or ESQL
Data enrichment	► Built-in database lookup mediation primitive	► Built-in nodes for database access (ESQL, Java, graphical mapping)
Validation	► Validation of the input message against its schema by configuration of primitives	► Validation of input and output message against its schema definition.
Event-driven processing	► Supports event-driven processing by leverage adapters for capture and dissemination of business events	► Supports complex event processing (processing of events formed by several earlier ones)
Security	► HTTPS support ► Authentication and authorization as part of J2EE ► Support for WS-Security	► HTTPS support ► Authentication and authorization by the operating system environment

	WebSphere Enterprise Service Bus V6.0	**WebSphere Message Broker V6.0**
Quality of service	► Assured delivery support by service integration bus ► Transaction support provided by WebSphere Application Server ► Configurative within SCA module components	► Assured delivery support by WebSphere MQ ► Transaction support by WebSphere MQ (limited for JDBC connections) ► Configurative within node properties
Management	► High availability and scalability provided by WebSphere Application Server environment ► Built-in administration tools ► Import bindings can be modified using the administration console ► CEI support. Entry, exit and failure events can be activated on all SCA components within the mediation modules ► Common Base Event browser for viewing events from the CEI	► A high level of availability can be achieved using multiple brokers in combination with WebSphere MQ clustering ► Built-in administration tools

3.2.2 Summary

In conclusion, consider the following:

- WebSphere Enterprise Service Bus

  Building an ESB that is based entirely on WebSphere Enterprise Service Bus is an option when Web services support is critical and the service provider and consumer environment is predominantly built on open standards. WebSphere Enterprise Service Bus is most suitable for environments that are based on Web services standards and provides facilities to integrate services that are offered through enterprise application integration messaging and other sources. However, if integration with non-Web service standards-based services is a major requirement then WebSphere Enterprise Service Bus may not be the right choice.

- WebSphere Message Broker

  WebSphere Message Broker is suitable where advanced ESB functionality is required. WebSphere Message Broker is an option when Web services support is not critical and quality-of-service requirements demand the use of mature middleware. WebSphere Message Broker can support the majority of the ESB capabilities that WebSphere Enterprise Service Bus does but is not limited to open standards. However, in comparison with WebSphere Enterprise Service Bus, it lacks the sophistication of Web services support that might be required in an ESB implementation which makes extensive use of these standards.

3.2.3 IBM SOA Foundation and Patterns for e-business

The IBM SOA Foundation is a reference architecture used to build new, or extend existing, applications and business processes. The IBM SOA Foundation includes an integration architecture, best practices, patterns, and SOA scenarios to help simplify the packaging and use of IBM open standards-based software.

The IBM Patterns for e-business and are a group of proven, reusable assets that can be used to increase the speed of developing and deploying On Demand business applications. The Patterns for e-business approach enables architects to implement successful e-business solutions through the reuse of components and solution elements from proven successful experiences.

Using a combined SOA process identified by IBM, both the SOA Foundation and Patterns for e-business can be used to help select the appropriate architecture and products to build ESB solutions. WebSphere Enterprise Service Bus and WebSphere Message Broker both fit into the Service Connectivity SOA scenario.

Please consult the following resources for more information:

- IBM SOA Foundation and the Service Connectivity SOA scenario
  - *Patterns: SOA Foundation - Service Creation Scenario, SG24-7240*
  - *Patterns: SOA Foundation - Service Connectivity Scenario*, SG24-7228
- IBM Patterns for e-business
  - `http://www.ibm.com/developerworks/patterns`

3.3 Development environment

Some products provide specific development environments. For example WebSphere Message Broker provides a Eclipse-based graphical administration and development tools.

All WebSphere Application Server-based product provide built-in administration tools but for developers there is a set of development products, packaged for different user roles.

3.3.1 User roles

A user role is an abstract collection of skills, needs, and responsibilities. User roles do not necessarily map to a specific person. Individuals will assume a user role, determined by the activity they are involved in. Dependent of the phase of the process a person is working on, a single person can assume many roles.

For the life cycle of an SOA the ESB user roles are shown in Figure 3-9.

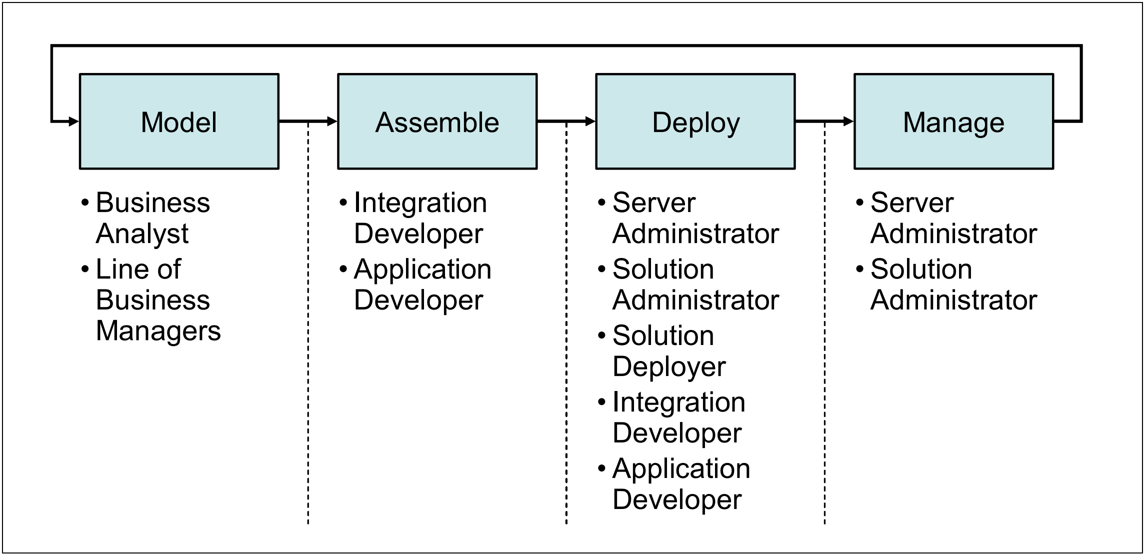


Figure 3-9 SOA life cycle role mapping

The line of business manager and the business analyst are responsible for modeling the solution. While the line of business manager is concentrating on the strategy, the business analyst models the solution that supports the strategy. The integration developer and the application developer assemble the solution and work together with the administrators and deployers to deploy the solution. Then the administrators are responsible for managing the solution.

Here we will concentrate on the two roles involved in the assembly phase:

Integration developer

The integration developer focuses on the SOA solutions. This role needs some programming experience, but expects the tools to simplify and abstract advanced implementation details. This role develops SCA-based applications, such as mediation modules, using existing components.

The integration developer uses WebSphere Integration Developer for all the tasks in this role.

Application developer

The application developer focuses on development of components and services used by the SOA solution. This includes developing Web services, resource adapters, and custom mediations.

The application developer uses Rational Application Developer for development.

3.3.2 Rational Application Developer V6.0

For J2EE application developers IBM offers Rational Application Developer for WebSphere software. It is based on the Rational Software Development Platform. It includes all features of Rational Web Developer included in WebSphere Application Server - Express and additionally provides EJB and Web service development tools. So all components to be deployed to WebSphere Application Server and WebSphere Application Server Network Deployment can be developed using Rational Application Developer.

For developers using modeling language technologies for creating Java and service-oriented applications, Rational Software Architect adds modeling features to the functions available in Rational Application Developer.

3.3.3 WebSphere Integration Developer V6.0

In WebSphere Integration Developer the integration developer can create all components to be deployed to WebSphere Enterprise Service Bus and WebSphere Process Server.

WebSphere Integration Developer provides editors to work on SCA modules, interfaces, data types, and all kinds of SCA components. It also supports integrated debugging for modules and components, a unit test environment, and a end-to-end framework.

WebSphere Integration Developer supports the top-down and the bottom-up development approach. It is designed to hide the complexity of WSDL, XSD, XPath and XSLT. So integration developers do not need to have deep skills in these technologies to create a solution. Wherever the supplied mediation primitives do not meet the needs, custom mediation primitives can be created visually or by writing Java code. These more advanced task are typically performed by more advanced integration developers or application developers.

The relationship between WebSphere Integration Developer and the other Rational® Software Development Platform products is shown in Figure 3-10 on page 53.

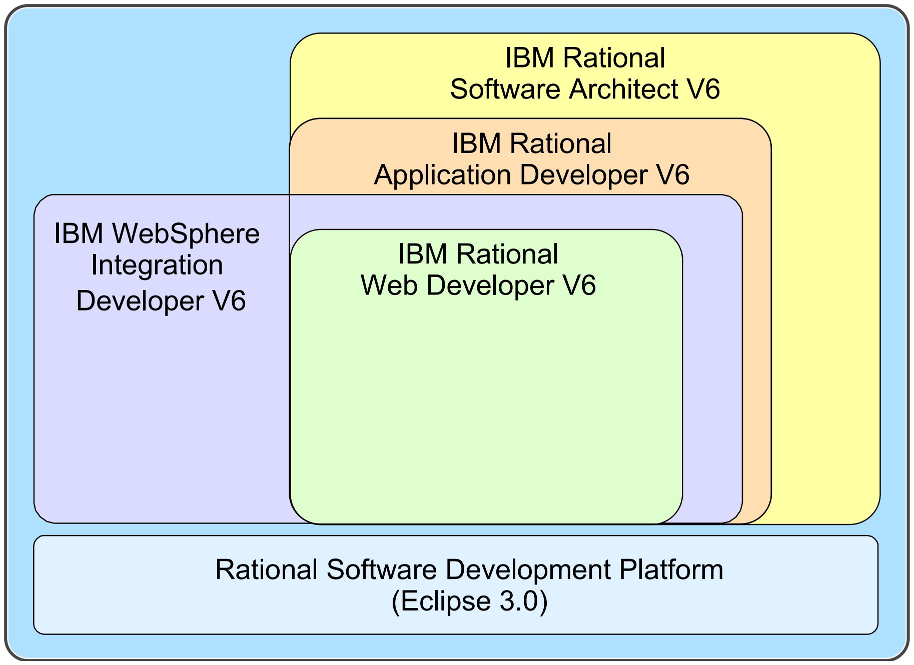


Figure 3-10 Development tools overview

WebSphere Integration Developer includes most functions provided by Rational Application Developer, but not all of them. For example the Crystal Report tools and WebSphere Portal development tools are not included in WebSphere Integration Developer, but in Rational Application Developer. Because both products are based on the Rational Software Development Platform, they can be combined in a single development environment.

If a single user assumes both the integration developer role and the application developer role, he can either use the J2EE development tools in WebSphere Integration Developer, or use both development products, WebSphere Integration Developer and Rational Application Developer.

Part 2

Configuration and usage

4

Setting up the development environment

This chapter discusses the WebSphere Enterprise Service Bus development environment, WebSphere Integration Developer V6.0.1.

This chapter discusses the following topics:

- Overview of the development environment
- Planning for multiple development environments
- Installing the development tool
- Team development
- Integration test considerations
- Troubleshooting installation issues

4.1 Overview of development environment

WebSphere Integration Developer V6.0.1 is the development environment for WebSphere Enterprise Service Bus V6.0 and WebSphere Process Server V6.0.1. It provides an environment for building and testing integrated applications based on a services-oriented architecture.

The application development in WebSphere Integration Developer V6.0.1 is based on Service Component Architecture. Besides developing Service Component Architecture components and modules, WebSphere Integration Developer is also used to assemble mediations, components using mediation primitives, and to create mediation modules. WebSphere Integration Developer includes an integrated unit test environment for WebSphere Process Server and WebSphere Enterprise Service Bus, allowing developers to deploy their modules to the integrated test server and perform unit testing using the integration test client.

4.1.1 Hardware and software requirements

The WebSphere Integration Developer product Web site provides the list of minimum hardware and software required. This information can be found at:

`http://www.ibm.com/software/integration/wid/sysreqs`

4.1.2 Consider your current environment

WebSphere Integration Developer is based on Rational Software Development Platform which is shared by several IBM products. The list of IBM products that are based on Rational Software Development Platform can be found at:

`http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.install.doc/topics/cinsdp.html`

Note that Rational Software Development Platform is installed only once when the first product is installed. Subsequent products use the common user interface and add product specific functionality that is provided by the plug-ins.

If your current environment has any existing Rational Software Development Platform products installed, the installation of WebSphere Integration Developer will integrate into the existing Rational Software Development Platform.

WebSphere Integration Developer V6.0.1 is based on Rational Software Development Platform V6.0.1 and is only compatible with other products that are based on this level. If you have a product that uses an earlier version of Rational Software Development Platform, you will be required to upgrade that product or uninstall it, so WebSphere Integration Developer V6.0.1 can be installed.

WebSphere Integration Developer can co-exist with WebSphere Studio Application Developer Integration Edition V5.1.1 and previous releases. WebSphere Integration Developer V6.0.1 cannot coexist with WebSphere Integration Developer V6.0

4.2 Planning for multiple development environments

When setting up multiple WebSphere Integration Developer environments, ensure that each developer workstation has sufficient disk space for the installation. WebSphere Integration Developer installation requires about 5.5 GB of disk space for the install directory as well as 1 GB of disk space for temporary files.

While it might be convenient to install on multiple workstations from a network installation image, due to the size of the install image, it is recommended that the install image be copied to the local drive of the workstation before starting the WebSphere Integration Developer installation. This will ensure the installation completes successfully, and does not get affected by network issues.

Information on creating a network installation image can be found at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/com.ibm.wbit.help.install.doc/topics/tinstnet.html

When installing on Windows operating systems, WebSphere Integration Developer by default gets installed in C:\Program Files\IBM\WebSphere\ID\6.0. Its recommended the installation directory be changed to a short fully qualified directory name to avoid problems with path length exceeding 256 characters on Windows. When setting up multiple development environment, it helps to use a standard directory name for install, for ease of management.

4.2.1 Silent installation

Instead of using the installation wizard, it is possible to install WebSphere Integration Developer non-interactively, using a response file. Silent installation could be very useful when setting up multiple development environments.

Note that installing WebSphere Integration Developer silently using the provided response file will install the default features. It will install the Integrated Development Environment, and will not install the Integrated Test Environment.

Prior to performing a silent install, you must:

1. Configure the response file used for the installation.
2. Check the workstation for any issues related to co-existence or upgrade

3. Ensure the workstation has sufficient disk space for the install.

The sample response file is called responsefile.txt and is located in the \disk1\util directory. Based on your development environment requirements, it is recommended you make a copy of the response file and modify it accordingly.

If your developers intend to work mainly on developing mediation flow components, and test only applications created by the mediation flow editor, you can modify the response file and select WebSphere Enterprise Service Bus server only as the unit test server for install and do not have to install WebSphere Process Server.

Information on configuring the response file and starting a silent install can be found at:

```
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/c
om.ibm.wbit.help.install.doc/topics/thushadditional.html
```

4.2.2 Roles

WebSphere Integration Developer supports several user roles for different activities and functionality. WebSphere Integration Developer hides product features based on the role selected. A specific user role can be enabled or disabled from the Welcome page on WebSphere Integration Developer. The hidden features can be enabled when first accessed or through **Window -> Preferences -> Workbench -> Capabilities**. Integration Developer is primary role used for developing mediation modules and build SOA solutions.

For more information on user roles, see 3.3.1, “User roles” on page 51.

4.3 Installing the development environment

This section describes how to install WebSphere Integration Developer, and essential product updates. It assumes a Windows environment.

This section contains the following sections:

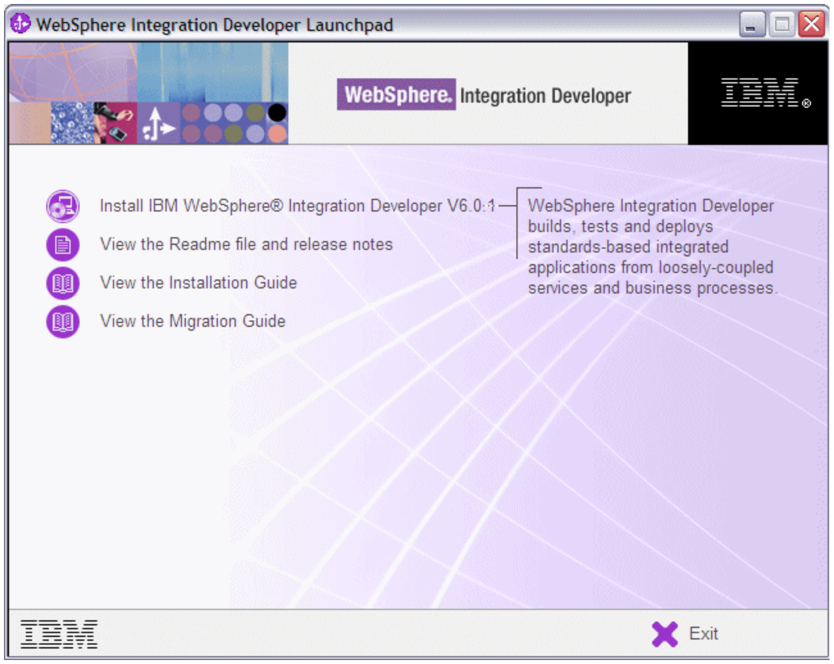
- Installing WebSphere Integration Developer
- Using Rational Product Updater
- Starting WebSphere Integration Developer

4.3.1 Installing WebSphere Integration Developer

This section walks you through all the steps of installing a default configuration of WebSphere Integration Developer V6.0.1. Before beginning the installation, ensure that you are logged in as a user with administrative privileges, then perform the following:

Note: Installation time will vary, and may extend up to two hours if both WebSphere Enterprise Service Bus and WebSphere Process Server test environments are installed.

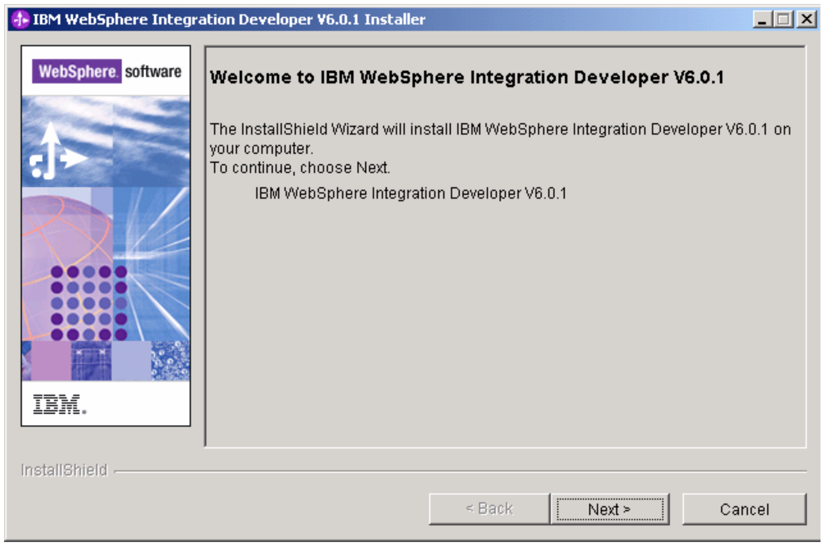
1. Start the **launchpad.exe** from disk1 of the installation CDs. This opens the launchpad shown in Figure 4-1. It is recommended you review the readme file and release notes as they contain late breaking information.


WebSphere Integration Developer Launchpad
WebSphere. Integration Developer
IBM
Install IBM WebSphere® Integration Developer V6.0.1
View the Readme file and release notes
View the Installation Guide
View the Migration Guide
Exit

WebSphere Integration Developer builds, tests and deploys standards-based integrated applications from loosely-coupled services and business processes.

Figure 4-1 Install wizard - Launchpad

2. Click **Install IBM WebSphere Integration Developer V6.0.1** to start the installation.
3. The IBM WebSphere Integration Developer V6.0.1 Installer will launch (Figure 4-2). Click **Next**.



Welcome to IBM WebSphere Integration Developer V6.0.1

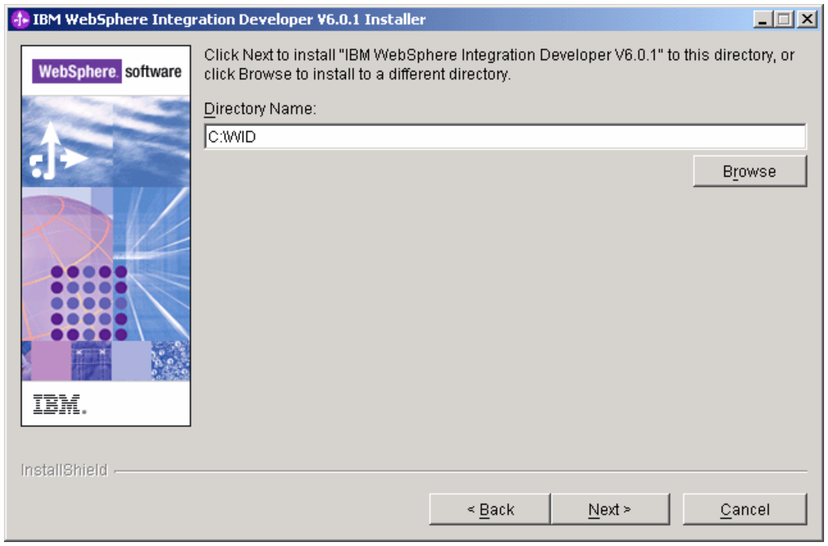
The InstallShield Wizard will install IBM WebSphere Integration Developer V6.0.1 on your computer.
To continue, choose Next.

IBM WebSphere Integration Developer V6.0.1

Figure 4-2 Install wizard - Welcome screen

4. The Software License Agreement will be displayed. Accept the agreement, then click **Next**.
5. Specify an installation directory as shown in Figure 4-3, then click **Next**.

> **Note:** Its recommended the installation directory be changed to a short fully qualified directory name to avoid problems with path length exceeding 256 characters on Windows.


IBM WebSphere Integration Developer V6.0.1 Installer

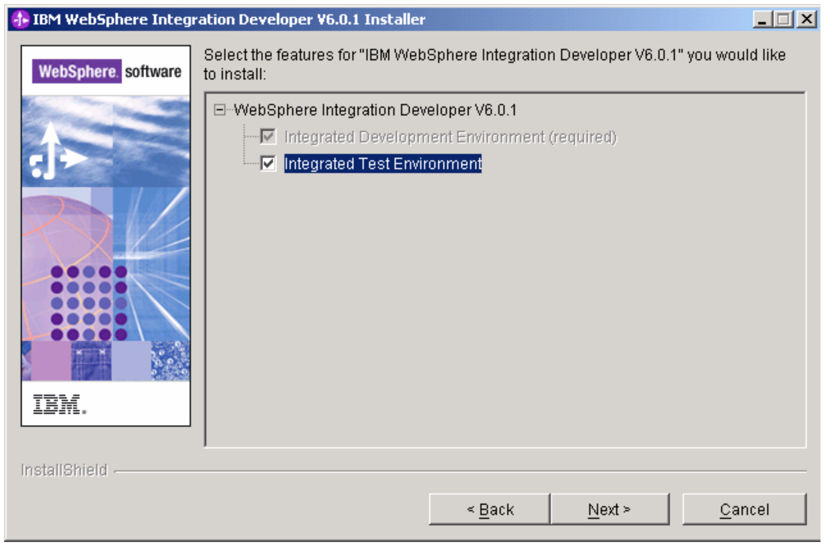
Click Next to install "IBM WebSphere Integration Developer V6.0.1" to this directory, or click Browse to install to a different directory.

Directory Name:

C:WID

Figure 4-3 Install wizard - Install directory

6. The installation of WebSphere Integration Developer is divided into two parts. The first part is to install the Integrated Development Environment. This part is required. The second part is the Integrated Test Environment which allows you to deploy, run, and test artifacts you have built in the Integrated Development Environment. This part is optional but strongly recommended if you want to unit test your artifacts. Click **Integrated Test Environment** and click **Next** (Figure 4-4).

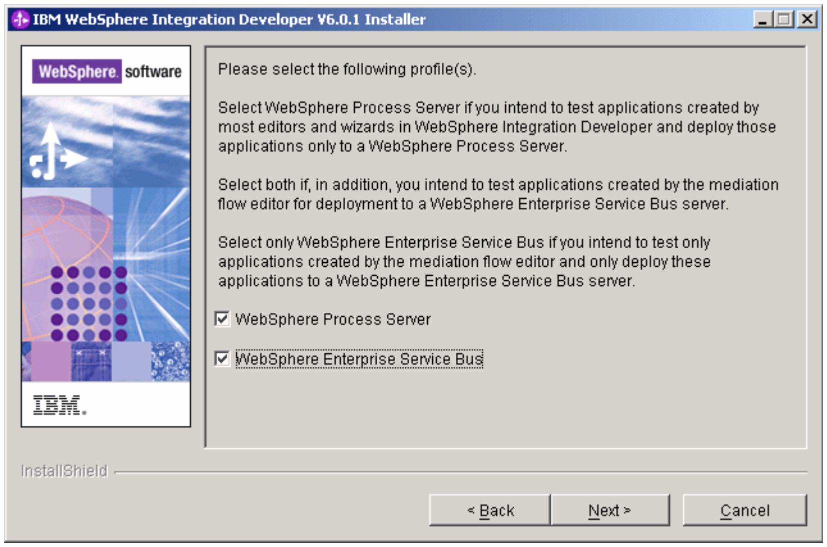


Select the features for "IBM WebSphere Integration Developer V6.0.1" you would like to install:

- WebSphere Integration Developer V6.0.1
  - [✓] Integrated Development Environment (required)
  - [✓] Integrated Test Environment

Figure 4-4 Install wizard - Integrated Test Environment

7. The Integrated Test Environment of WebSphere Integration Developer offers two server types: WebSphere Process Server and WebSphere Enterprise Service Bus. We recommend installing both. Select **WebSphere Enterprise Service Bus** in addition to WebSphere Process Server and click **Next** (Figure 4-5).



Please select the following profile(s).

Select WebSphere Process Server if you intend to test applications created by most editors and wizards in WebSphere Integration Developer and deploy those applications only to a WebSphere Process Server.

Select both if, in addition, you intend to test applications created by the mediation flow editor for deployment to a WebSphere Enterprise Service Bus server.

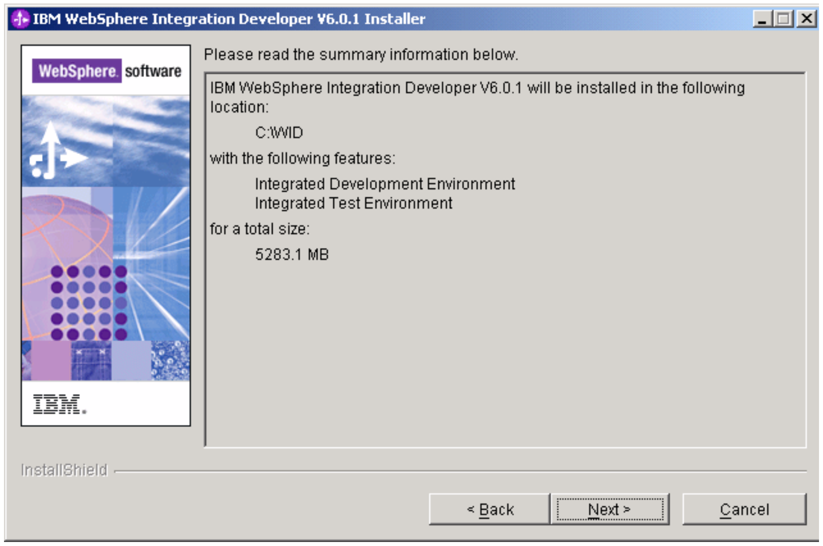
Select only WebSphere Enterprise Service Bus if you intend to test only applications created by the mediation flow editor and only deploy these applications to a WebSphere Enterprise Service Bus server.

[✓] WebSphere Process Server

[✓] WebSphere Enterprise Service Bus

Figure 4-5 Install wizard - Select Integration Test Environment

8. The next screen shows a summary of installation and shows the disk space required depending on the options selected (Figure 4-6). Click **Next** to start the installation.



Please read the summary information below.

IBM WebSphere Integration Developer V6.0.1 will be installed in the following location:

C:WID

with the following features:

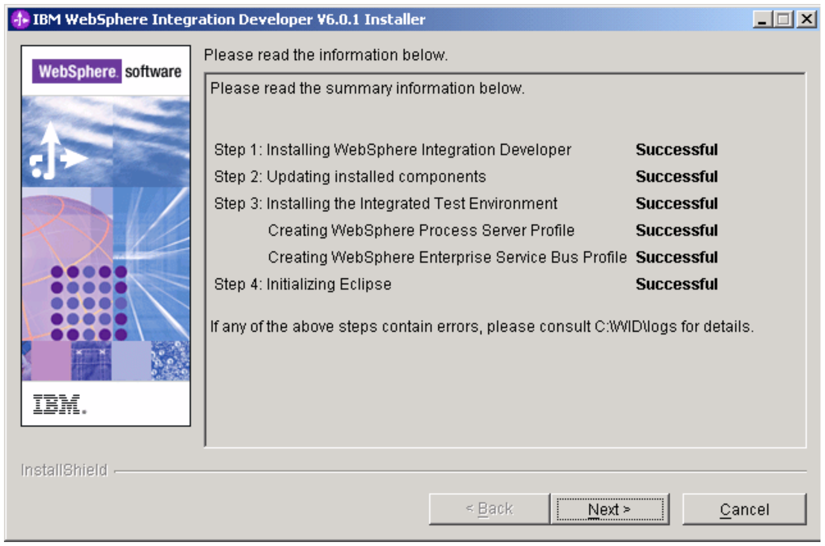
Integrated Development Environment
Integrated Test Environment

for a total size:

5283.1 MB

Figure 4-6 Install wizard - Summary and disk space required

9. At the end of the installation, a summary is shown along with the status for each of the component as shown in Figure 4-7. Click **Next**.



Please read the information below.

Please read the summary information below.

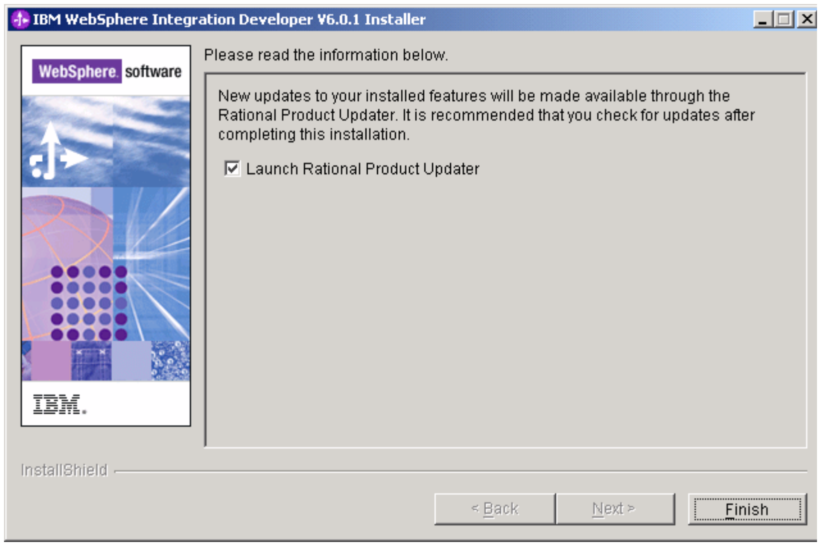
Step 1: Installing WebSphere Integration Developer	**Successful**
Step 2: Updating installed components	**Successful**
Step 3: Installing the Integrated Test Environment	**Successful**
Creating WebSphere Process Server Profile	**Successful**
Creating WebSphere Enterprise Service Bus Profile	**Successful**
Step 4: Initializing Eclipse	**Successful**

If any of the above steps contain errors, please consult C:\WID\logs for details.

Figure 4-7 Install wizard - Complete install summary

10. You will be asked if you wish to view the readme file containing late-breaking information. Click **Next**, then **Next** again.

11. Finally you will be asked if you wish to launch the Rational Product Updater. You can use this tool to apply interim fixes and product updates. 4.3.2, “Using Rational Product Updater” on page 68 provides more information on this tool. We recommend you select **Launch Rational Product Updater** (Figure 4-8). Click **Finish**.



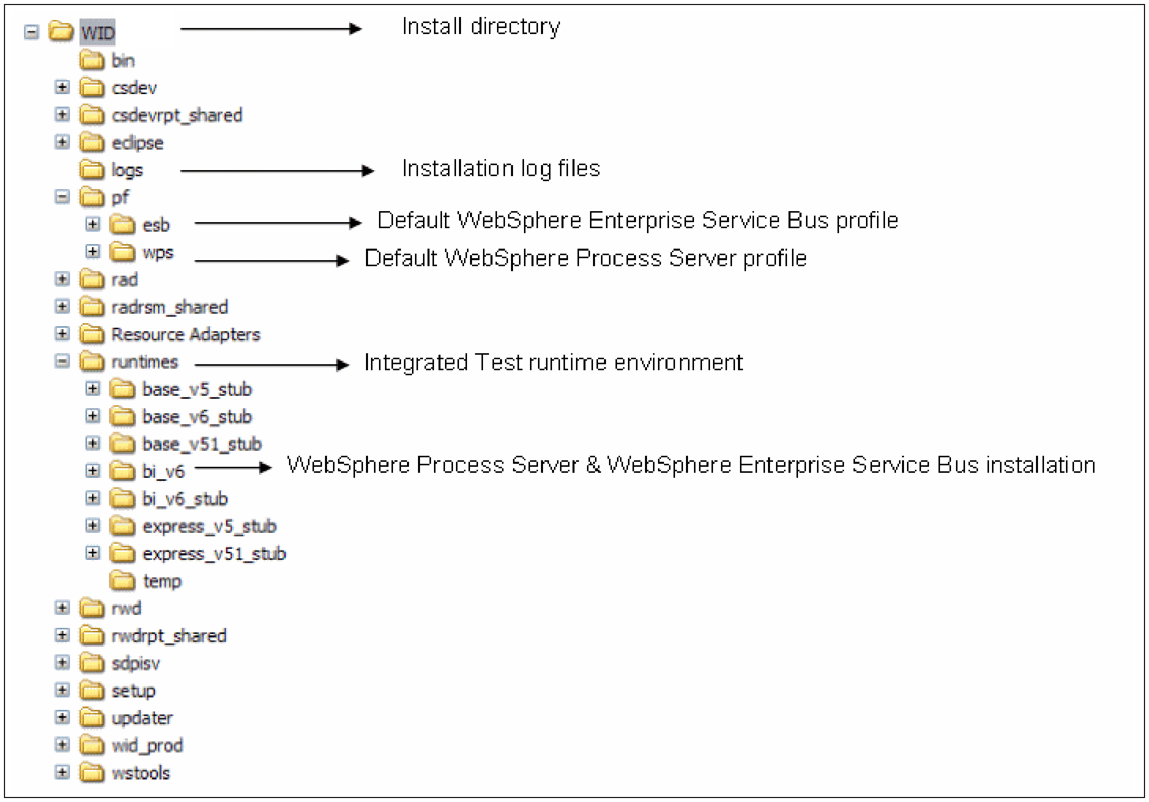
Please read the information below.

New updates to your installed features will be made available through the Rational Product Updater. It is recommended that you check for updates after completing this installation.

[✓] Launch Rational Product Updater

Figure 4-8 Install wizard - Launch product updater

12. The directory structure from a WebSphere Integration Developer installation is as shown in Figure 4-9 on page 68.



```
WID                 → Install directory
  bin
  csdev
  csdevrpt_shared
  eclipse
  logs              → Installation log files
  pf
    esb             → Default WebSphere Enterprise Service Bus profile
    wps             → Default WebSphere Process Server profile
  rad
  radrsm_shared
  Resource Adapters
  runtimes          → Integrated Test runtime environment
    base_v5_stub
    base_v6_stub
    base_v51_stub
    bi_v6           → WebSphere Process Server & WebSphere Enterprise Service Bus installation
    bi_v6_stub
    express_v5_stub
    express_v51_stub
    temp
  rwd
  rwdrpt_shared
  sdpisv
  setup
  updater
  wid_prod
  wstools
```

Figure 4-9 Directory structure of WebSphere Integration Developer installation

Note: You can create stand-alone server profiles using the **`esbpcatWindows.exe`** utility, located in:

```
<WID_INSTALL_DIR>\runtimes\bi_v6\bin\ProfileCreator_wbi
```

4.3.2 Using Rational Product Updater

Rational Product Updater is the tool provided to install maintenance updates for WebSphere Integration Developer as well as other products based on Rational Software Development Platform. This tool accesses the update server on the internet, locates and installs product updates as well as optional new features

After completing a successful install, it is strongly recommended to check for product updates, so you can install fixes, and prevent encountering known problems.

Rational Product Updater can be launched from the install wizard, or from **Start -> Programs -> IBM WebSphere -> Integration Developer V6.0.1 -> Rational Product Updater**.

It is possible to change the update site preference within Rational Product Updater, so updates can be installed from a local or a network drive rather than from the internet update server. You will need to download the updates to a local driver and change the update site preference.

Recommended updates for WebSphere Integration Developer can be downloaded from:

```
http://www-1.ibm.com/support/docview.wss?rs=2308&uid=swg27006685
```

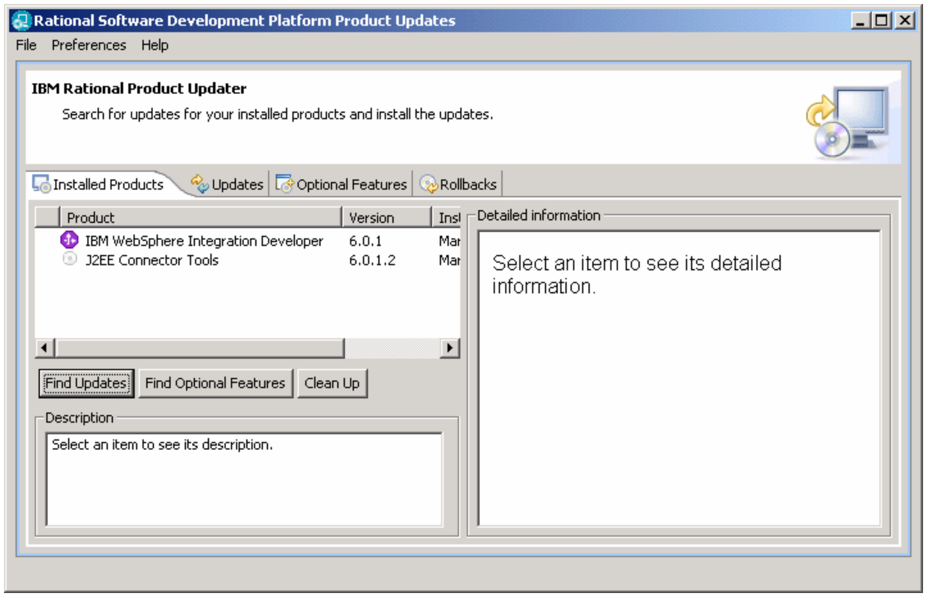
Further information on changing the update site preference can be found at:

```
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp?topic=/c
om.ibm.orca.updater.doc/topics/tupdatesites.html
```

While you can modify the update policy as required, you can also use the update policy that is shipped with each fixpack. This can be found in the `WID` directory after unzipping the fixpack and the filename is policy_601_interim_fixnnn.xml where nnn is the fix number.

After completing WebSphere Integration Developer install, we installed updates using Rational Product Updater. We have downloaded the updates and used the update policy shipped with the fix. Below is each step in installing an update using Rational Product Updater.

1. If the Rational Product Updater s not running, launch it from **Start -> Programs -> IBM WebSphere -> Integration Developer V6.0.1 -> Rational Product Updater**. This opens the Rational Software Development Platform Product Updates screen as shown in Figure 4-10.



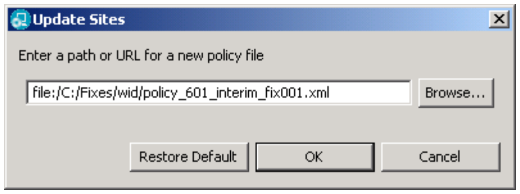
Product	Version	Inst
IBM WebSphere Integration Developer	6.0.1	Mar
J2EE Connector Tools	6.0.1.2	Mar

Select an item to see its detailed information.

Select an item to see its description.

Figure 4-10 Rational Product Updater

2. If you wish to provide a local update policy file instead of downloading one using the default internet update server, perform the following:
   a. Click on **Preferences -> Update Sites**.
   b. Use Browse to locate your policy name as shown in Figure 4-11.
   c. Click **OK**.



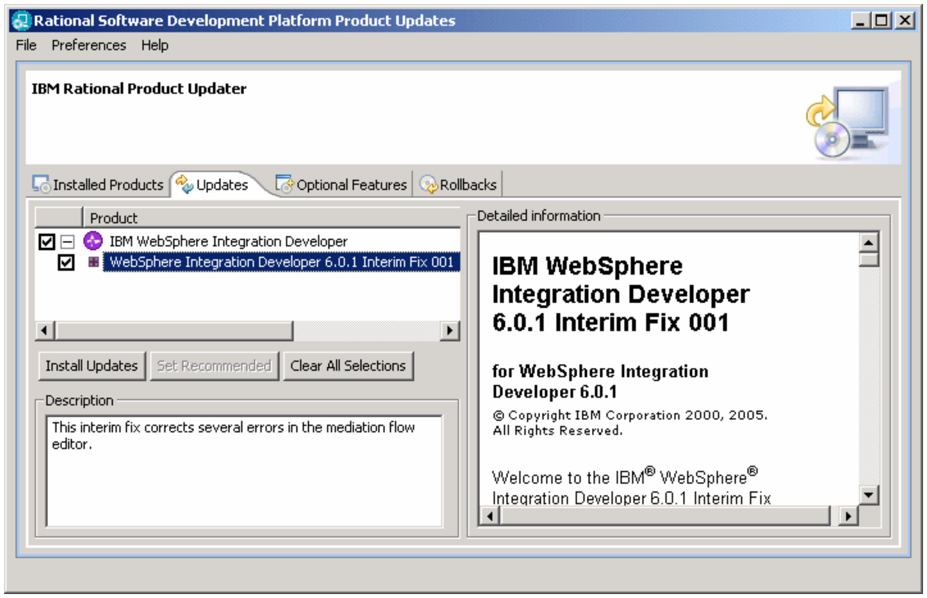
Enter a path or URL for a new policy file

file:/C:/Fixes/wid/policy_601_interim_fix001.xml

Figure 4-11 Provide your update policy

3. Under the Installed Products tab, highlight **IBM WebSphere Integration Developer** and click **Find Updates**.
4. Once the updates have been located (either locally or using the internet) the Product Updater will switch to the Updates view (Figure 4-12 on page 71).

   Highlighting each update shown in the update window will provide a brief description of that update in the Description box, and detailed information about that update in the Detailed information box. You can select or deselect an update to install using the check box. After completing selections, click on the **Install Updates** button to start the install.


Rational Software Development Platform Product Updates
File Preferences Help
IBM Rational Product Updater
Installed Products
Updates
Optional Features
Rollbacks
Product
IBM WebSphere Integration Developer
WebSphere Integration Developer 6.0.1 Interim Fix 001
Install Updates
Set Recommended
Clear All Selections
Description
Detailed information

This interim fix corrects several errors in the mediation flow editor.

IBM WebSphere Integration Developer 6.0.1 Interim Fix 001

for WebSphere Integration Developer 6.0.1

© Copyright IBM Corporation 2000, 2005.
All Rights Reserved.

Welcome to the IBM® WebSphere® Integration Developer 6.0.1 Interim Fix

Figure 4-12 Selecting the updates to install

5. After installation is complete, the installed updates will show on the **Installed Products** pane as shown in Figure 4-13.

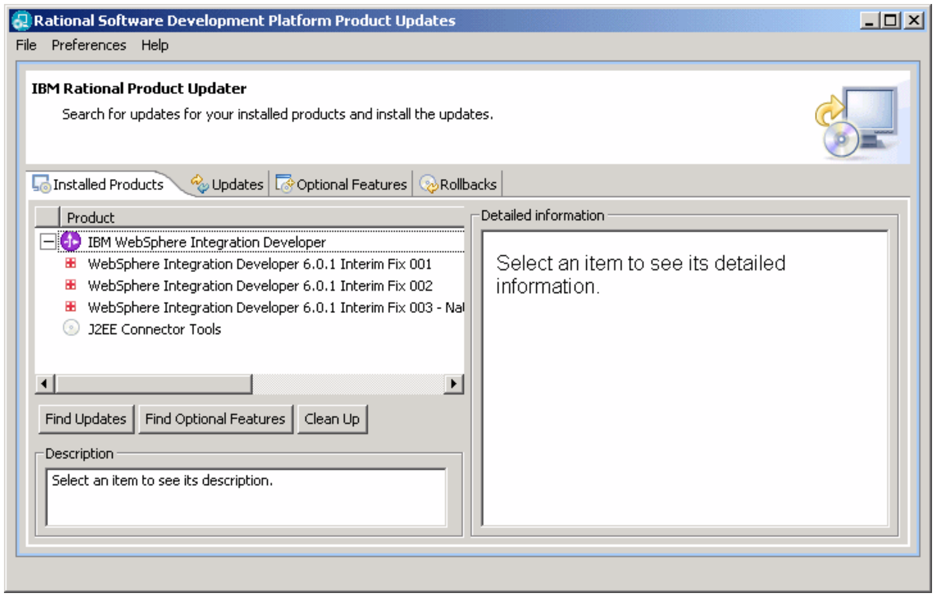

Rational Software Development Platform Product Updates
File Preferences Help
IBM Rational Product Updater
Search for updates for your installed products and install the updates.
Installed Products
Updates
Optional Features
Rollbacks
Product
IBM WebSphere Integration Developer
WebSphere Integration Developer 6.0.1 Interim Fix 001
WebSphere Integration Developer 6.0.1 Interim Fix 002
WebSphere Integration Developer 6.0.1 Interim Fix 003 - Nal
J2EE Connector Tools
Find Updates
Find Optional Features
Clean Up
Description
Select an item to see its description.
Detailed information
Select an item to see its detailed information.

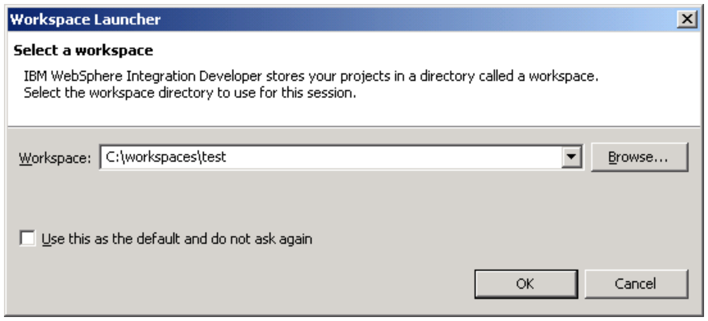
Figure 4-13 Install updates

4.3.3 Starting WebSphere Integration Developer

To start WebSphere Integration Developer, perform the following:

1. Click **Start -> Programs -> IBM WebSphere -> Integration Developer V6.0.1 -> WebSphere Integration Developer V6.0.1**.
2. This will launch the Workspace Launcher window (Figure 4-14 on page 73). Enter a path where a workspace should be created and click **OK**.

Note: We recommend using short path names for the workspace directory, to minimize the chances of exceeding the 256 character path length limit on Windows operating systems.


Workspace Launcher

Select a workspace

IBM WebSphere Integration Developer stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\workspaces\test

Figure 4-14 Workspace Launcher

3. WebSphere Integration Developer will start and open the Welcome page (Figure 4-15). Depending on the development activity, a specific role, or roles can be selected on this Welcome page. Roles can enable or disable using the button at bottom right corner on this page.

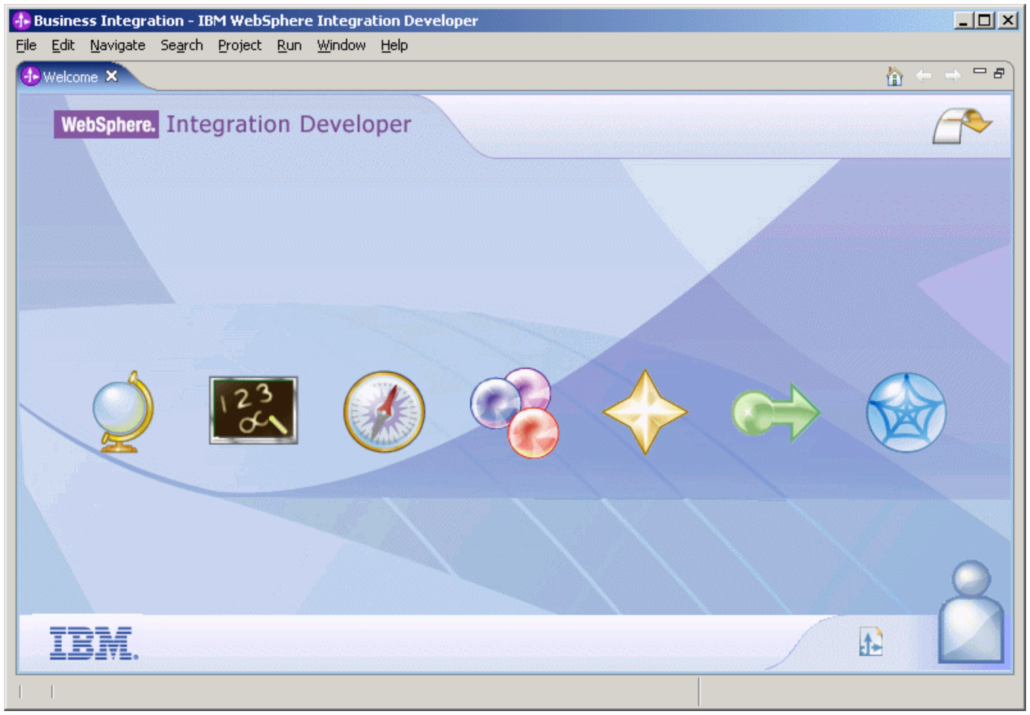

Business Integration - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Welcome
WebSphere. Integration Developer
IBM.

Figure 4-15 WebSphere Integration Developer welcome page

4.4 Team development

A typical development environment could involve multiple developers working collectively on a project. WebSphere Integration Developer provides the ability to share common artifacts across modules through a library project.

Besides the ability to share common artifacts using a library project, WebSphere Integration Developer V6.0.1 provides team development functionality from the Business Integration perspective and has integrated support for IBM Rational ClearCase® and CVS source code management systems. All Rational Software Development Platform based products include a CVS Repository Exploring perspective and a Team Synchronizing perspective. In addition, WebSphere Integration Developer's Business Integration perspective directly supports team development functionality.

The Business Integration perspective in WebSphere Integration Developer V6.0.1 supports the following operations for team development support.

- Share a project
- Synchronize changes between the local application and source code repository
- Commit local changes to source code repository
- Update local workspace artifacts with the version on source code repository

Note: This section assumes the CVS source code management system as the source code repository.

You must configure a connection to the CVS server using CVS Repository Exploring perspective. For information on CVS server configuration and implementation, and how to connect a CVS client to it, see Chapter 26 of the redbook *Rational Application Developer V6 Programming Guide*, SG24-6449.

The primary artifact that will be managed in a team development environment is a module. While WebSphere Integration Developer generates several staging projects when each application project is built, these generated projects are marked as derived. By default, CVS will not commit any files or folders from the generated staging projects marked as derived. The staging projects will get regenerated once a user extracts the module from CVS, as WebSphere Integration Developer by default has the **Build Automatically** flag checked.

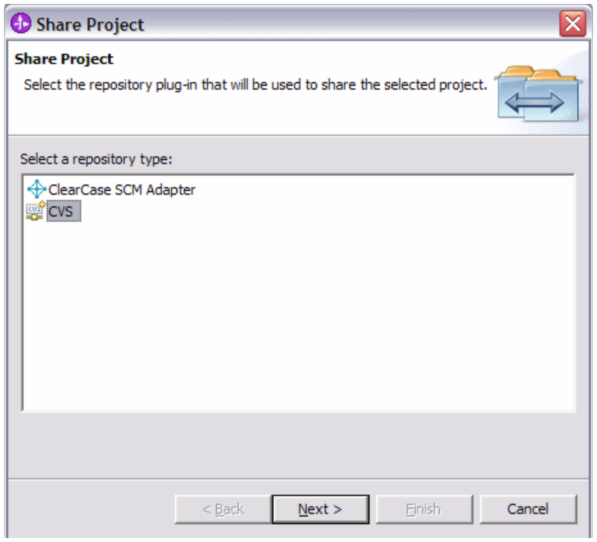
Note: If the default of **Build Automatically** is modified, you must build the project by clicking on **Project -> Build Project**, after check out from CVS.

From more information on packaging in WebSphere Integration Developer refer to:

`http://www-128.ibm.com/developerworks/websphere/library/techarticles/0512_phan2/0512_phan2.html`

Developers can share the project by checking it into a CVS repository. This can be done from Business Integration perspective using the following instructions.

1. Right click a mediation module and select **Team -> Share Project**.
2. The next screen requires you to select the repository plug-in that will be used for source code management system. Select **CVS** and click **Next** (Figure 4-16).


Share Project

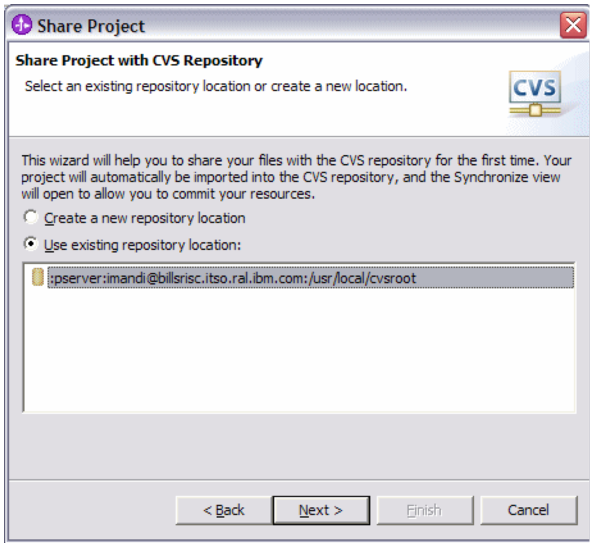
Select the repository plug-in that will be used to share the selected project.

Select a repository type:

- ClearCase SCM Adapter
- CVS

Figure 4-16 Select the repository plug-in

3. You can select an existing repository location that has been setup, or create a new repository location as shown in Figure 4-17 on page 76. Click **Next**.



Share Project

Share Project with CVS Repository

Select an existing repository location or create a new location.

This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Synchronize view will open to allow you to commit your resources.

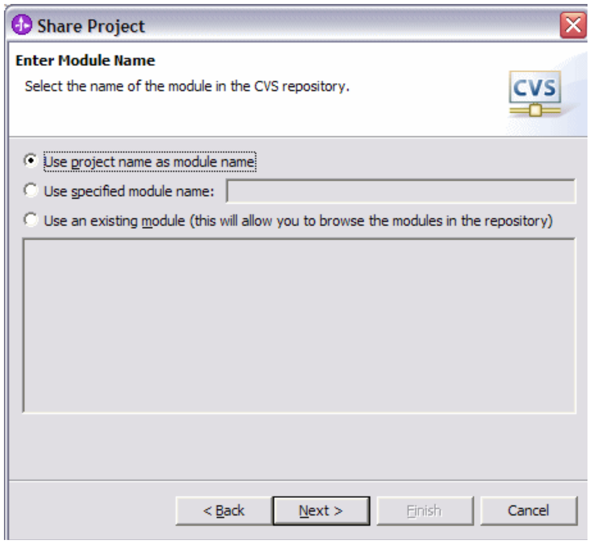
( ) Create a new repository location

(●) Use existing repository location:

:pserver:imandi@billsrisc.itso.ral.ibm.com:/usr/local/cvsroot

Figure 4-17 Select repository location

4. Provide the module name that this project will be known as in the CVS repository. Its important you check the radio button **Use project name as module name** as shows in Figure 4-18. In WebSphere Integration Developer the module file depends on the module name, and any changes in the module name will result in naming errors.



Share Project

Enter Module Name

Select the name of the module in the CVS repository.

(●) Use project name as module name

( ) Use specified module name:

( ) Use an existing module (this will allow you to browse the modules in the repository)

Figure 4-18 Module name

5. The next panel shows the all the files for the module that will be checked into CVS repository as shown in Figure 4-19. Click **Finish**.

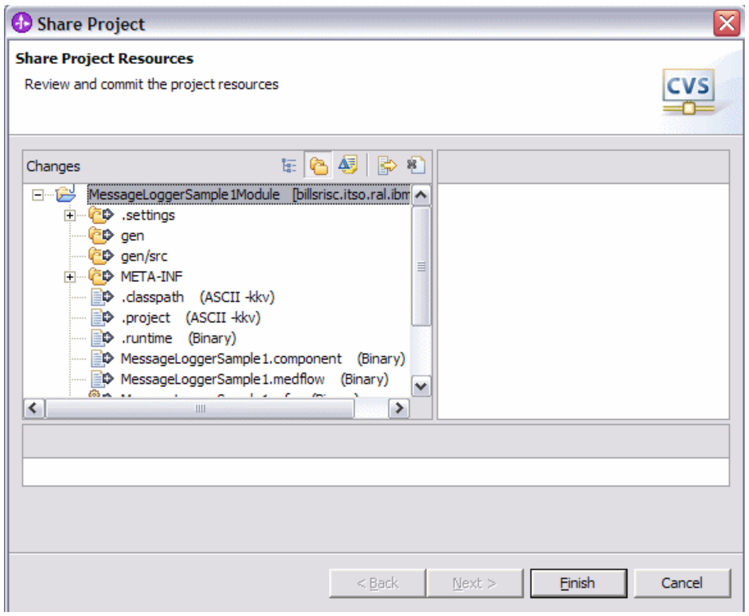
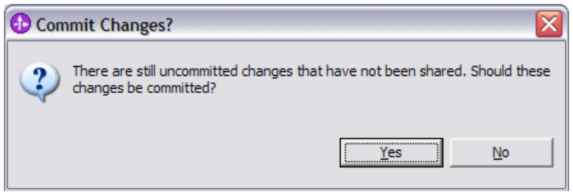

Share Project
Share Project Resources
Review and commit the project resources
CVS
Changes
MessageLoggerSample1Module [billsrisc.itso.ral.ibm
.settings
gen
gen/src
META-INF
.classpath (ASCII -kkv)
.project (ASCII -kkv)
.runtime (Binary)
MessageLoggerSample1.component (Binary)
MessageLoggerSample1.medflow (Binary)
< Back
Next >
Finish
Cancel

Figure 4-19 Share project resources

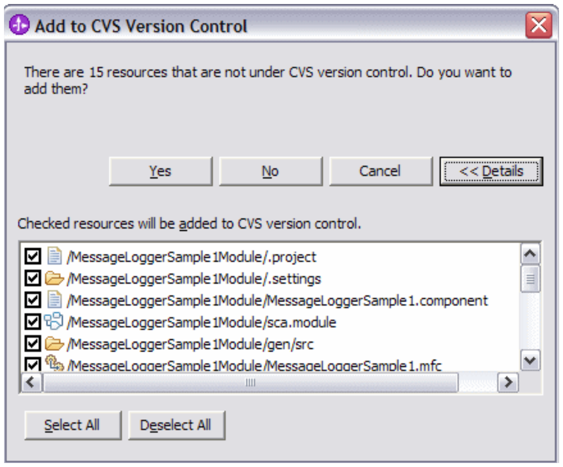
6. You will notify that there are uncommitted changes that are yet to be shared as shown in Figure 4-20. Click **Yes**.



There are still uncommitted changes that have not been shared. Should these changes be committed?

Figure 4-20 Commit changes

7. You will next be presented with a window that shows you the number of files that are will be added to CVS repository. Click **Details** to see the list of files that will be added (Figure 4-21 on page 78). Since the entire module is being added to CVS repository, all the files will be selected. Click **Yes**.



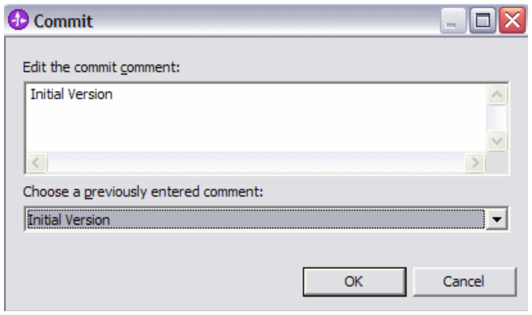
There are 15 resources that are not under CVS version control. Do you want to add them?

Checked resources will be added to CVS version control.

- ☑ /MessageLoggerSample1Module/.project
- ☑ /MessageLoggerSample1Module/.settings
- ☑ /MessageLoggerSample1Module/MessageLoggerSample1.component
- ☑ /MessageLoggerSample1Module/sca.module
- ☑ /MessageLoggerSample1Module/gen/src
- ☑ /MessageLoggerSample1Module/MessageLoggerSample1.mfc

Figure 4-21 Number of resources

8. You can add an optional comment on version tracking as shown in Figure 4-22. Click **OK**.



Edit the commit comment: Initial Version

Choose a previously entered comment: Initial Version

Figure 4-22 Version

9. All the files and folders for the module will be uploaded to the CVS repository and the module will be available for check out. In the workspace, the module and the artifacts within the module that have been shared will be marked with a small disk decorator as shown in Figure 4-23 on page 79, to indicate that they are now shared resources. Also note, the CVS repository name will appear in brackets next to the module name. This will help differentiate local projects and shared projects.

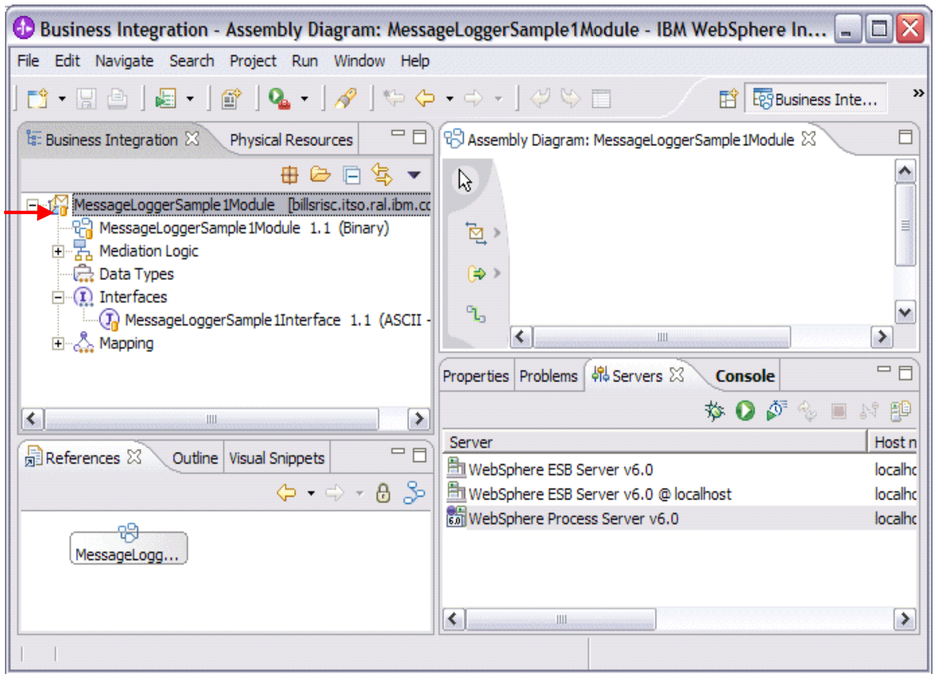

Business Integration - Assembly Diagram: MessageLoggerSample1Module - IBM WebSphere In...
File Edit Navigate Search Project Run Window Help
Business Inte...
Business Integration
Physical Resources
Assembly Diagram: MessageLoggerSample1Module
MessageLoggerSample1Module [billsrisc.itso.ral.ibm.cc
MessageLoggerSample1Module 1.1 (Binary)
Mediation Logic
Data Types
Interfaces
MessageLoggerSample1Interface 1.1 (ASCII -
Mapping
Properties
Problems
Servers
Console
Server
Host n
WebSphere ESB Server v6.0
localhc
WebSphere ESB Server v6.0 @ localhost
localhc
WebSphere Process Server v6.0
localhc
References
Outline
Visual Snippets
MessageLogg...

Figure 4-23 Shared resources

Users that have access to this CVS repository can now check out this module to their local workspace. This can be done from the CVS Repository Exploring perspective:

1. Open the CVS Repository Exploring perspective
2. Expand the **HEAD** repository location.
3. Select the module to check out, right click and select **Check Out** (Figure 4-24).

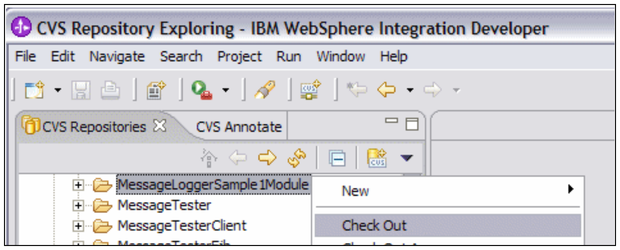
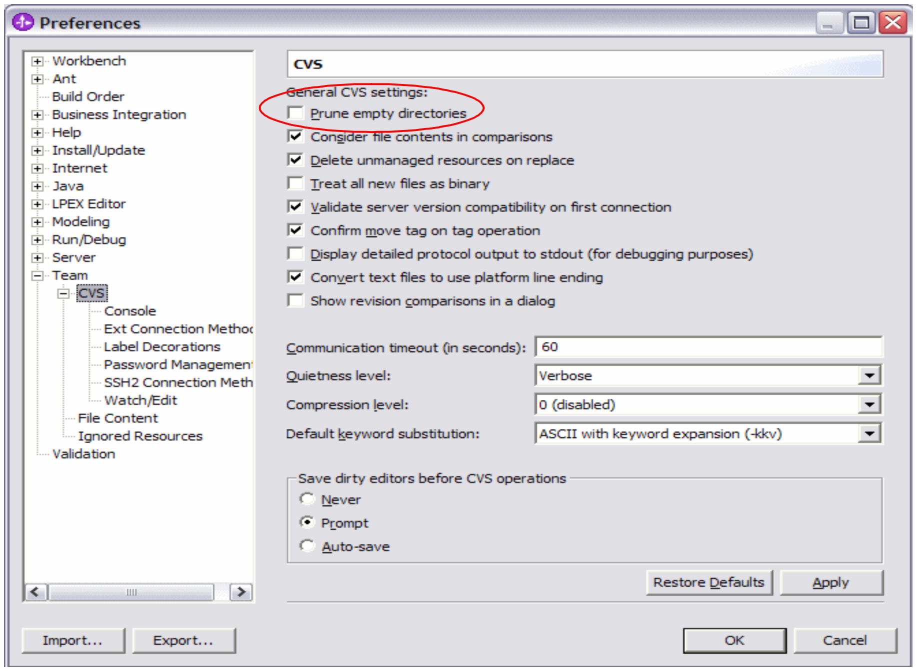

CVS Repository Exploring - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
CVS Repositories
CVS Annotate
MessageLoggerSample1Module
MessageTester
MessageTesterClient
New
Check Out

Figure 4-24 Checking out a module

It is not required to create a module in the workspace before it is checked out from the CVS repository. The check out will extract the entire service module from CVS and WebSphere Integration Developer will build this project generating all the staging projects. The module will appear in the Business Integration perspective.

There are a few known issues when using team development functionality with a CVS repository as discussed below.

- After check out a module, you may get an error in the module that the gen/src folder is missing. The gen/src folder contains derived files generated by WebSphere Integration Developer when building the project. Since CVS does not commit derived files on check in, the gen/src folder will get committed as an empty folder structure, CVS by default will prune empty folders when the module is checked out to the local workspace. You can disable the directory pruning option from **Window -> Preferences -> Team -> CVS** and uncheck **Prune empty directories** option as shown in Figure 4-25.



CVS

General CVS settings:

- [ ] Prune empty directories
- [✓] Consider file contents in comparisons
- [✓] Delete unmanaged resources on replace
- [ ] Treat all new files as binary
- [✓] Validate server version compatibility on first connection
- [✓] Confirm move tag on tag operation
- [ ] Display detailed protocol output to stdout (for debugging purposes)
- [✓] Convert text files to use platform line ending
- [ ] Show revision comparisons in a dialog

Communication timeout (in seconds):	60
Quietness level:	Verbose
Compression level:	0 (disabled)
Default keyword substitution:	ASCII with keyword expansion (-kkv)

Save dirty editors before CVS operations

Never [✓ Prompt] Auto-save

Figure 4-25 Prune empty directories

- During a team development project, it is possible for several users to check out the same modules and make modifications. Its highly recommended that users synchronize often to identify content discrepancies between local and remote versions of the artifact. By synchronizing often, it will be possible to identify conflicts as early as possible. Its a good practice to avoid having two or more people develop the same artifact at the same time.

Synchronizing between local workspace and CVS repository can be performed from the Business Integration perspective by right clicking on a project and selecting **Team -> Synchronize with repository**.

- Eclipse provides the option of committing files to CVS repository without synchronizing the workspace first. This can be done by right-clicking on a project, and selecting **Team -> Commit** from the context menu. This is not recommended as all the local changes will be committed to the server and will overwrite any existing conflicts between the local workspace and the CVS repository. It is recommended to perform a Synchronize with repository first and check the conflicts if any before committing the local workspace to CVS.
- If synchronize with repository shows conflicts between the local workspace and the repository, it is recommended you check the conflicts, and accordingly either **Update** the local workspace with the remote version on CVS, or **Commit** the local workspace version to CVS.
- Besides service modules, it is possible to manage library projects using a CVS repository. It should be noted that module projects could be dependant on a library, and changes to artifacts in the library can affect these module projects.

Note: For more in-depth information about team development, see:

- Team development with WebSphere Integration Developer and WebSphere Process Server: Developing applications using CVS

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0604_beers/0604_beers.html`

4.5 Integration test considerations

This section refers to the development environment not just as a software tool or a development workstation, but as a complete *stage* in the integration development life cycle.

Typically, a development project consists of a number of *integration developers* building a solution. Each developer is responsible for a set of deliverables and need not be concerned with the specifics of how one module connects to another.

Integration developers write mediation modules, unit test their components using the Integration Test Client and commit their modules to a source repository, such as CVS.

Assembling the complete solution by specifying bindings between modules is the responsibility of the *integration specialist*. Figure 4-26 shows the typical topology of a development environment.

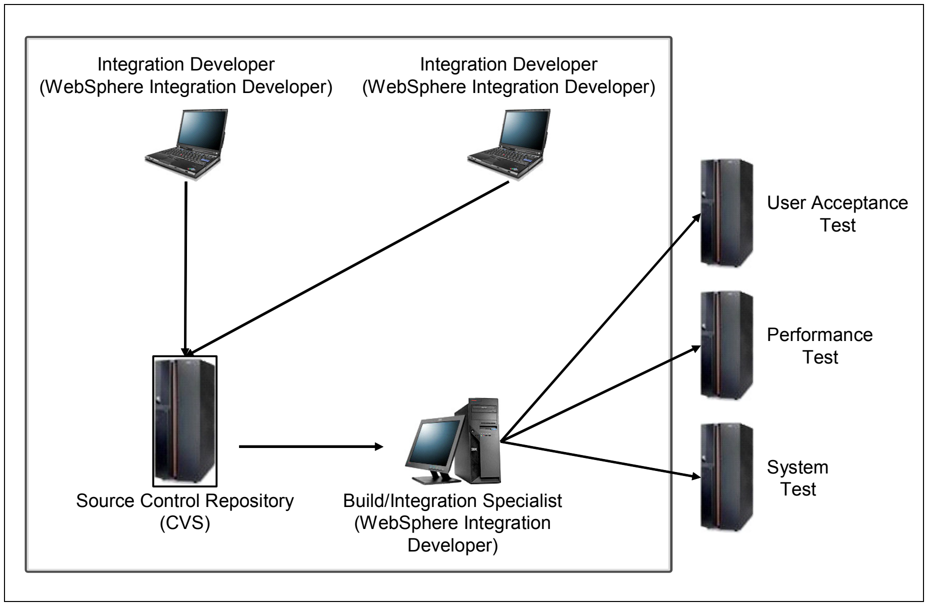

Integration Developer (WebSphere Integration Developer)
Integration Developer (WebSphere Integration Developer)
User Acceptance Test
Performance Test
System Test
Source Control Repository (CVS)
Build/Integration Specialist (WebSphere Integration Developer)

Figure 4-26 Development environment

The need for an integration workspace

Integration developers will typically write and test their components against interfaces which are available in shared libraries. This means that integration developers may not have the complete solution in their individual workspaces, and therefore may not be equipped to resolve the relationships between their modules and other modules.

The integration specialist is responsible for making sure the solution builds correctly, is properly assembled, and passes a set of basic tests before it can be promoted to external test environments and eventually production.

The tasks performed by the integration specialist include:

- Managing code versions
- Checking out modules from code repository
- Building modules
- Assembling inter-module connections
- Testing the solution

- Exporting EAR files
- Shipping code to external environments

The integration specialist works with WebSphere Integration Developer, the same tool used by integration developers to write components.

Periodically, the integration developer will enforce a code freeze, check all modules out from the source code repository, ensure all modules build without errors, make sure the bindings between modules are set correctly, smoke-test the solution, tag all modules with a version level and export the complete solution as a set of EAR files to be promoted to external test environments.

It is common, at the start of an integration project, to allocate integration test activities to a member of the development team. However, as the project grows and the number of activities increases, you will need a dedicated resource to own these tasks and manage communications between the development environment and other external environments.

Given the nature of integration projects, you should plan for a dedicated resource to fulfill the role of integration specialist.

4.6 Troubleshooting installation issues

The final panel of WebSphere Integration Developer installation provides a detailed summary of the installation. This panel shows all the components that were selected for installation and shows the status of the installation for each of these components. This panel also shows the directory name where logs are located.

The logs are located in <WID_INSTALL_DIR>\logs. If any of the components show a status of failure, review the logs for further information.

Depending on the components selected for installation, an installation of WebSphere Integration Developer may also install a complete WebSphere Process Server V6.0.1 and WebSphere Enterprise Service Bus V6.0.1 environment to be used as integration test servers.

The servers will be installed in <WID_INSTALL_DIR>\runtimes\bi_v6 directory. The logs for the server installation are located in <WID_INSTALL_DIR>\runtimes\bi_v6\logs directory.

If the installation summary shows a failure when installing the server, you will need to check the logs for the server.

Besides logs, further debugging for installation can be enabled by starting the install using below command.

```
<disk1>\setup\setup.exe -log # !C:\install-log.txt @ALL
```

For silent installation using a response file, further debugging can be started using the below command

```
<..disk1\setup>\setup.exe -silent -options c:\myresponsefile.txt -log #
!C:\install-log.txt @ALL
```

Its always recommended to check the WebSphere Integration Developer support Web site for any known issues.

```
http://www-306.ibm.com/software/integration/wid/support/
```

5

Setting up the runtime environment

This chapter discusses the WebSphere Enterprise Service Bus runtime environment. We provide an overview of the runtime environment, and a description of the runtime topologies that can be configured. We also provide details of the runtime environments we installed. Finally, we discuss some considerations for establishing various test and production environments that address common requirements.

In this chapter, the following topics are discussed:

- Overview of the runtime
- Stand-alone server topology
- Network Deployment topology
- Extending WebSphere Application Server Network Deployment V6
- Installing WebSphere Enterprise Service Bus
- Planning for multiple test and production environments
- Performing problem determination of runtime installation and customization

5.1 Overview of the runtime environment

In this section we discuss the runtime environment in general terms, so that you can get an understanding of what is involved in establishing the runtime environments needed for development, test, and production activities. If you are familiar with WebSphere Application Server V5, the terminology and configuration topologies are identical to those you've already encountered. If you have experience with WebSphere Application Server V6, you are already acquainted with the idea of a profile, and it is used in the same way for WebSphere Enterprise Service Bus.

5.1.1 Hardware and software requirements

It is always best to consult the WebSphere Enterprise Service Bus product Web site to obtain the most up-to-date information regarding supported hardware and software. This information can be found at:

http://www.ibm.com/software/integration/wsesb/sysreqs

5.1.2 Consider your current environment

WebSphere Enterprise Service Bus V6.01 is based on WebSphere Application Server Network Deployment V6.0.2.3. Your first decision when planning the runtime environment, is between extending your current WebSphere Application Server Network Deployment environment by adding WebSphere Enterprise Service Bus to it, or creating separate WebSphere Enterprise Service Bus servers.

If you have WebSphere Application Server Network Deployment V6 installed and configured to support your development, test and production requirements, you will most likely want to extend these environments by adding WebSphere Enterprise Service Bus to them. We discuss more about how to do that in section 5.4, "Extending WebSphere Application Server V6" on page 96.

If you do not have an existing WebSphere Application Server Network Deployment V6 runtime environment, then you will want to create new servers to run WebSphere Enterprise Service Bus. This is the approach we took, and we discuss the installation and customization steps we took in detail in section 5.5, "Installing WebSphere Enterprise Service Bus" on page 100.

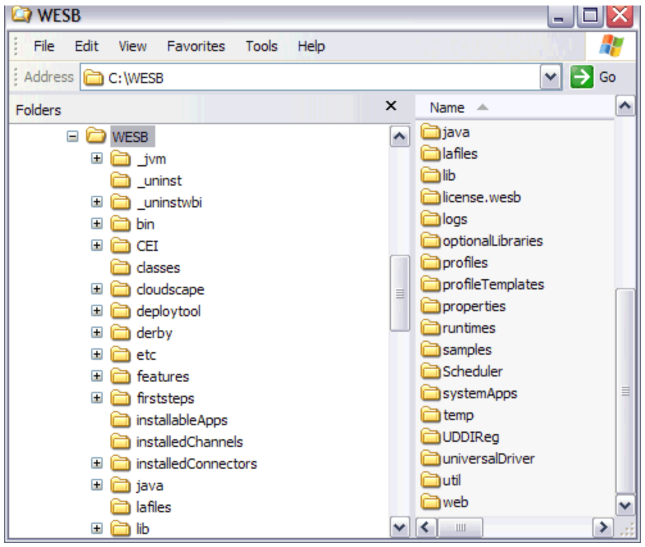
The following sections describe installation and customization steps you will take when you do not have an existing WebSphere Application Server Network Deployment V6 installation, or if you do, in the case where you chose to not extend that installation with the WebSphere Enterprise Service Bus libraries.

5.1.3 What gets installed?

When you install WebSphere Enterprise Service Bus on a target machine, the installation wizard will perform a set of checks against your system. If the checks pass, it will first install WebSphere Application Server Network Deployment V6.0.2.3. When that is complete, the WebSphere Enterprise Service Bus product is installed into the same set of directories as were used for the first step. Finally, a set of profiles need to be created in order to customize the installation and allow you to start servers.

When using the Installation wizard, you will have the option to perform a Complete installation or a Custom installation. A Complete installation will automatically create a default profile which corresponds to a stand-alone server topology. If you choose the Custom installation, this profile is not created and you can use the First Steps dialog, or the start menu options to continue with profile creation.

Lets look first at the results of a Complete installation performed on a Windows machine. The directory structure shown in Figure 5-1 on page 87 is the result of the installation steps.



```
WESB
  _jvm
  _uninst
  _uninstwbi
  bin
  CEI
  classes
  cloudscape
  deploytool
  derby
  etc
  features
  firststeps
  installableApps
  installedChannels
  installedConnectors
  java
  lafiles
  lib
```

```
java
lafiles
lib
license.wesb
logs
optionalLibraries
profiles
profileTemplates
properties
runtimes
samples
Scheduler
systemApps
temp
UDDIReg
universalDriver
util
web
```

Figure 5-1 Installation directories created

Look at both panes of the Windows Explorer output in Figure 5-1 on page 87 to see the full set of subdirectories created. This resembles very closely the set of directories created by a WebSphere Application Server Network Deployment V6 installation, but it does have some important additions.

Lets first look at the samples subdirectory. Notice there is an ESBSamplesGallery and an ESBStockQuoteSample directory available which are shown in Figure 5-2 on page 88.

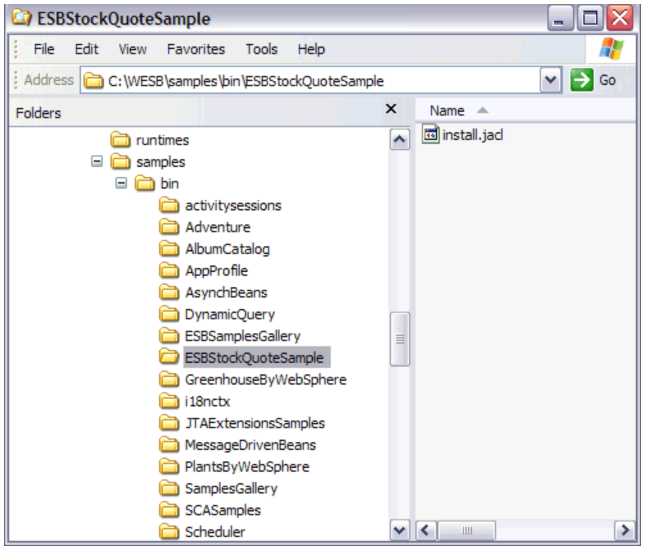
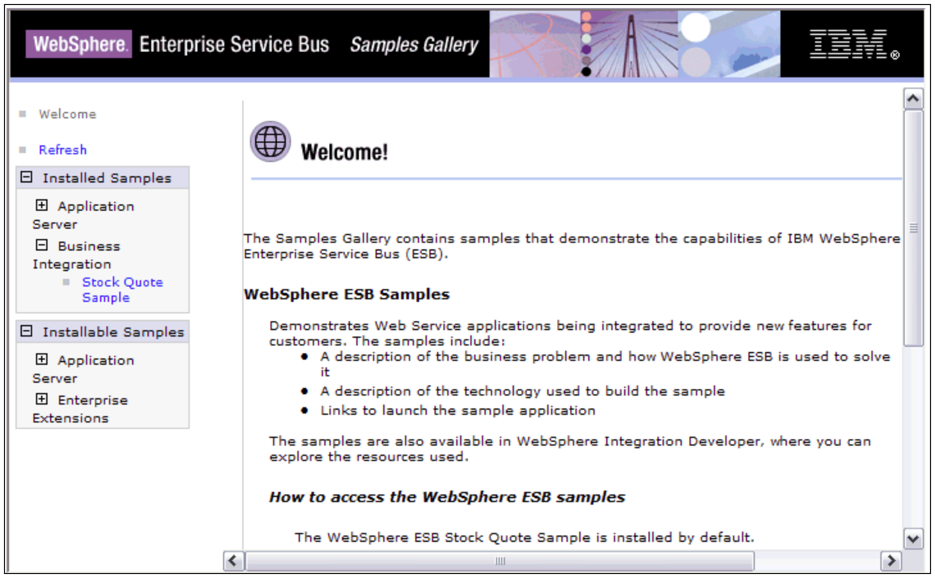


Figure 5-2 Samples available after installation

When you choose the Complete installation, these samples are deployed for you automatically and the required database and server configuration is done for you. After your server is started, you can launch the samples from the First Steps menu item accessed from your default profile on the Windows Start options. See Figure 5-21 on page 108 for a view of the start option menu items. Alternatively there is a Samples Gallery option available on the default profile menu. When you launch the samples gallery, you will see the browser page shown in Figure 5-3 on page 89.



Welcome!

The Samples Gallery contains samples that demonstrate the capabilities of IBM WebSphere Enterprise Service Bus (ESB).

WebSphere ESB Samples

Demonstrates Web Service applications being integrated to provide new features for customers. The samples include:

- A description of the business problem and how WebSphere ESB is used to solve it
- A description of the technology used to build the sample
- Links to launch the sample application

The samples are also available in WebSphere Integration Developer, where you can explore the resources used.

How to access the WebSphere ESB samples

The WebSphere ESB Stock Quote Sample is installed by default.

Figure 5-3 The WebSphere Enterprise Service Bus Samples Gallery

Looking a bit deeper at the installation directory, we see a two more subdirectories that are specific to WebSphere Enterprise Service Bus.

The first is the CEI subdirectory, as shown in Figure 5-4 on page 89.

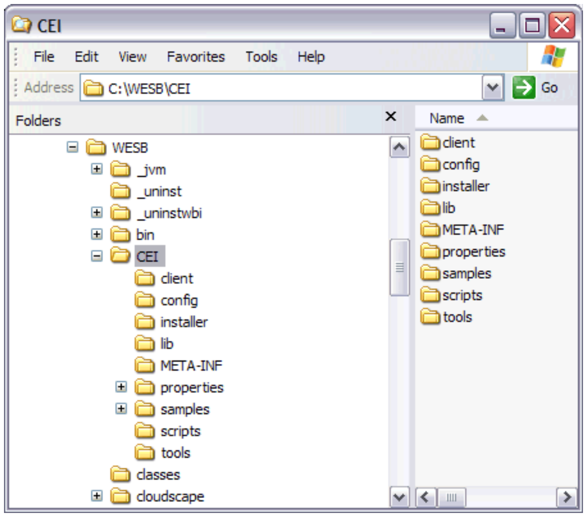
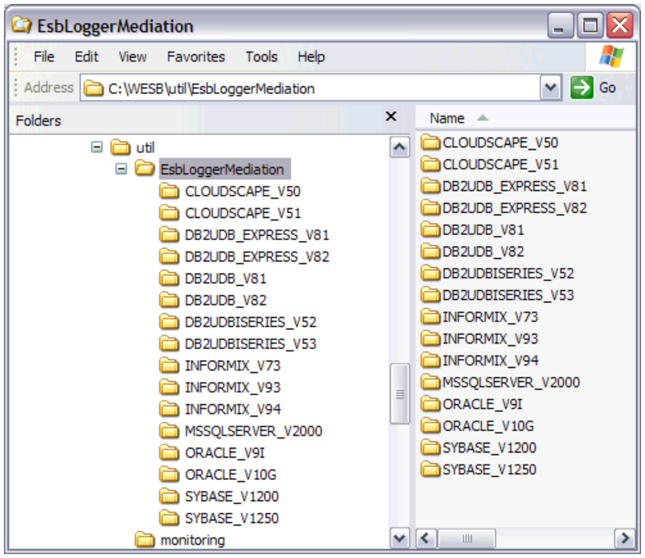


Figure 5-4 CEI subdirectory after installation

CEI is the Common Events Infrastructure that is used by WebSphere Enterprise Service Bus components to log execution activity. You will become more familiar with CEI as you examine the functions of WebSphere Enterprise Service Bus.

A second subdirectory of interest appears in the \util directory, named EsbLoggerMediation, shown Figure 5-5 on page 90.



CLOUDSCAPE_V50
CLOUDSCAPE_V51
DB2UDB_EXPRESS_V81
DB2UDB_EXPRESS_V82
DB2UDB_V81
DB2UDB_V82
DB2UDBISERIES_V52
DB2UDBISERIES_V53
INFORMIX_V73
INFORMIX_V93
INFORMIX_V94
MSSQLSERVER_V2000
ORACLE_V9I
ORACLE_V10G
SYBASE_V1200
SYBASE_V1250

Figure 5-5 EsbLoggerMediation directory

This directory is a unique part of WebSphere Enterprise Service Bus and its contents will help you to configure a more advanced persistence mechanism for components that use the Logger mediation primitive. Initially, Cloudscape™ is used for this purpose and its already configured for you, yet as you move to more advanced stages of the test cycle, you will probably want to use a relational database for this purpose.

Finally, you should notice a number of subdirectories named *logs*. These directories contain files generated during installation and customization that capture the activities performed during those steps. If your installation completed successfully, it is sufficient to simply know that the logs exist, and if you are very inquisitive, you may want to go and browse the files there. If you have problems with installation, the logs will be needed to diagnose what went wrong, and we discuss this in more detail in section 5.7, “Problem determination for runtime installation and customization” on page 127.

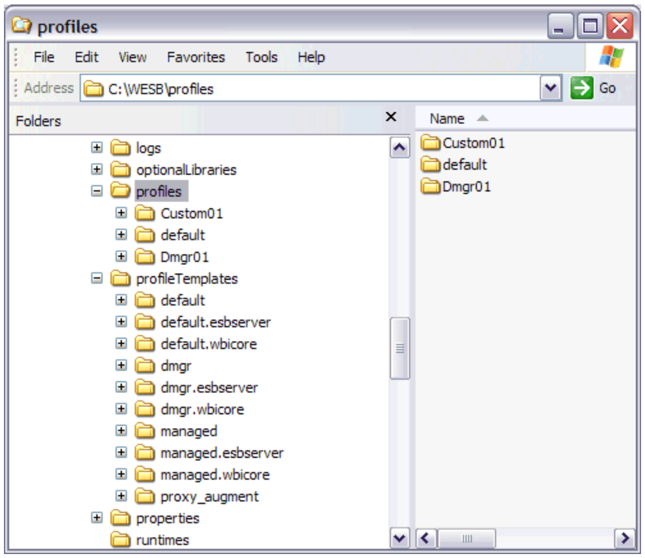
5.1.4 What gets customized?

Generally the process of customization is one that modifies the product installation libraries so that a working version is created. For WebSphere Enterprise Service Bus the customization process is encapsulated by the tasks performed to create *profiles*. This is a set of directories and files on the machine specific to a given server. You can have multiple profiles on a machine, each representing a different server, its configuration, and commands to interact with it. These profiles make use of the product installation libraries as though they were in read-only mode. Files that must be updated for a particular server are copied and kept as part of the profile.

As mentioned above, if you choose a Complete installation, a default profile is created for you, which provides a customized stand-alone server. You can continue to create other profiles for a Deployment Manager and a Custom Node, which we discuss in detail in section 5.5.2, “A common development integration test runtime environment” on page 108. If you choose to perform a Custom installation, no profiles are created automatically and you have full control over that process using the First Steps dialog.

All of the profiles can coexist on a given machine, although depending on what kind they are, and how they were built (using common or unique TCP/IP ports, for example) the servers associated with them may not all be started at the same time.

Profiles are commonly kept in the profiles directory for the installed copy of WebSphere Enterprise Service Bus. You can see in Figure 5-6 on page 92 the results of the customization steps we took after installation. We created a default, a Deployment Manager and a Custom profile on our machine.



- logs
- optionalLibraries
- profiles
  - Custom01
  - default
  - Dmgr01
- profileTemplates
  - default
  - default.esbserver
  - default.wbicore
  - dmgr
  - dmgr.esbserver
  - dmgr.wbicore
  - managed
  - managed.esbserver
  - managed.wbicore
  - proxy_augment
- properties
- runtimes

Custom01
default
Dmgr01

Figure 5-6 Profiles resulting from installation customization

5.1.5 What gets configured?

After installation and customization, which tend to be one-time kinds of tasks, is the task of configuring the runtime environment.

When you do a Complete installation, all configuration steps required to have an operational server are done for you. You can begin to deploy your mediation modules to this runtime immediately and begin testing.

The WebSphere Enterprise Service Bus Profile Creation wizard performs all of the configuration steps for you when you choose the Complete installation, or when you use the wizard to create additional stand-alone server profiles. If you start the server and log onto the administrative console, you can see evidence of the following configuration steps having been completed:

- A service integration bus named **SCA.SYSTEM.*cell_name*.Bus** has been configured. This is a standard service integration bus which will have bus destinations added when SCA modules are deployed. These destinations are used to hold messages that are being processed for components of a mediation module that uses asynchronous interactions. The bus will be configured to have your server as a bus member and a messaging engine will be created on your server.
- A second service integration bus named **SCA.APPLICATION.*cell_name*.Bus** is configured with the server as a bus

member and a messaging engine. This service integration bus is used to define JMS queue destinations (and other JMS resources) for modules deployed with JMS bindings.

- A third service integration bus named **CommonEventInfrastructure _Bus** is configured, again with the server as a bus member and messaging engine provider. This bus is used by the CEI service.
- The three service integration buses are set up to use Cloudscape for message persistence.
- An enterprise application named **sca.sib.mediation** is deployed to the server. This application collaborates with your mediation modules and the runtime to provide the mediation services required.
- The CEI service is configured to use the Cloudscape database for persistence.
- A database and datasource are configured for use by any mediation logger components that are deployed inside mediation modules.

For a Deployment Manager cell topology, you will use the administrative console to add servers to a managed node. When you do this, there will be a template for an ESBServer, and if you use it only the two SCA buses are configured, however they have no bus members initially. For this topology you need to do much of the configuration work described above, yourself, since the configuration of these resources in a cell with potentially numerous servers or clusters of servers offers many options beyond those of a stand-alone server environment. We discuss how to accomplish these steps in section 5.4.3, “Final configuration steps” on page 98.

5.2 Stand-alone server topology

This is the simplest configuration to create and we recommend building one at least once to familiarize yourself with the basic product and its configuration.

With a stand-alone server topology, you will have a fully functional product on a single server. You can administer the server using the administration console, or with the scripting interface, yet it is completely separate from all other servers.

You can have as many instances of a stand-alone server as you require. If you intend to install multiple instances of a stand-alone server on a single machine, you should make certain there are sufficient resources available to support the workload. You should also take care to use unique TCP/IP ports for each stand-alone server, if you intend to run them concurrently.

When you install WebSphere Enterprise Service Bus, you are copying the product libraries to the target machine, and they are not modified afterward, except through the installation of maintenance to the product. Installation only needs to happen once for any machine that will run WebSphere Enterprise Service Bus, in any topology.

In order to create a customized runtime server, you create a profile. If you intend to customize multiple servers on a single machine, the names of the profiles must be unique. Figure 5-7 on page 94 illustrates a stand-alone server with the default profile.

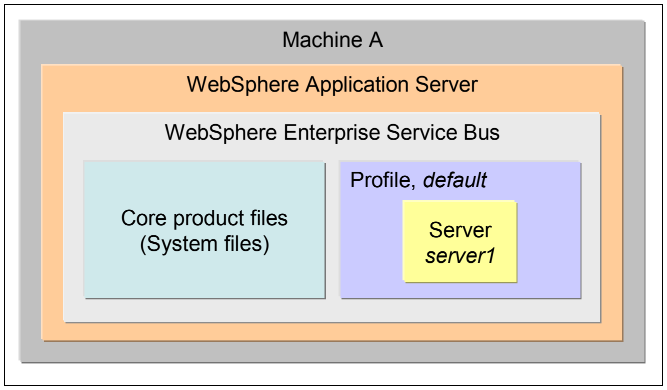

Machine A
WebSphere Application Server
WebSphere Enterprise Service Bus
Core product files (System files)
Profile, *default*
Server *server1*

Figure 5-7 Stand-alone server topology

There are occasions when you may want to install WebSphere Enterprise Service Bus more than once on a given machine. Consider a situation where you want to create a system integration test (SIT) environment, and a quality assurance (QA) environment on the same machine. You could install the product twice, and create two profiles to support the two test environments. You would also have the flexibility to install maintenance for WebSphere Enterprise Service Bus into the SIT environment without disturbing the QA environment. When fully tested, the service could subsequently be installed to the QA environment.

Tip: Installing WebSphere Enterprise Service Bus a second time on the same machine gives you the ability to run servers at different maintenance levels.

5.3 Network Deployment topology

In a Network Deployment topology, a set of servers is organized into a cell for the purpose of having a central place for administration. Each cell is comprised of two or more nodes which each contain one or more servers. Just like with the stand-alone server, the WebSphere Enterprise Service Bus product is only installed once on each machine that will be hosting a server. Unlike a stand-alone server, you will create two distinct profiles:

- One profile will be created for the Deployment Manager. This server provides the administration for the cell. The administrative console runs in this server and can be used to manage the configuration of all the servers in the cell. Most often administrative scripts are executed against a running Deployment Manager, whose job is to distribute the updates throughout affected nodes in the cell. The deployment manager runs in its own node, which is a logical grouping of managed servers.
- The second profile is known as a custom node. A custom node is a profile that represents an empty node. Once you create a custom node, you will invoke a process known as federation, which brings the custom node into the cell managed by the Deployment Manager. Within a custom node is a process known as a node agent, which is responsible for managing the servers and their configuration. When a node has a node agent, it is considered a managed node. Figure 5-8 on page 95 illustrates a Network Deployment topology.

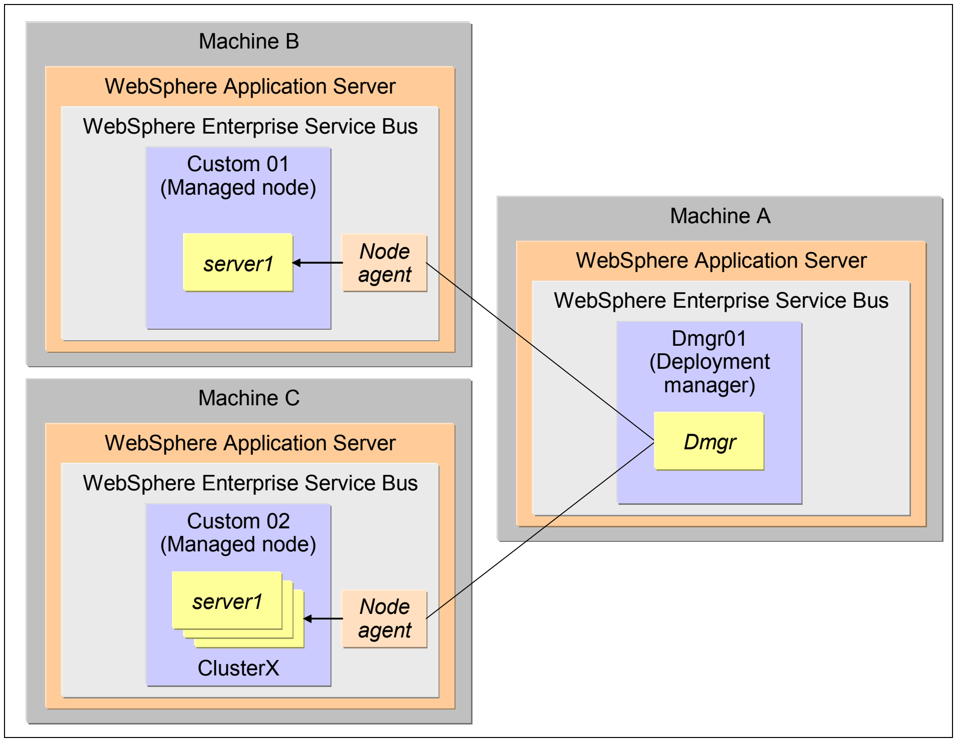

Machine B
WebSphere Application Server
WebSphere Enterprise Service Bus
Custom 01 (Managed node)
server1
Node agent
Machine C
WebSphere Application Server
WebSphere Enterprise Service Bus
Custom 02 (Managed node)
server1
Node agent
ClusterX
Machine A
WebSphere Application Server
WebSphere Enterprise Service Bus
Dmgr01 (Deployment manager)
Dmgr

Figure 5-8 Network Deployment topology

Since the custom node is initially empty (except for the node agent), you will need to add servers to the node after it is federated. Adding servers to a node is a task which is completed using the administrative console or the wsadmin scripting interface. Servers must be added to a custom node before applications or mediation modules can be deployed.

The Deployment Manager can be used to start and stop servers in the cell, however, you must start the node agent(s) before those tasks can be performed. The Deployment Manager cannot start node agents. You can use the startNode batch file or shell script to start a node agent.

You can configure a cell with multiple nodes that all exist on one physical machine, or they can span multiple machines. Each node in the cell exists only on one physical machine. You can see that there are a number of topology choices available which allow the cell to meet scalability and fail-over requirements.

5.4 Extending WebSphere Application Server V6

In the sections that follow we will discuss how you can use an existing WebSphere Application Server Network Deployment V6 and extend it to incorporate the features and functions of WebSphere Enterprise Service Bus. We will cover:

- Installation
- Augmenting profiles
- Final configuration steps

5.4.1 Installation

If you have WebSphere Application Server Network Deployment V6 already installed and configured, you can use it as the basis of your WebSphere Enterprise Service Bus installation. During the install of WebSphere Enterprise Service Bus, the wizard will detect the presence of WebSphere Application Server and ask if you want to do a completely new install or extend your current version 6.0x installation. If you chose to extend your current installation, the following will occur:

- Your WebSphere Application Server service level will be upgraded to version 6.0.2.3.
- The product libraries for WebSphere Enterprise Service Bus will be installed in the same directories as your product libraries for WebSphere Application Server Network Deployment.

Extending your current WebSphere Application Server Network Deployment environment with WebSphere Enterprise Service Bus allows you to manage only one set of product libraries making application of maintenance and procedures for backup and recovery more straightforward.

5.4.2 Augmenting profiles

Once you have completed the installation of WebSphere Enterprise Service Bus, you will need to augment the profiles associated with your WebSphere Application Server Network Deployment configuration. Augmenting the profiles adds the required additional configuration for WebSphere Enterprise Service Bus to your existing profiles.

For a Stand-alone server configuration this is a single step performed with the server stopped. Launch the Profile creation wizard. Select a Stand-alone Application Server, and the wizard will detect your existing profiles of that type and ask which one you would like augment. Completing the wizard will result in an updated profile that has all of the original configuration information as well as the new configuration elements for WebSphere Enterprise Service Bus.

If you have a Deployment Manager cell, the process requires several more steps as outlined below:

1. Stop all of the servers and node agents in the cell.
2. Stop the Deployment Manager.
3. Launch the Profile creation wizard and augment the Deployment Manager profile.
4. Start the Deployment Manager.
5. Unfederate each managed node in the cell. You can use the **`removeNode`** command found in the \bin directory of the custom profile for the node.
6. Launch the Profile creation wizard and augment the custom profile associated with the unfederated node.
7. Federate the node back into the cell. You can use the **`addNode`** command in the \bin directory of the custom profile for the node.

Note: Use caution when unfederating and federating managed nodes. Options on the command will allow you to preserve existing servers and application deployments, but they may be lost if not specified properly. Use the WebSphere Enterprise Service Bus InfoCenter to find details on the addNode and removeNode commands.

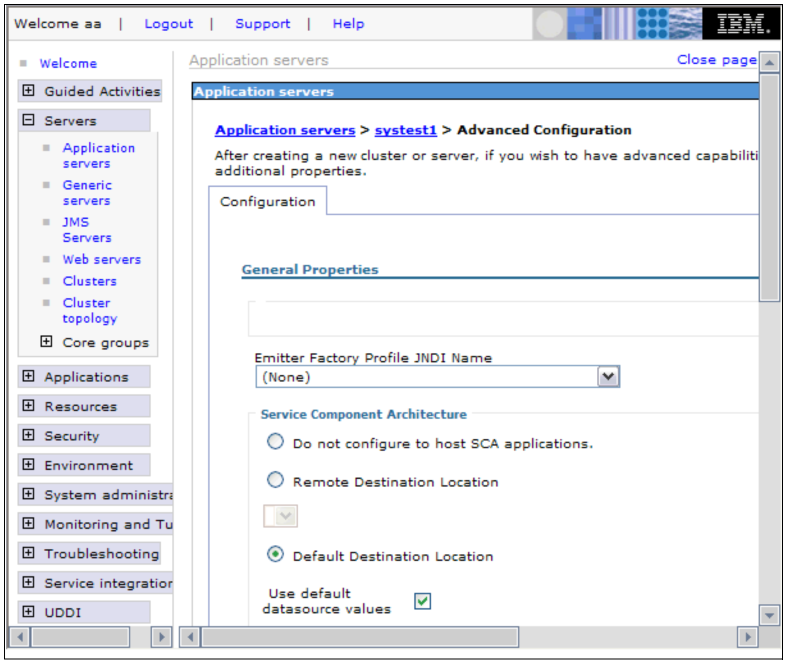
5.4.3 Final configuration steps

After you augment the profiles, there remain two final configuration steps.

First, you must enable the server to host SCA modules. If you look at the server before you start this process, you will notice that it has the two SCA buses defined to it, but has not associated them with a messaging engine (there are no bus members).

You will need to configure the persistence mechanism used by the buses, and configure their bus members. Fortunately there is a *helper dialog* to help you with this task the first time the server gains ESB capabilities.

1. In the administrative console, select **Servers**-> **Application servers**-> ***your_server*** and the **Advanced Configuration** link under the Business Integration heading.
2. In the Service Component Architecture section of the pane, select **Default Destination Location** if you want to be able to deploy mediation modules to this server and if you want to use the messaging resources of this server. Also select the checkbox indicating you want to **use the default datasource values** for message persistence, which is Cloudscape. By choosing this option, the service integration bus will automatically create the Cloudscape datasources and tables required for the bus members. These selections are shown in Figure 5-9 on page 99.



Application servers > systest1 > Advanced Configuration

After creating a new cluster or server, if you wish to have advanced capabiliti additional properties.

Configuration

General Properties

Emitter Factory Profile JNDI Name
(None)

Service Component Architecture

( ) Do not configure to host SCA applications.

( ) Remote Destination Location

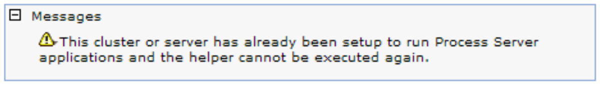
(●) Default Destination Location

Use default datasource values ✓

Figure 5-9 Configuring the server to host SCA modules

There are many alternatives to the combination of settings we chose here. It is possible to disable the deployment of SCA modules to this server if you select Do not configure to host SCA applications. Also, you can configure the server so that SCA applications can be deployed to it, but it will use the messaging resources of a remote server for the destinations and activation specifications required by any mediation module you deploy. Setting a remote server adds it as a member of the two SCA buses. Similarly, there are numerous alternatives for the persistence datasource, and you should be sure your choice here is consistent with the intended purpose of your server.

Note that the helper dialog can only be executed a single time. If you make changes and save them, subsequent attempts to use the helper dialog will show the message which appears in Figure 5-10 on page 99.


Messages

This cluster or server has already been setup to run Process Server applications and the helper cannot be executed again.

Figure 5-10 Advanced configuration helper is only enabled for one-time usage

Note also, that if you revisit the helper dialog after having made updates, the status of the SCA configuration may not be displayed properly. In order to determine the actual status, you must look at the buses and datasources directly.

The second configuration task is for the Common Event Infrastructure (CEI) service and its datasource, (notice that CEI is automatically configured for you if you are augmenting a Stand-alone server). This can be accomplished most easily in the above pane as well, if you want to again use the default Cloudscape database and configuration for CEI. In that case select **None** from the dropdown named **Emitter Factory Profile JNDI Name**.

When you have made the necessary selections in this dialog, click **OK** and save your changes.

5.5 Installing WebSphere Enterprise Service Bus

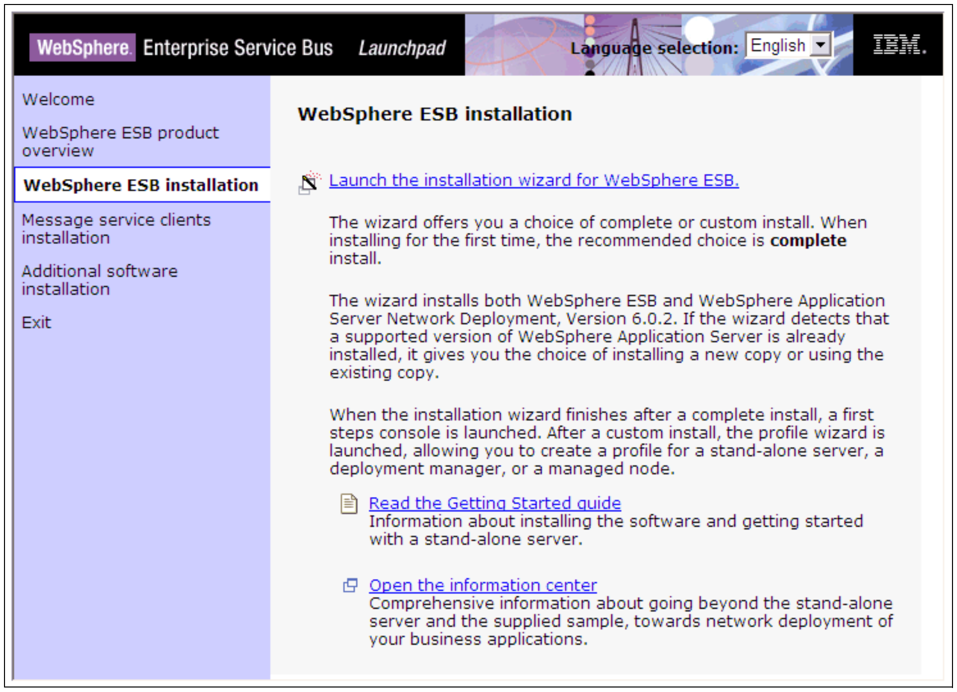
This section details the steps we took to install the product and configure two runtime environments we expect to be common start-up configurations.

5.5.1 An initial runtime environment

This section describes a common initial WebSphere Enterprise Service Bus installation. It describes the steps to install a Stand-alone server topology, using the Complete installation option.

To install the product in the quickest and most straightforward way, follow these steps:

1. To begin installation, start the Launchpad from the product distribution media. You will see the screen shown in Figure 5-11 on page 101.


WebSphere. Enterprise Service Bus Launchpad
Language selection: English
IBM.
Welcome
WebSphere ESB product overview
WebSphere ESB installation
Message service clients installation
Additional software installation
Exit

WebSphere ESB installation

Launch the installation wizard for WebSphere ESB.

The wizard offers you a choice of complete or custom install. When installing for the first time, the recommended choice is **complete** install.

The wizard installs both WebSphere ESB and WebSphere Application Server Network Deployment, Version 6.0.2. If the wizard detects that a supported version of WebSphere Application Server is already installed, it gives you the choice of installing a new copy or using the existing copy.

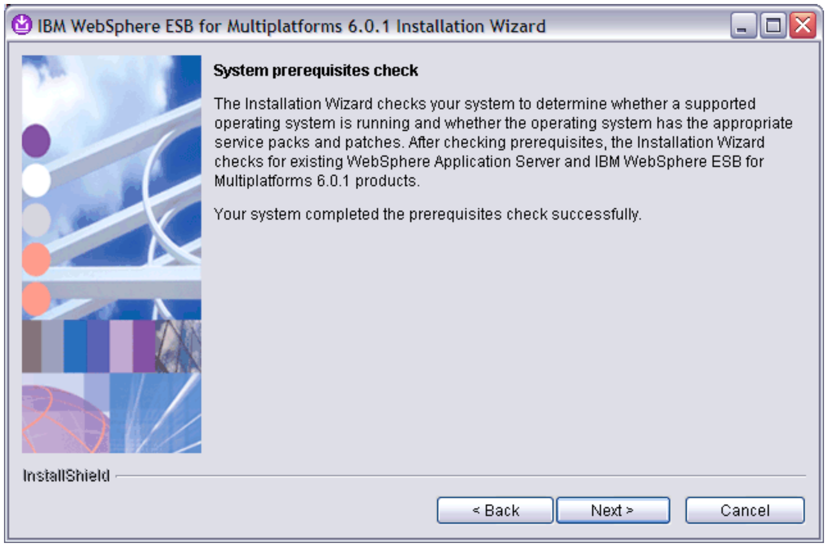
When the installation wizard finishes after a complete install, a first steps console is launched. After a custom install, the profile wizard is launched, allowing you to create a profile for a stand-alone server, a deployment manager, or a managed node.

Read the Getting Started guide
Information about installing the software and getting started with a stand-alone server.

Open the information center
Comprehensive information about going beyond the stand-alone server and the supplied sample, towards network deployment of your business applications.

Figure 5-11 Installation Launchpad

2. If you haven't already done so, we strongly recommend that you review the Getting Started Guide to make sure your operating system is set up properly for installation to succeed. You will also find the InfoCenter to be a valuable resource, and might want to bookmark it in your browser.
3. Click on **Launch the installation wizard for WebSphere ESB**.
4. Make sure the prerequisite checking completes successfully, as shown in Figure 5-12 on page 102 and click **Next**.


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard

System prerequisites check

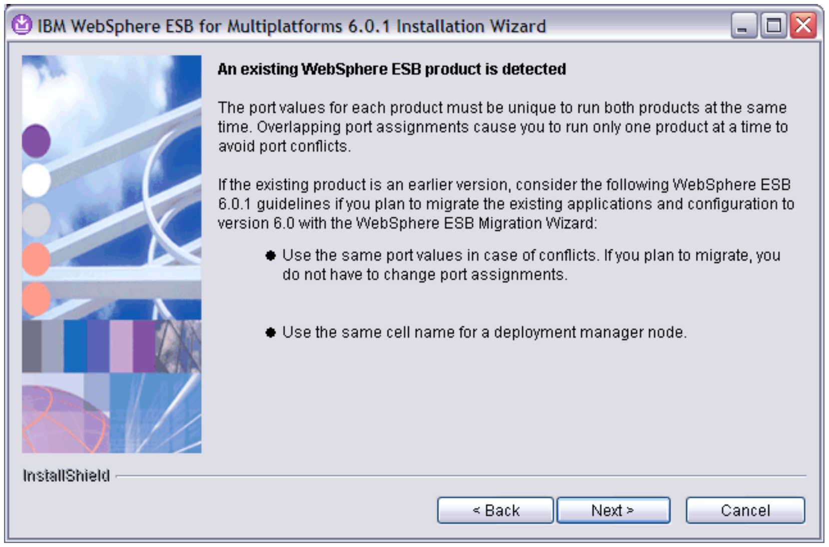
The Installation Wizard checks your system to determine whether a supported operating system is running and whether the operating system has the appropriate service packs and patches. After checking prerequisites, the Installation Wizard checks for existing WebSphere Application Server and IBM WebSphere ESB for Multiplatforms 6.0.1 products.

Your system completed the prerequisites check successfully.

Figure 5-12 Installation Wizard - prerequisite check

5. Review the results of checking for an existing WebSphere Enterprise Service Bus, as shown in Figure 5-13 on page 103 and click **Next**.

Note: This installation was done on a workstation that also had WebSphere Integration Developer installed, and the checks for existing products found the runtime installed there. Since we will not run the WebSphere Integration Developer Test Environment concurrently with the WebSphere Enterprise Service Bus server, we chose to not modify the default port assignments.


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard
InstallShield
< Back
Next >
Cancel

An existing WebSphere ESB product is detected

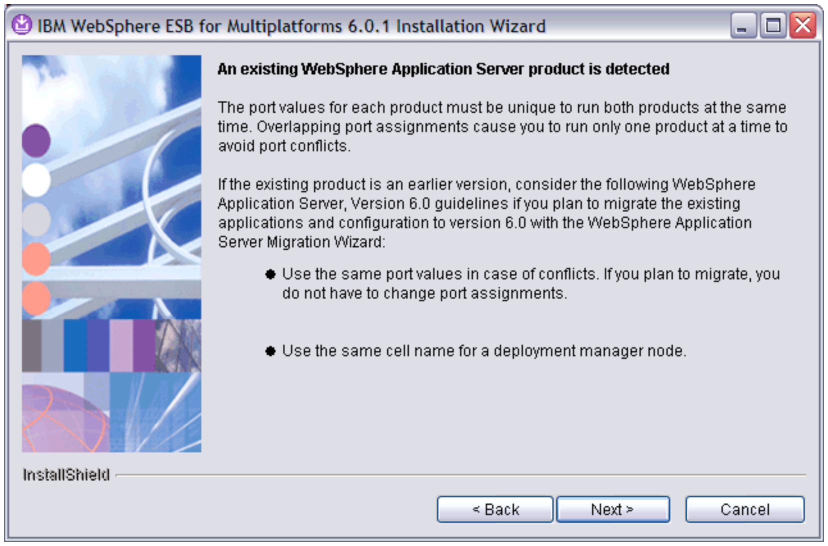
The port values for each product must be unique to run both products at the same time. Overlapping port assignments cause you to run only one product at a time to avoid port conflicts.

If the existing product is an earlier version, consider the following WebSphere ESB 6.0.1 guidelines if you plan to migrate the existing applications and configuration to version 6.0 with the WebSphere ESB Migration Wizard:

- Use the same port values in case of conflicts. If you plan to migrate, you do not have to change port assignments.
- Use the same cell name for a deployment manager node.

Figure 5-13 Installation Wizard - checking for existing installation

6. Review the results of checking for an existing WebSphere Application Server, as shown in Figure 5-14 on page 103, and click **Next**.


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard
InstallShield
< Back
Next >
Cancel

An existing WebSphere Application Server product is detected

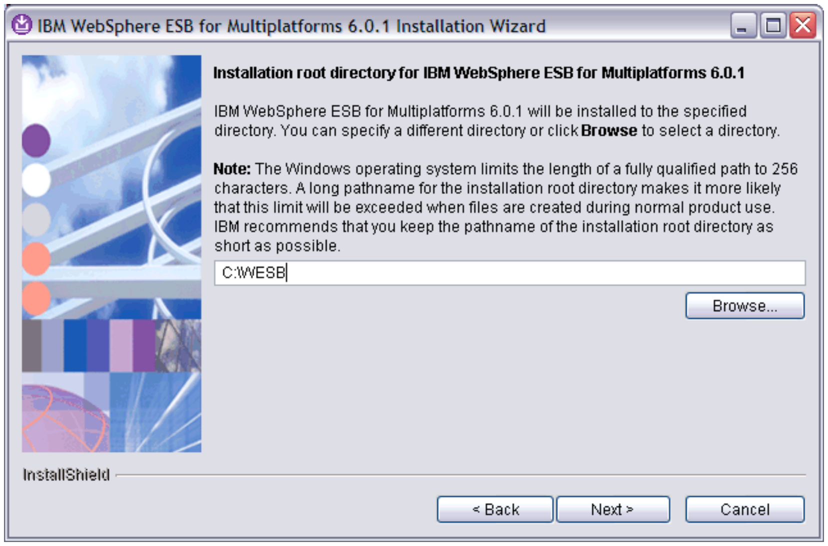
The port values for each product must be unique to run both products at the same time. Overlapping port assignments cause you to run only one product at a time to avoid port conflicts.

If the existing product is an earlier version, consider the following WebSphere Application Server, Version 6.0 guidelines if you plan to migrate the existing applications and configuration to version 6.0 with the WebSphere Application Server Migration Wizard:

- Use the same port values in case of conflicts. If you plan to migrate, you do not have to change port assignments.
- Use the same cell name for a deployment manager node.

Figure 5-14 Installation Wizard - checking for existing installation

7. Enter or browse to the location where you want to install the product and click **Next** (Figure 5-15 on page 104). On Windows platforms, keep the directory path short to avoid any issues with excessive path lengths.


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard

Installation root directory for IBM WebSphere ESB for Multiplatforms 6.0.1

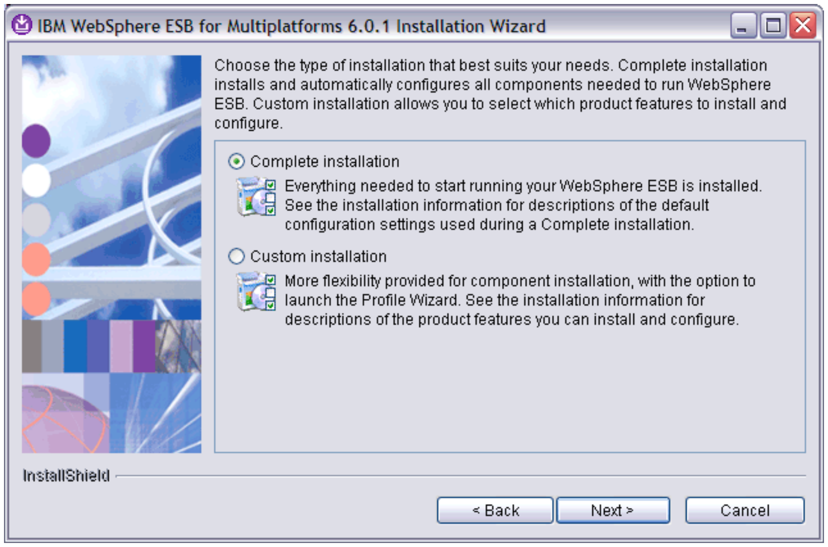
IBM WebSphere ESB for Multiplatforms 6.0.1 will be installed to the specified directory. You can specify a different directory or click **Browse** to select a directory.

Note: The Windows operating system limits the length of a fully qualified path to 256 characters. A long pathname for the installation root directory makes it more likely that this limit will be exceeded when files are created during normal product use. IBM recommends that you keep the pathname of the installation root directory as short as possible.

C:\WESB

Figure 5-15 Installation Wizard - installation directory

8. For this case, we want to do a complete installation, so select the **Complete installation** and click **Next** (Figure 5-16 on page 105).


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard

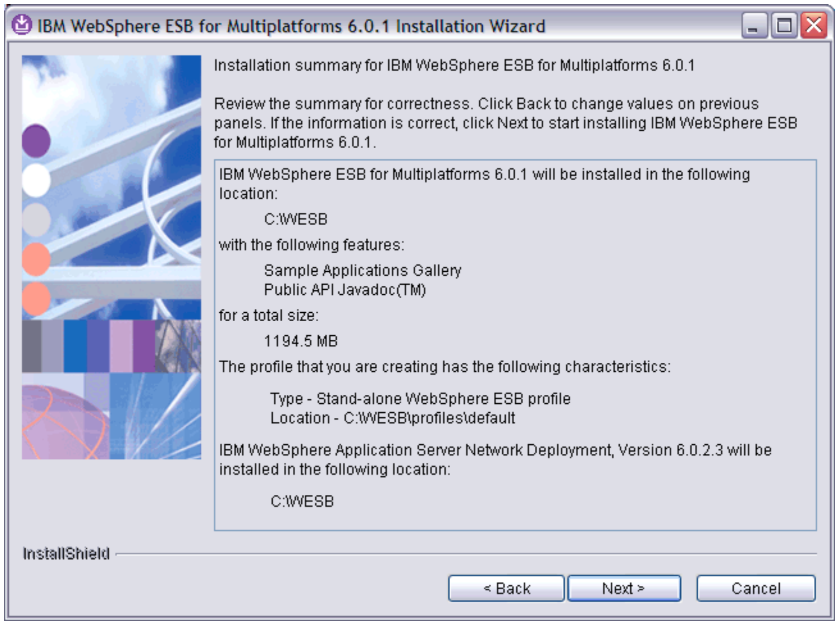
Choose the type of installation that best suits your needs. Complete installation installs and automatically configures all components needed to run WebSphere ESB. Custom installation allows you to select which product features to install and configure.

[✓ Complete installation]
Everything needed to start running your WebSphere ESB is installed. See the installation information for descriptions of the default configuration settings used during a Complete installation.

Custom installation
More flexibility provided for component installation, with the option to launch the Profile Wizard. See the installation information for descriptions of the product features you can install and configure.

Figure 5-16 Installation Wizard - installation type

9. Review the summary before the installation actually starts. When it is correct, click **Next** (Figure 5-17 on page 105).


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard

Installation summary for IBM WebSphere ESB for Multiplatforms 6.0.1

Review the summary for correctness. Click Back to change values on previous panels. If the information is correct, click Next to start installing IBM WebSphere ESB for Multiplatforms 6.0.1.

IBM WebSphere ESB for Multiplatforms 6.0.1 will be installed in the following location:

C:\WESB

with the following features:

Sample Applications Gallery
Public API Javadoc(TM)

for a total size:

1194.5 MB

The profile that you are creating has the following characteristics:

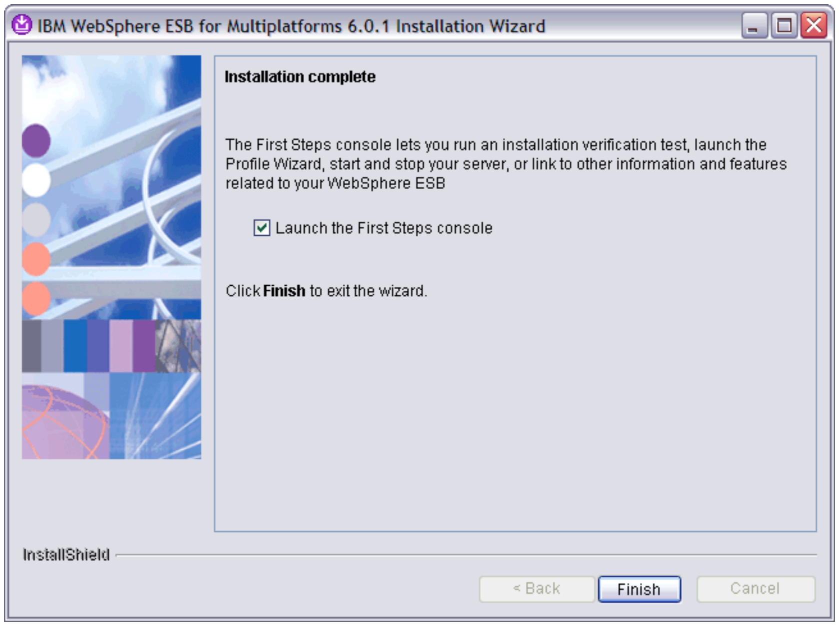
Type - Stand-alone WebSphere ESB profile
Location - C:\WESB\profiles\default

IBM WebSphere Application Server Network Deployment, Version 6.0.2.3 will be installed in the following location:

C:\WESB

Figure 5-17 Installation Wizard - summary

10. Review the results of the installation and click **Finish** (Figure 5-18 on page 106).


IBM WebSphere ESB for Multiplatforms 6.0.1 Installation Wizard
InstallShield
< Back
Finish
Cancel

Installation complete

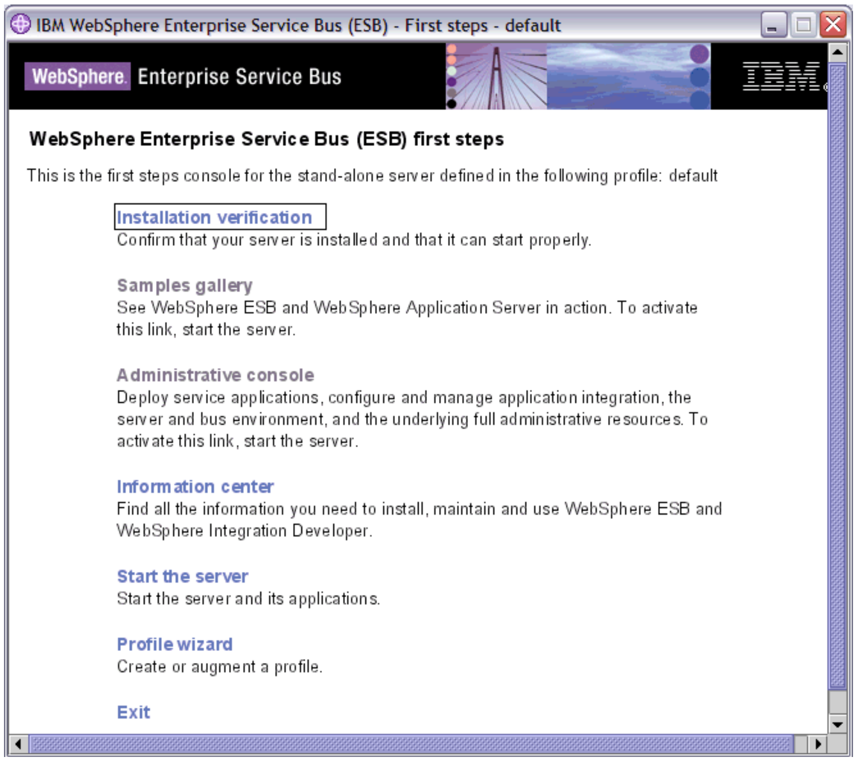
The First Steps console lets you run an installation verification test, launch the Profile Wizard, start and stop your server, or link to other information and features related to your WebSphere ESB

[✓] Launch the First Steps console

Click **Finish** to exit the wizard.

Figure 5-18 Installation status

11. After successful installation, the First Steps menu is displayed, as shown in Figure 5-19 on page 107.


IBM WebSphere Enterprise Service Bus (ESB) - First steps - default

WebSphere. Enterprise Service Bus

WebSphere Enterprise Service Bus (ESB) first steps

This is the first steps console for the stand-alone server defined in the following profile: default

Installation verification
Confirm that your server is installed and that it can start properly.

Samples gallery
See WebSphere ESB and WebSphere Application Server in action. To activate this link, start the server.

Administrative console
Deploy service applications, configure and manage application integration, the server and bus environment, and the underlying full administrative resources. To activate this link, start the server.

Information center
Find all the information you need to install, maintain and use WebSphere ESB and WebSphere Integration Developer.

Start the server
Start the server and its applications.

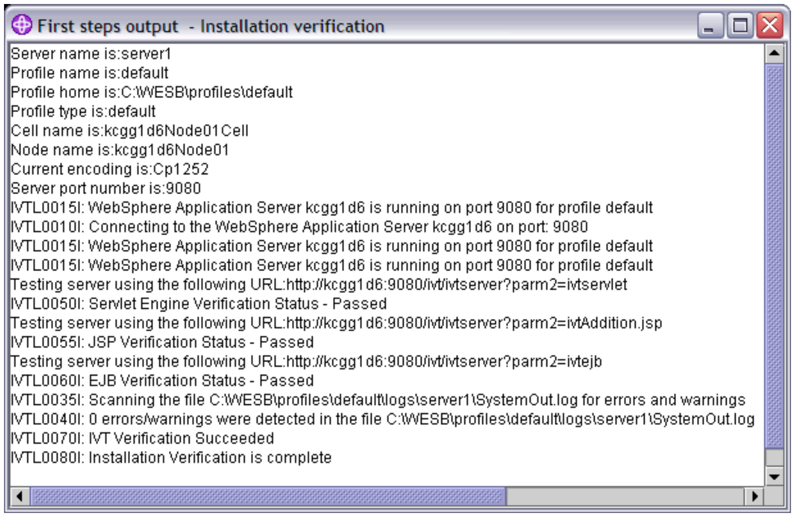
Profile wizard
Create or augment a profile.

Exit

Figure 5-19 First Steps menu

12. At this point, you should select the **Start the server** option in the First Steps dialog. This option will cause a command window to open and you will be able to see the status of the processing. If the window goes away, the server has been successfully started. If the window persists, it will show any errors associated with starting the server.

13. Next, click the **Installation verification** option on the First Steps dialog to run the test. We did that and received the results shown in Figure 5-20 on page 108.



```
Server name is:server1
Profile name is:default
Profile home is:C:\WESB\profiles\default
Profile type is:default
Cell name is:kcgg1d6Node01Cell
Node name is:kcgg1d6Node01
Current encoding is:Cp1252
Server port number is:9080
IVTL0015I: WebSphere Application Server kcgg1d6 is running on port 9080 for profile default
IVTL0010I: Connecting to the WebSphere Application Server kcgg1d6 on port: 9080
IVTL0015I: WebSphere Application Server kcgg1d6 is running on port 9080 for profile default
IVTL0015I: WebSphere Application Server kcgg1d6 is running on port 9080 for profile default
Testing server using the following URL:http://kcgg1d6:9080/ivt/ivtserver?parm2=ivtservlet
IVTL0050I: Servlet Engine Verification Status - Passed
Testing server using the following URL:http://kcgg1d6:9080/ivt/ivtserver?parm2=ivtAddition.jsp
IVTL0055I: JSP Verification Status - Passed
Testing server using the following URL:http://kcgg1d6:9080/ivt/ivtserver?parm2=ivtejb
IVTL0060I: EJB Verification Status - Passed
IVTL0035I: Scanning the file C:\WESB\profiles\default\logs\server1\SystemOut.log for errors and warnings
IVTL0040I: 0 errors/warnings were detected in the file C:\WESB\profiles\default\logs\server1\SystemOut.log
IVTL0070I: IVT Verification Succeeded
IVTL0080I: Installation Verification is complete
```

Figure 5-20 Results of successful IVT

These tests verify the basic functioning of the Application Server. If you want to see WebSphere Enterprise Service Bus in action, launch the Samples Gallery and invoke the sample Stock Quote application. This will further verify your installation of WebSphere Enterprise Service Bus.

For our Windows environment, after we closed the First Steps dialog, we used the Start menu options to control the operation of the runtime environment. To start and stop the default stand-alone server we used **Start** -> **All Programs -> IBM WebSphere** -> **Enterprise Service 6.0** -> **Profiles** -> **default** as shown in Figure 5-21 on page 108.

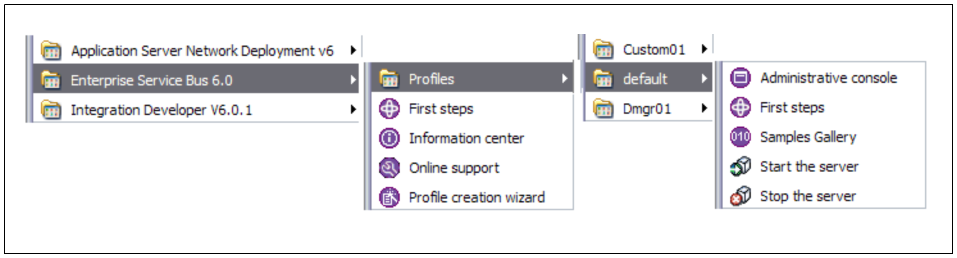


Figure 5-21 Windows Start menu options

5.5.2 A common development integration test runtime environment

In this section we describe a runtime environment that uses the Network Deployment topology. This configuration could support multiple developers doing integration test activities. Since we configure a cell with multiple servers, common application packaging and deployment scenarios can be used. For example, it is good practice to deploy Web service providers in a server separate from Web service consumers, and this configuration would support that, while still providing a single point of administrative control.

We have already done the installation steps described in the previous section, and we are going to configure our cell on a single machine, so we only have customization and configuration tasks to complete. We proceeded through the following sequence of steps:

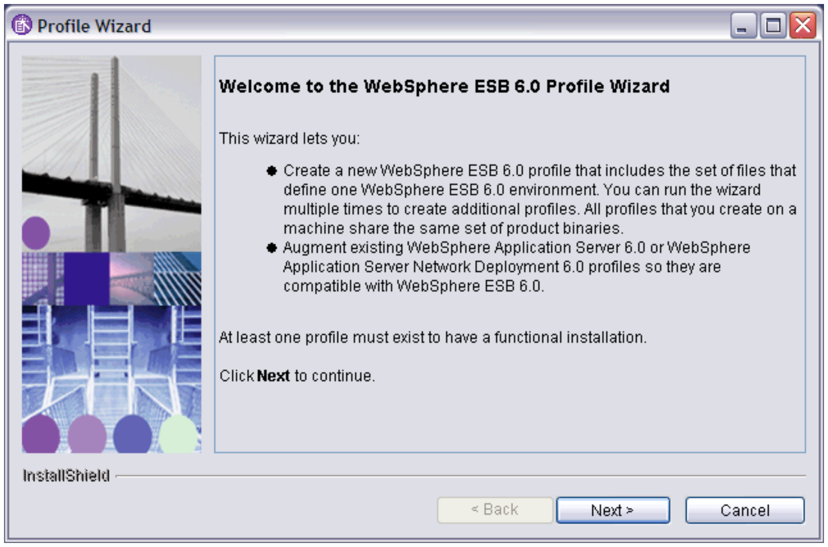
1. Create a new WebSphere Enterprise Service Bus deployment manager profile.
2. Start the deployment manager server.
3. Create a new WebSphere Enterprise Service Bus custom profile.
4. Federate the custom node to the deployment manager.
5. Add a server to the new managed node.
6. Configure the SCA service integration buses in the new server.
7. Configure the CEI service in the new server.

In order to customize WebSphere Enterprise Service Bus for a Network Deployment topology, follow these steps, which are described in more detail below.

Creating a Deployment Manager profile

To create a new Deployment Manager profile, perform the following:

1. Launch the Profile Creation Wizard, which is shown in Figure 5-22 on page 110.



Welcome to the WebSphere ESB 6.0 Profile Wizard

This wizard lets you:

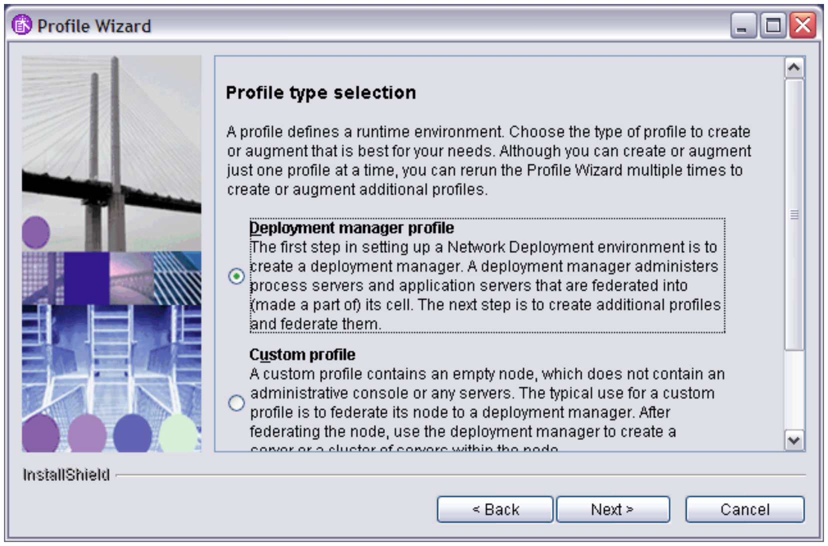
- Create a new WebSphere ESB 6.0 profile that includes the set of files that define one WebSphere ESB 6.0 environment. You can run the wizard multiple times to create additional profiles. All profiles that you create on a machine share the same set of product binaries.
- Augment existing WebSphere Application Server 6.0 or WebSphere Application Server Network Deployment 6.0 profiles so they are compatible with WebSphere ESB 6.0.

At least one profile must exist to have a functional installation.

Click **Next** to continue.

Figure 5-22 Profile creation wizard

2. Click **Next** to continue, select **Deployment manager profile** and click **Next** (Figure 5-23 on page 110).



Profile type selection

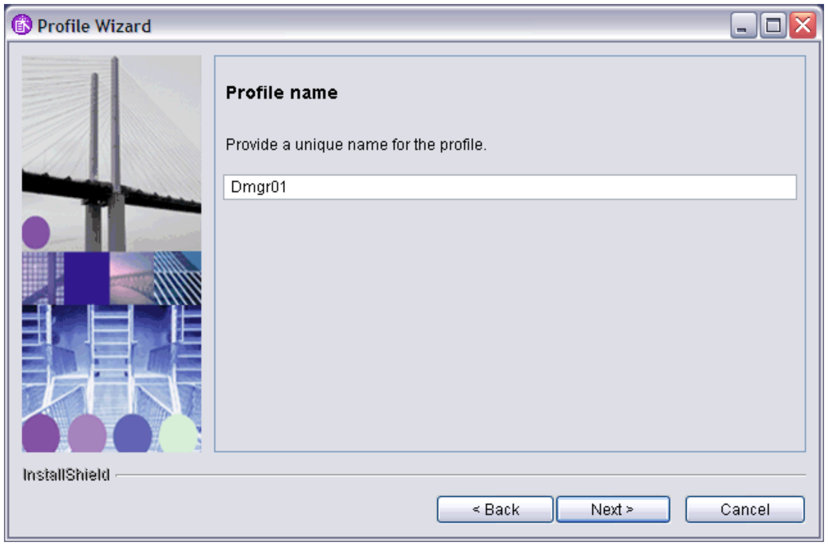
A profile defines a runtime environment. Choose the type of profile to create or augment that is best for your needs. Although you can create or augment just one profile at a time, you can rerun the Profile Wizard multiple times to create or augment additional profiles.

[✓ **Deployment manager profile**]
The first step in setting up a Network Deployment environment is to create a deployment manager. A deployment manager administers process servers and application servers that are federated into (made a part of) its cell. The next step is to create additional profiles and federate them.

Custom profile
A custom profile contains an empty node, which does not contain an administrative console or any servers. The typical use for a custom profile is to federate its node to a deployment manager. After federating the node, use the deployment manager to create a

Figure 5-23 Profile creation wizard - type selection

3. Enter a name for the profile. We used the default name for a Deployment Manager profile. (Figure 5-24 on page 111). Click **Next**.



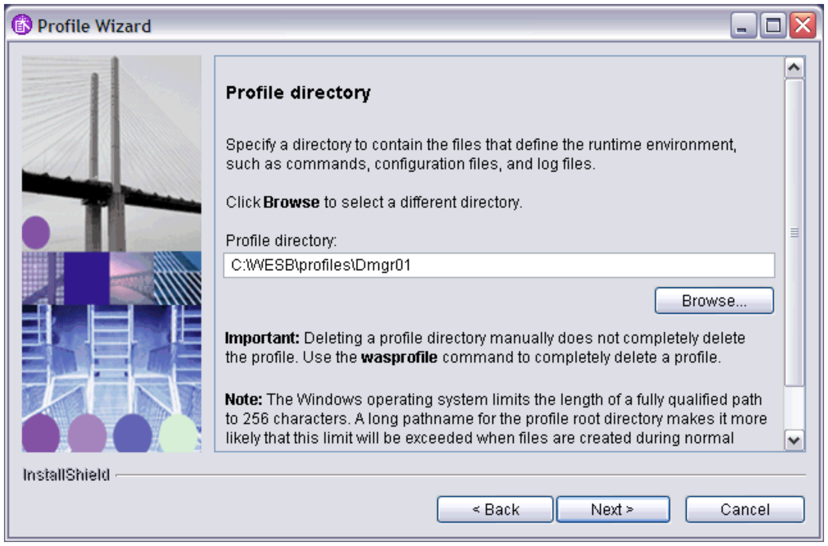
Profile name

Provide a unique name for the profile.

Dmgr01

Figure 5-24 Profile creation wizard- deployment manager name

4. Enter the target directory for the profile (Figure 5-25 on page 111). Click **Next**.



Profile directory

Specify a directory to contain the files that define the runtime environment, such as commands, configuration files, and log files.

Click **Browse** to select a different directory.

Profile directory:

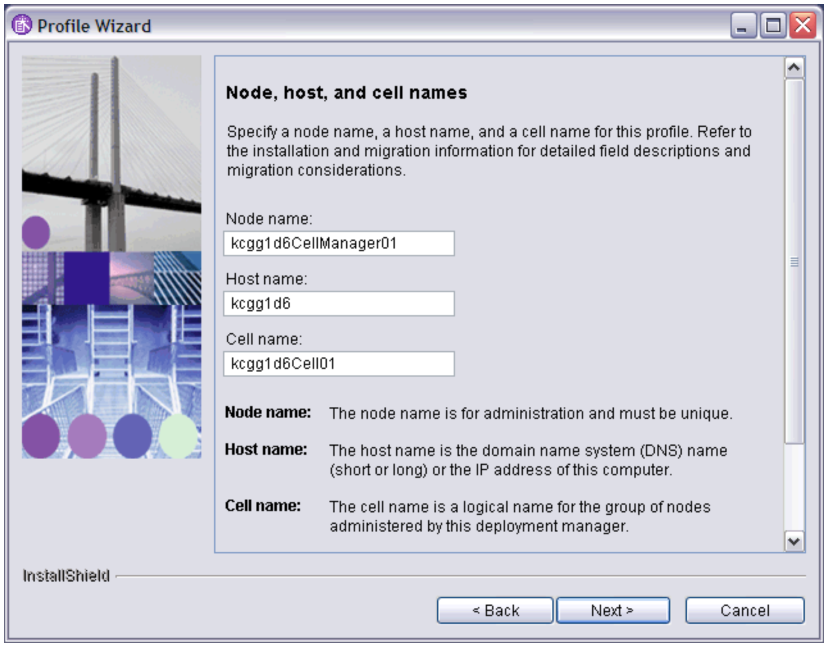
C:\WESB\profiles\Dmgr01

Important: Deleting a profile directory manually does not completely delete the profile. Use the **wasprofile** command to completely delete a profile.

Note: The Windows operating system limits the length of a fully qualified path to 256 characters. A long pathname for the profile root directory makes it more likely that this limit will be exceeded when files are created during normal

Figure 5-25 Profile creation wizard - directory name

5. Take the default values for the Host, Node and Cell names, as shown in Figure 5-26 on page 112, or modify them to meet your requirements. Click **Next**.



Node, host, and cell names

Specify a node name, a host name, and a cell name for this profile. Refer to the installation and migration information for detailed field descriptions and migration considerations.

Node name: kcgg1d6CellManager01

Host name: kcgg1d6

Cell name: kcgg1d6Cell01

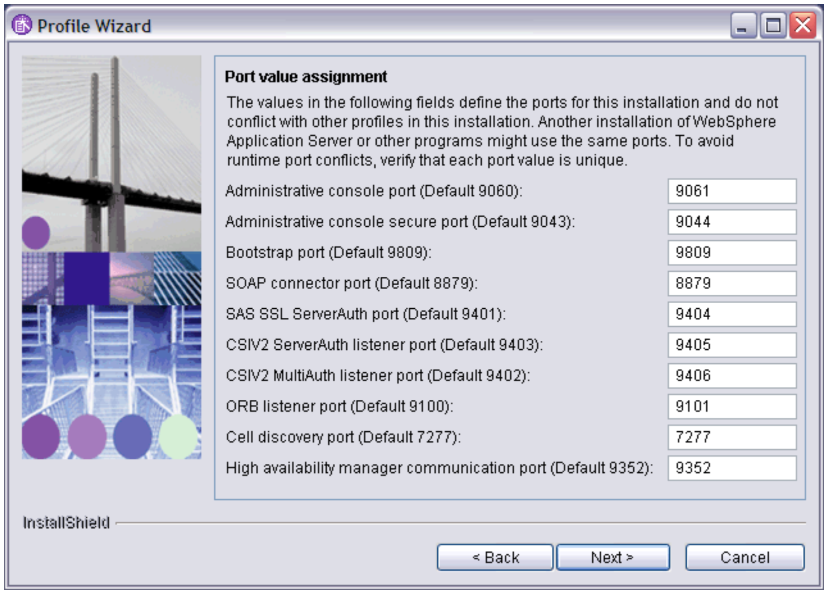
Node name: The node name is for administration and must be unique.

Host name: The host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

Cell name: The cell name is a logical name for the group of nodes administered by this deployment manager.

Figure 5-26 Profile creation wizard - node, host and cell names

6. Review the port value assignments. We let these default, as shown in Figure 5-27 on page 113. When you are satisfied with the port value assignments, click **Next**.



Port value assignment

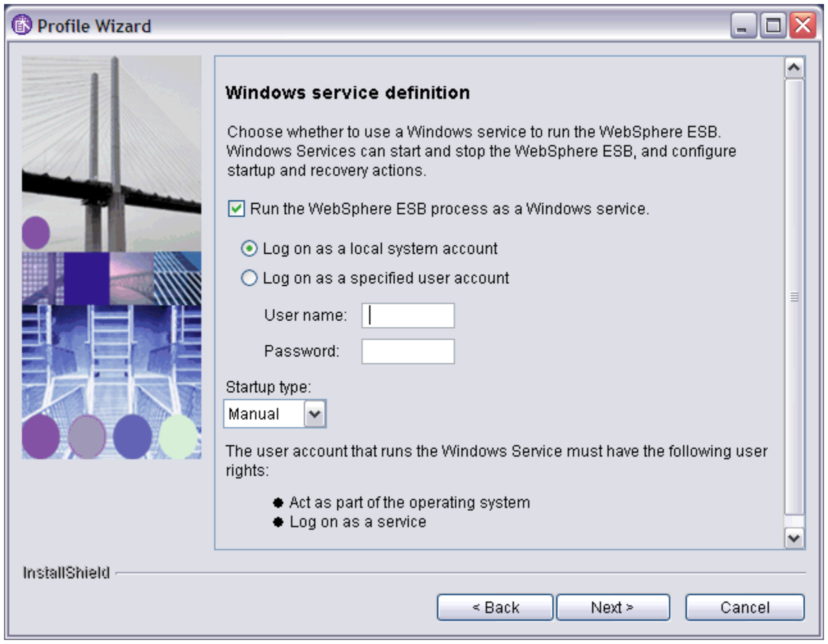
The values in the following fields define the ports for this installation and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid runtime port conflicts, verify that each port value is unique.

Administrative console port (Default 9060):	9061
Administrative console secure port (Default 9043):	9044
Bootstrap port (Default 9809):	9809
SOAP connector port (Default 8879):	8879
SAS SSL ServerAuth port (Default 9401):	9404
CSIV2 ServerAuth listener port (Default 9403):	9405
CSIV2 MultiAuth listener port (Default 9402):	9406
ORB listener port (Default 9100):	9101
Cell discovery port (Default 7277):	7277
High availability manager communication port (Default 9352):	9352

Figure 5-27 Profile creation wizard - port value assignment

Note: Notice the port values we used here are not the standard values. In our case that is due to the fact that we already have a default profile for the stand-alone server on this machine. We could have modified them to the standard if we never intend to have the stand-alone server and deployment manager server running at the same time on this machine.

7. Select how to run the Deployment Manager server. We chose to manually start the server as a Windows process. Our selections are shown in Figure 5-28 on page 114. Click **Next**.



Windows service definition

Choose whether to use a Windows service to run the WebSphere ESB. Windows Services can start and stop the WebSphere ESB, and configure startup and recovery actions.

[✓] Run the WebSphere ESB process as a Windows service.

[✓ Log on as a local system account]
Log on as a specified user account

User name:
Password:

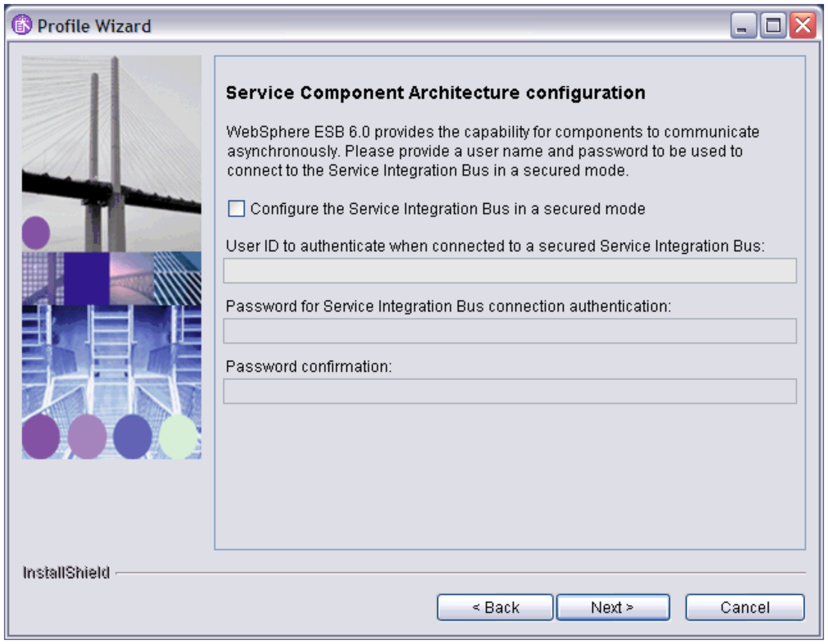
Startup type: Manual

The user account that runs the Windows Service must have the following user rights:

- Act as part of the operating system
- Log on as a service

Figure 5-28 Profile creation wizard - Windows service definition

8. Chose whether to run the service integration bus in secured mode. We chose to leave this unsecured, as shown in Figure 5-29 on page 115. Click **Next**.



Service Component Architecture configuration

WebSphere ESB 6.0 provides the capability for components to communicate asynchronously. Please provide a user name and password to be used to connect to the Service Integration Bus in a secured mode.

☐ Configure the Service Integration Bus in a secured mode

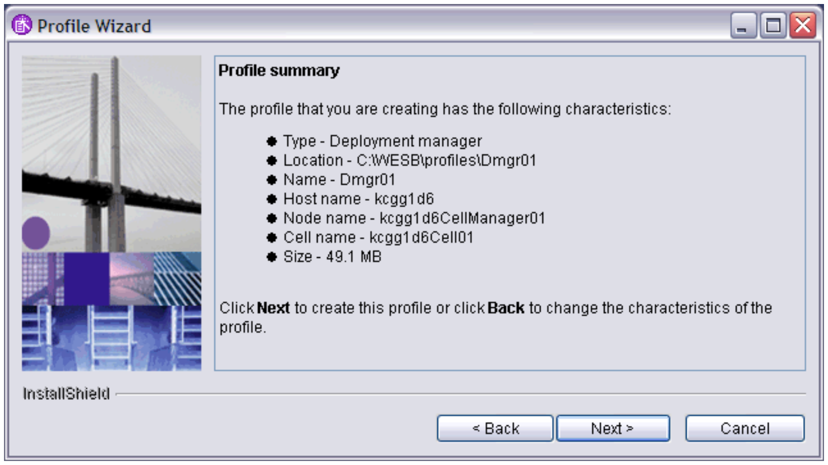
User ID to authenticate when connected to a secured Service Integration Bus:

Password for Service Integration Bus connection authentication:

Password confirmation:

Figure 5-29 Profile creation wizard - system integration bus security

9. Review the Profile summary(Figure 5-30 on page 115). When you click **Next**, the new Deployment Manager profile is created.



Profile summary

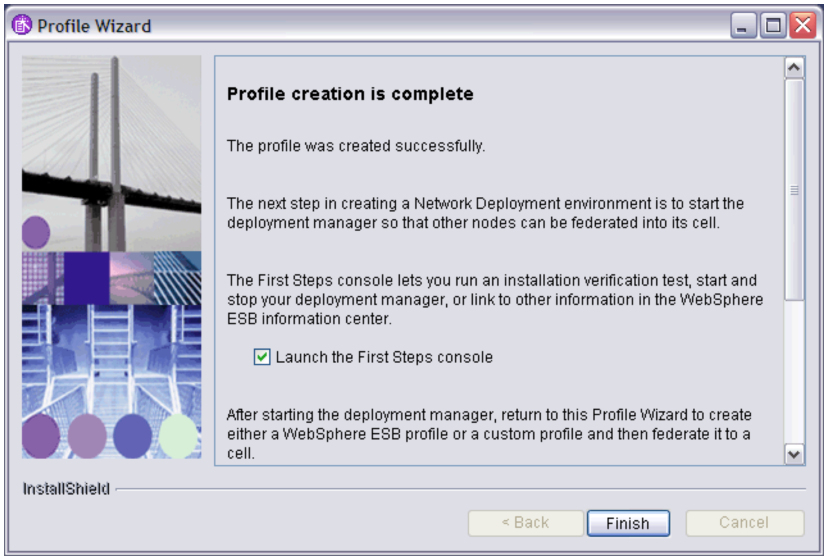
The profile that you are creating has the following characteristics:

- Type - Deployment manager
- Location - C:\WESB\profiles\Dmgr01
- Name - Dmgr01
- Host name - kcgg1d6
- Node name - kcgg1d6CellManager01
- Cell name - kcgg1d6Cell01
- Size - 49.1 MB

Click **Next** to create this profile or click **Back** to change the characteristics of the profile.

Figure 5-30 Profile creation wizard - summary

10. When the wizard completes, it displays the results of the processing, as shown in Figure 5-31 on page 116.



Profile creation is complete

The profile was created successfully.

The next step in creating a Network Deployment environment is to start the deployment manager so that other nodes can be federated into its cell.

The First Steps console lets you run an installation verification test, start and stop your deployment manager, or link to other information in the WebSphere ESB information center.

[✓] Launch the First Steps console

After starting the deployment manager, return to this Profile Wizard to create either a WebSphere ESB profile or a custom profile and then federate it to a cell.

Figure 5-31 Profile creation wizard - results

11. Start the Deployment Manager server.

12. Start the administrative console and verify that you can log on. Log off and close the browser. You can leave the Deployment Manager server running.

13. For our Windows environment we used the Start menu options to control the operation of the runtime environment. We used **Start** -> **All Programs** -> **IBM WebSphere** -> **Enterprise Service 6.0 -> Profiles**-> **Dmgr01** to start and stop the deployment manager, and to open the administration console.

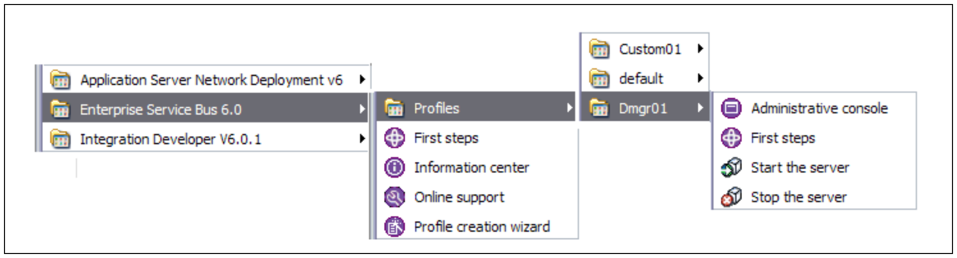
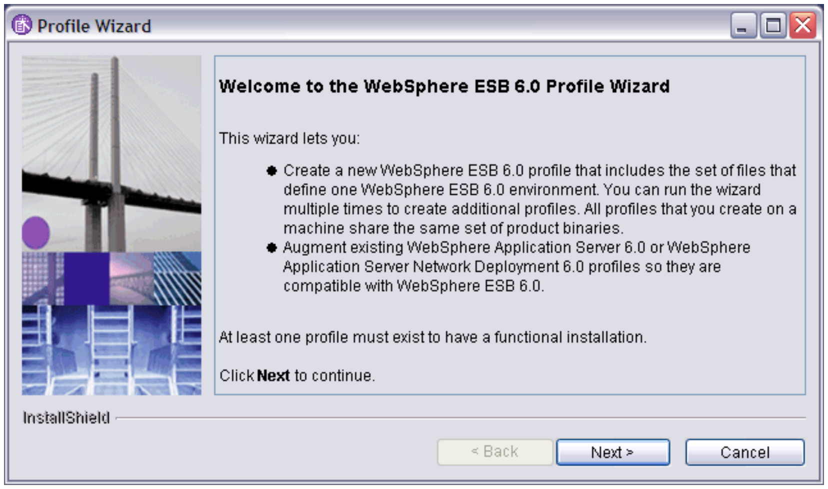


Figure 5-32 Windows Start menu options

Creating a Custom node profile

Once we have a Deployment Manager server configured and started we can create a profile representing a managed node which we will add to the cell. To do this, perform the following:

1. Start the Profile Creation wizard in order to create the Custom profile (Figure 5-33 on page 117). Click **Next**.



Profile Wizard

Welcome to the WebSphere ESB 6.0 Profile Wizard

This wizard lets you:

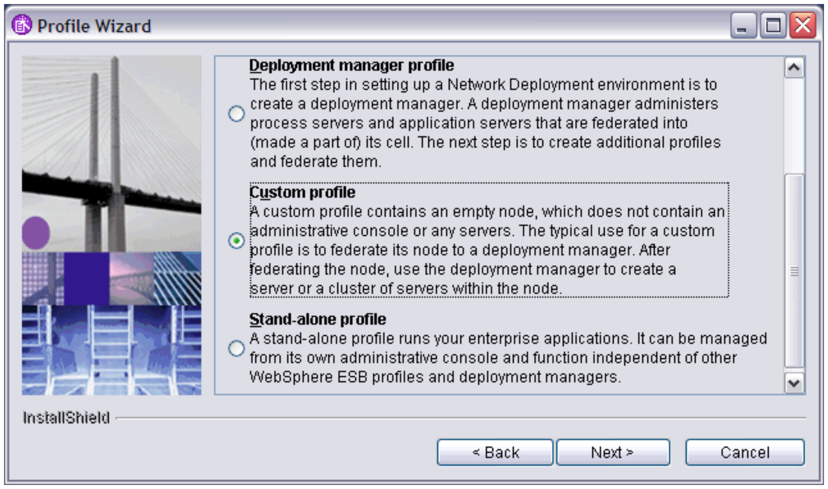
- Create a new WebSphere ESB 6.0 profile that includes the set of files that define one WebSphere ESB 6.0 environment. You can run the wizard multiple times to create additional profiles. All profiles that you create on a machine share the same set of product binaries.
- Augment existing WebSphere Application Server 6.0 or WebSphere Application Server Network Deployment 6.0 profiles so they are compatible with WebSphere ESB 6.0.

At least one profile must exist to have a functional installation.

Click **Next** to continue.

Figure 5-33 Profile creation wizard

2. Select **Custom profile** (Figure 5-34 on page 117) and click **Next**.



Profile Wizard

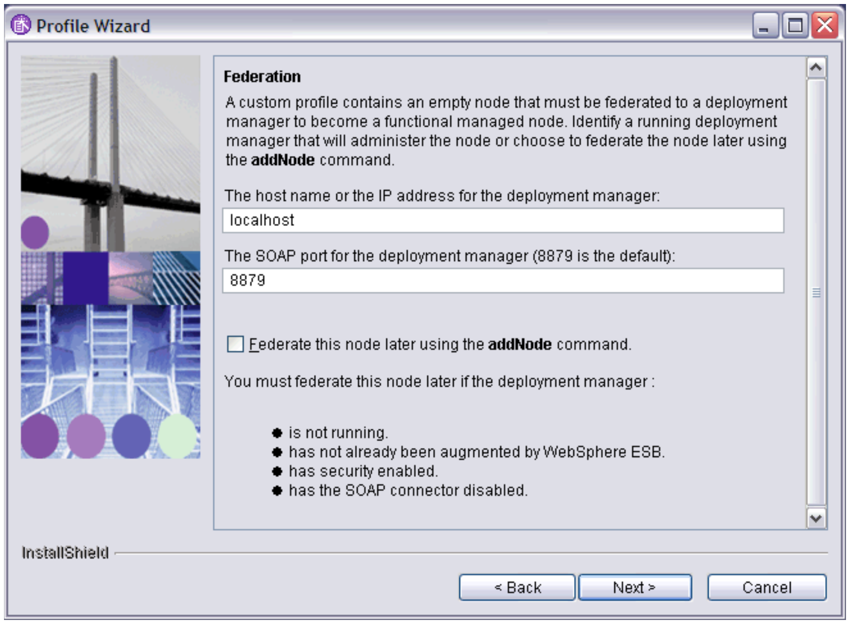
Deployment manager profile
The first step in setting up a Network Deployment environment is to create a deployment manager. A deployment manager administers process servers and application servers that are federated into (made a part of) its cell. The next step is to create additional profiles and federate them.

Custom profile
A custom profile contains an empty node, which does not contain an administrative console or any servers. The typical use for a custom profile is to federate its node to a deployment manager. After federating the node, use the deployment manager to create a server or a cluster of servers within the node.

Stand-alone profile
A stand-alone profile runs your enterprise applications. It can be managed from its own administrative console and function independent of other WebSphere ESB profiles and deployment managers.

Figure 5-34 Profile creation wizard - type selection

3. You can generally take the defaults for hostname and SOAP port, as we did in Figure 5-35 on page 118. Notice also, that we have left the checkbox **Federate this node later using the addNode command** unchecked. This instructs the wizard to federate the new managed node into our deployment manager cell. Click **Next** after you make any updates.



Federation

A custom profile contains an empty node that must be federated to a deployment manager to become a functional managed node. Identify a running deployment manager that will administer the node or choose to federate the node later using the **addNode** command.

The host name or the IP address for the deployment manager: localhost

The SOAP port for the deployment manager (8879 is the default): 8879

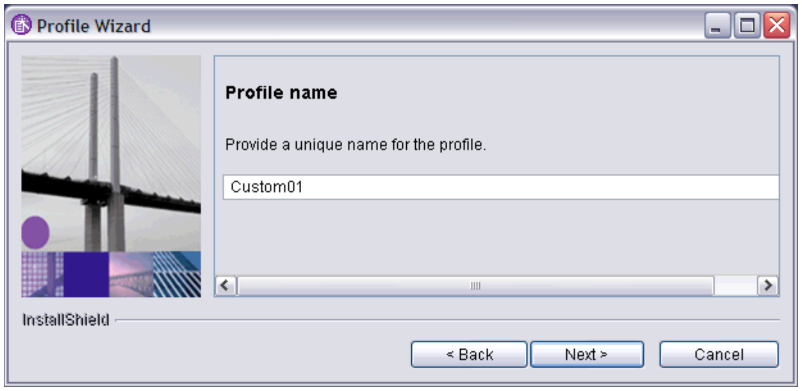
☐ Federate this node later using the **addNode** command.

You must federate this node later if the deployment manager :

- is not running.
- has not already been augmented by WebSphere ESB.
- has security enabled.
- has the SOAP connector disabled.

Figure 5-35 Profile creation wizard - federation

4. Enter a name for the profile. We took the default (Figure 5-36 on page 118), and clicked **Next**.



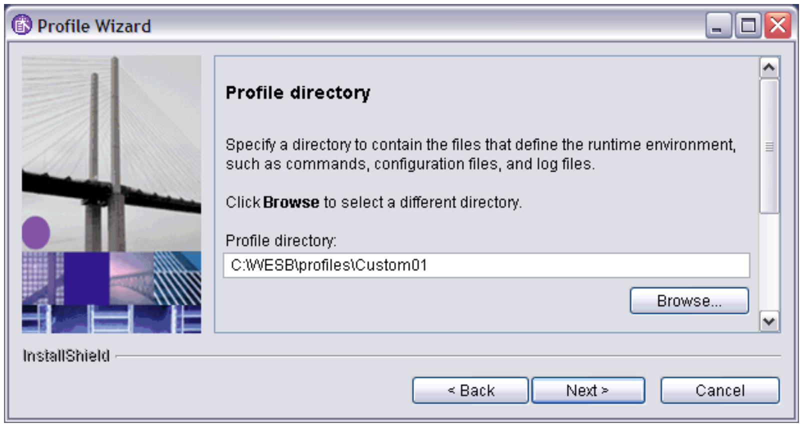
Profile name

Provide a unique name for the profile.

Custom01

Figure 5-36 Profile creation wizard - profile name

5. Chose a directory for the generated profile (Figure 5-37 on page 119), and click **Next**.



Profile directory

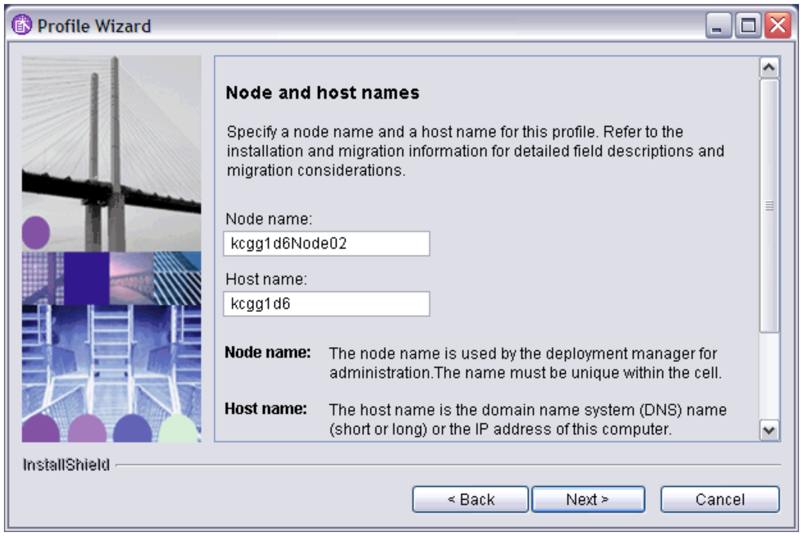
Specify a directory to contain the files that define the runtime environment, such as commands, configuration files, and log files.

Click **Browse** to select a different directory.

Profile directory:
C:\WESB\profiles\Custom01

Figure 5-37 Profile creation wizard - directory name

6. Enter the host and node names. We used the defaults, as seen in Figure 5-38 on page 119. Click **Next** after you make any desired updates.



Node and host names

Specify a node name and a host name for this profile. Refer to the installation and migration information for detailed field descriptions and migration considerations.

Node name:
kcgg1d6Node02

Host name:
kcgg1d6

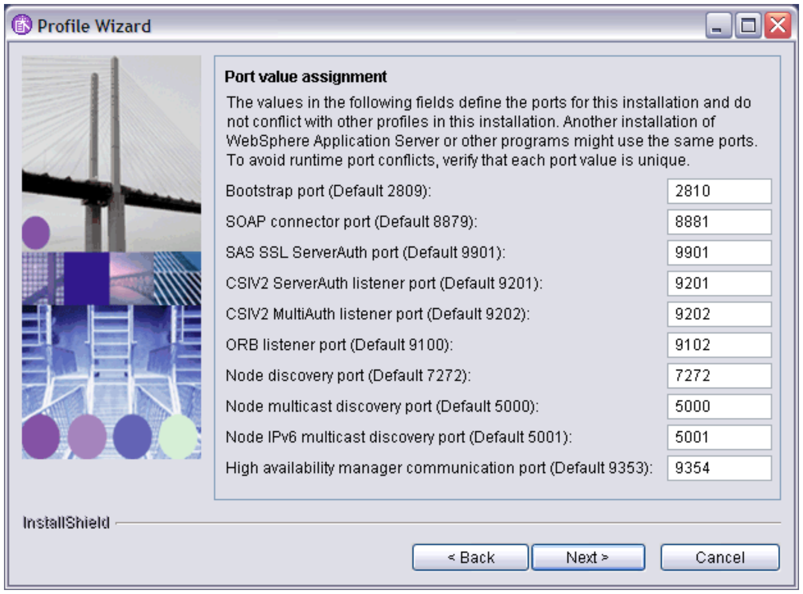
Node name: The node name is used by the deployment manager for administration.The name must be unique within the cell.

Host name: The host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

Figure 5-38 Profile creation wizard - node and host names

7. Click **Next** and review the port assignments. Notice that we accepted the port assignments generated by the wizard, although they are not the default values (Figure 5-39 on page 120). Since we have a stand-alone server

already configured on this machine, the wizard generated port values that would not conflict with that configuration. Click **Next**.



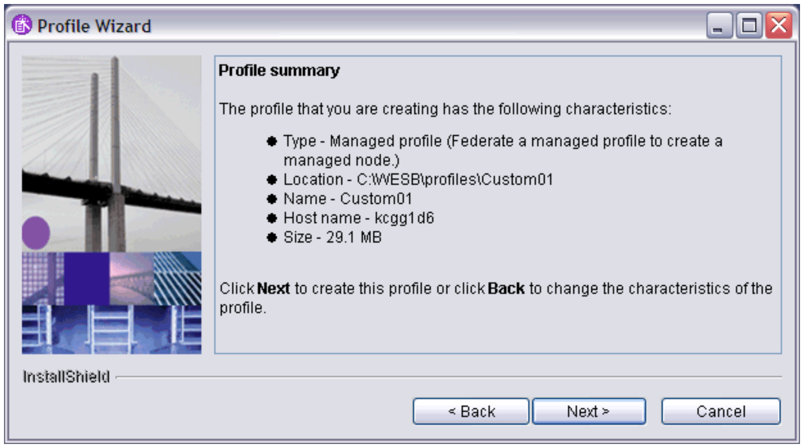
Port value assignment

The values in the following fields define the ports for this installation and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid runtime port conflicts, verify that each port value is unique.

Bootstrap port (Default 2809):	2810
SOAP connector port (Default 8879):	8881
SAS SSL ServerAuth port (Default 9901):	9901
CSIV2 ServerAuth listener port (Default 9201):	9201
CSIV2 MultiAuth listener port (Default 9202):	9202
ORB listener port (Default 9100):	9102
Node discovery port (Default 7272):	7272
Node multicast discovery port (Default 5000):	5000
Node IPv6 multicast discovery port (Default 5001):	5001
High availability manager communication port (Default 9353):	9354

Figure 5-39 Profile creation wizard - port value assignments

8. Review the summary information for accuracy. When you click **Next**, the profile is created (Figure 5-40 on page 120).



Profile summary

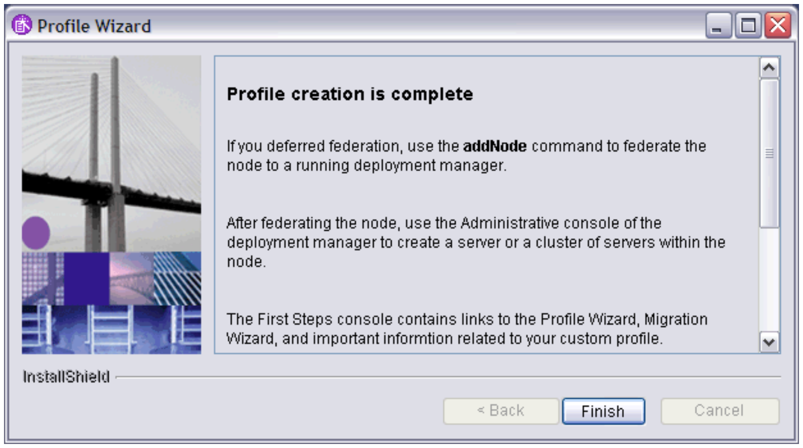
The profile that you are creating has the following characteristics:

- Type - Managed profile (Federate a managed profile to create a managed node.)
- Location - C:\WESB\profiles\Custom01
- Name - Custom01
- Host name - kcgg1d6
- Size - 29.1 MB

Click **Next** to create this profile or click **Back** to change the characteristics of the profile.

Figure 5-40 Profile creation wizard - summary

9. The final pane of the dialog reports the results of the profile creation, as shown in Figure 5-41 on page 121. Click **Finish** when your review is complete.



Profile creation is complete

If you deferred federation, use the **addNode** command to federate the node to a running deployment manager.

After federating the node, use the Administrative console of the deployment manager to create a server or a cluster of servers within the node.

The First Steps console contains links to the Profile Wizard, Migration Wizard, and important informtion related to your custom profile.

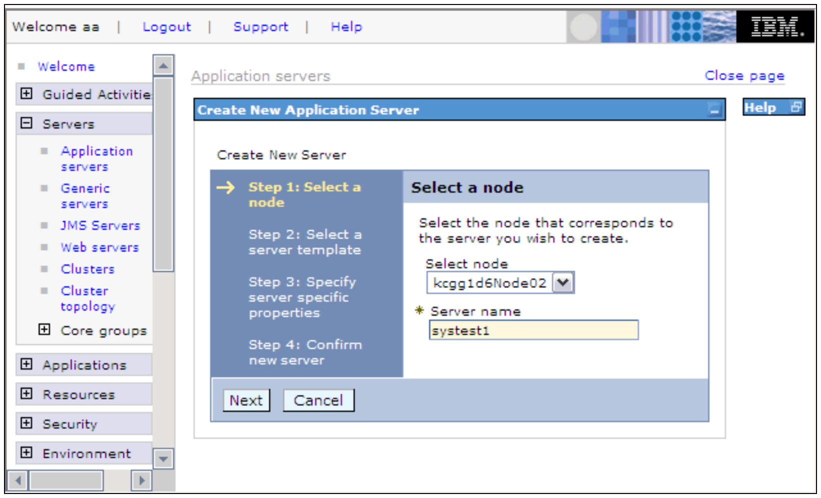
Figure 5-41 Profile creation wizard - results

10. Now that you have the custom profile created, you should start the node agent for the managed node. In the Windows Explorer, browse to the directory where you created the profile. From there, open the bin directory and you will find the **`startNode.bat`** command file. Run the command to start the node agent.

Creating a new server

With the Deployment Manager started and the node agent for the Custom node started, you can now add a server to the topology, which will be able to host your mediation modules. To do this, perform the following:

1. Log into the administrative console.
2. Create a new server by selecting **Servers** -> **New** and follow the steps in the dialog. Enter a name for the server (Figure 5-42 on page 122). We named ours `systest1`. Click **Next**.



Create New Application Server

Create New Server

Step 1: Select a node
Step 2: Select a server template
Step 3: Specify server specific properties
Step 4: Confirm new server

Select a node

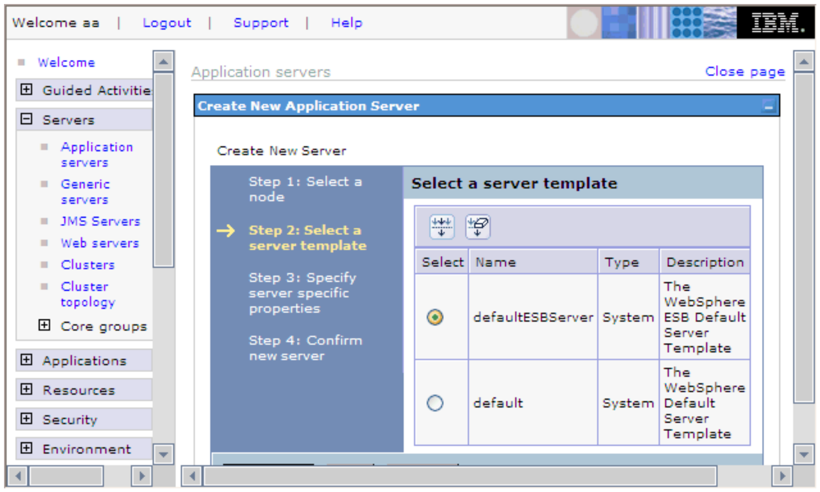
Select the node that corresponds to the server you wish to create.

Select node: kcgg1d6Node02

* Server name: systest1

Figure 5-42 Create server - server name

3. Select the **defaultESBServer** template, as shown in Figure 5-43 on page 122, and click **Next**.



Create New Application Server

Create New Server

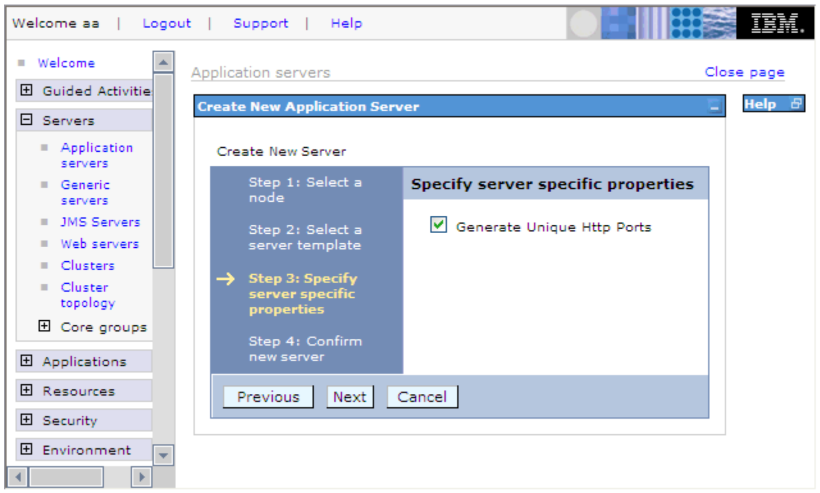
Step 1: Select a node
Step 2: Select a server template
Step 3: Specify server specific properties
Step 4: Confirm new server

Select a server template

Select	Name	Type	Description
●	defaultESBServer	System	The WebSphere ESB Default Server Template
○	default	System	The WebSphere Default Server Template

Figure 5-43 Create server - server template

4. Make sure the check box to generate unique HTTP ports is checked (Figure 5-44 on page 123) and click **Next**.



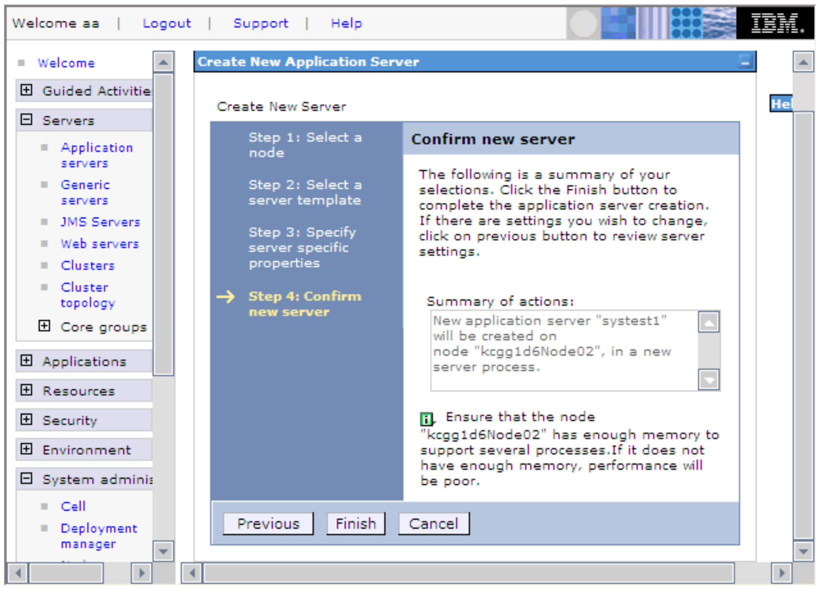
Create New Server

Specify server specific properties

✓ Generate Unique Http Ports

Figure 5-44 Create server - properties

5. Review the summary information (Figure 5-45 on page 123) and click **Finish** when you have confirmed your entries.



Create New Server

Confirm new server

The following is a summary of your selections. Click the Finish button to complete the application server creation. If there are settings you wish to change, click on previous button to review server settings.

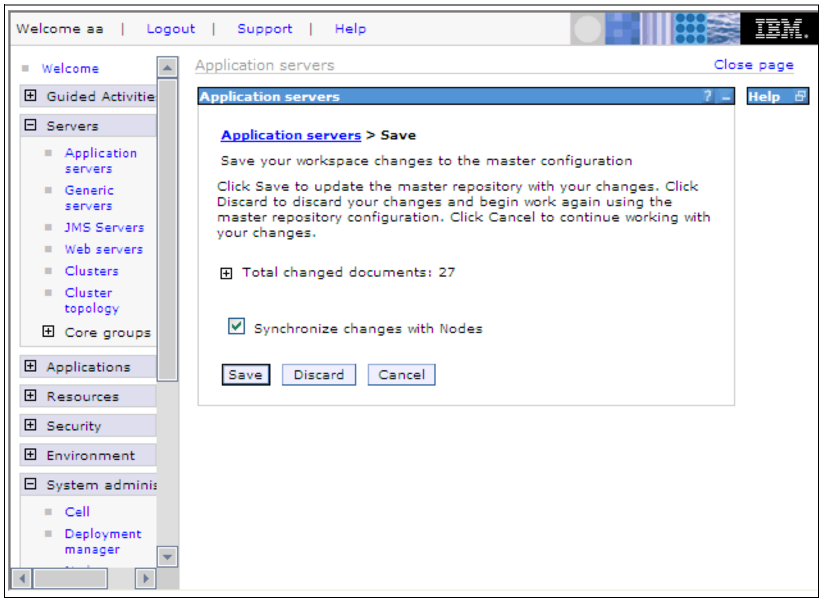
Summary of actions:

New application server "systest1" will be created on node "kcgg1d6Node02", in a new server process.

Ensure that the node "kcgg1d6Node02" has enough memory to support several processes.If it does not have enough memory, performance will be poor.

Figure 5-45 Create server - confirmation

6. Click **Save changes** and make sure the check box to **Synchronize changes with Nodes** is checked, as in Figure 5-46 on page 124. This tells the deployment manager to update the configuration of the managed node by interacting with the node agent.



Application servers > Save

Save your workspace changes to the master configuration

Click Save to update the master repository with your changes. Click Discard to discard your changes and begin work again using the master repository configuration. Click Cancel to continue working with your changes.

Total changed documents: 27

[✓] Synchronize changes with Nodes

Figure 5-46 Create server - save changes to master

You now have a managed node with an application server. You can use the administrative console to start and stop the server. If you intend to deploy mediation modules to the server, you will need to perform the final two configuration steps discussed in 5.4.3, "Final configuration steps" on page 98.

5.6 Guidelines for staged test and production environments

Planning for multiple staged test environments for WebSphere Enterprise Service Bus is very much like doing the same activity for WebSphere Application Server. You will want to establish numerous test environments to meet various requirements of the test cycle. If you have a set of existing WebSphere Application Server test environments, then it is very likely those can simply be extended to provide similar test stages for WebSphere Enterprise Service Bus. If those environments do not exist today, then you will want to consider the factors discussed in the following sections as you plan for multiple test stages.

5.6.1 Development integration test environment

The requirement for a development integration test environment is essentially to extend what each developer has on their desktop inside of WebSphere Integration Developer to a formal test stage allowing all of the components in a solution to be deployed and tested together. With the Unit Test Environment (UTE) inside of WebSphere Integration Developer developers can fully unit test all of the code they have developed, but will often emulate other components in the solution for expediency. The development integration test environment should be one that facilitates the task of testing the integrated components, and adds very little additional complexity to the testing. The following guidelines should be evaluated to meet your specific test requirements at this stage:

- A stand-alone server topology should meet the needs of this test stage.
- If possible, attempt to minimize the security requirements on your application during this phase. Certainly do not add additional requirements over those included in the UTE.
- Message persistence continues to be a quality of service that is frequently not critical to the test results in this phase. You can use the same configuration as was used in the UTE, which most commonly is Cloudscape.
- The behavior of the Common Event Infrastructure (CEI) can continue to be the same as it was in the UTE. Again, you can continue to use Cloudscape for persisting CEI events.
- If you are using the message logger mediation primitive, then Cloudscape should continue to provide the functions required in this test stage.
- You may want to allow developers to attach the integration debugger remotely to this environment to perform problem determination and to isolate code problems.
- Be sure the maintenance level of WebSphere Enterprise Service Bus is at least as high as that used in development and take steps to keep service at a very high level in both environments.
- Typically, the developers will be more than willing to perform administration responsibilities for this environment.

The value of this test stage is to the developers, as it affords them a test environment very similar to the UTE where they can extend their test scope to the entire application. Moreover, as they perform (possibly repeated) configuration updates to the server, they will begin to think about automating aspects of server configuration and application deployment.

5.6.2 System test environment

Moving to a system test environment begins to introduce many of the complexities of a real production environment, with perhaps the exception of scale. The application is now tested at a functional level and all of its components have been tested in the packaging scheme that is likely to exist in a production environment. So the requirements of this test stage are more those of moving the runtime toward a real production environment. The testing will include a higher load than was applied in the prior test phase, and many of the test cases dealing with security, failure and recovery may only be attempted now for the first time.

Consider the following:

- A Network Deployment topology will most likely be required since the need to administer the cell from a single point of control will probably be a high priority. Also, the throughput requirements may be such that a single messaging engine may not handle the load.
- Global security should be turned on for the cell to control access to the administrative console as well as to perform test cases dealing with authentication and authorized access to parts of the application.
- Message persistence should be configured to use a relational data store, such as DB2® Universal Database™. Also, the default settings for persistence may need to be modified to represent production requirements. Losing messages in a production environment is rarely acceptable.
- If the CEI events are to be enabled, then a relational database should be configured for these as well.
- There will be a need to run various utilities against the newly defined databases in this environment. For example, clean-up utilities may need to be run against the CEI database. Scripts are provided for DB2 Universal Database V8.1 and V8.2.1 which invoke the **`runstats`** and **`reorg`** DB2 utilities. Use **`runstats`** to update the database statistics after a large number of records have been purged from the database, or inserted into it. Also, after using the **`reorg`** script or adding/removing indexes from a table, **`runstats`** should be executed. The scripts are found in the *install_root*/event/dbscripts/db2 directory.
- Configure a relational database datasource if your mediation modules are using the logger primitive.
- Consider using automated scripts to configure the servers and deploy the applications into this test stage to make sure they will support the production environment.
- Use the problem determination facilities of your application and the WebSphere Enterprise Service Bus runtime to isolate problems.

- Consider modifying the configuration of the queue points and activation specification options to enable them to sustain higher loads.
- Consider configuring additional bus topologies, for example a foreign bus or MQLink to an existing WebSphere MQ network. A service integration bus must be wholly contained in a single cell. You can, however have more than one bus in a cell and you can connect buses with a foreign link whether they are in the same cell or different cells.
- The maintenance level of WebSphere Enterprise Service Bus should be the same as the maintenance level used to exit development integration testing, although service may not be applied as frequently to this environment

The system test environment is intended to support all of the possible test cases developed for the application.

5.6.3 Quality Assurance (QA) environment

The Quality Assurance environment is often configured to match as closely as possible the actual production environment. With the exception that the databases and back-end systems accessed will still not be those used for production, virtually everything else should be identical. This environment is commonly used for early user testing and frequently used to measure application performance, throughput and end user response time.

Generally this environment is built to scale in the same manner as a production environment, so the size and number of servers configured should match very closely with those in production. The configuration may be enhanced over that used in a system test environment to allow for additional clustering, fail-over, and workload balancing.

The maintenance level of WebSphere Enterprise Service Bus in this environment should match that on which system testing exited. Most often the maintenance level will match that being used in production with the possible exception of some very small windows where service updates are applied here and quick regression testing is done, prior to the service being rolled into production.

5.7 Problem determination for runtime installation and customization

Although the wizards that direct you through initial product installation and customization worked well for us, there may be cases where you encounter problems in your environment.

The online InfoCenter for WebSphere Enterprise Service Bus is an excellent resource for helping to identify the cause of an installation or customization failure. The URL for the InfoCenter is:

```
http://publib.boulder.ibm.com/infocenter/dmndhelp/v6rxmx/index.jsp
```

Under the topic *Installing* is the sub-topic *Troubleshooting installation* which should help you through the problem determination steps.

Generally problem determination will be broken down into three basic areas, as follows:

- Did installation of WebSphere Application Server Network Deployment complete successfully?
- Did installation of WebSphere Enterprise Service Bus complete successfully?
- Did profile creation or augmentation complete successfully?

The tasks you perform to do problem determination will depend on which of these three activities caused the problem.

In general, you should understand that each of the installation and customization steps you perform has associated with it a set of logs that capture the events that occur during that phase. The InfoCenter contains a table that describes each type of log and the kind of information it contains. If you can find the log file that contains the events that were being processed during the portion of installation or customization which failed, you should browse it to see if you can determine the error. The logs, however, are not all that easy to read or interpret, so if you find they are not helpful, you should contact IBM support to help diagnose your problem. The logs will most likely be requested by the support team, so be sure you can find them and that they correspond to the state of your installation which was known to have the error.

As always, paying careful attention to the system prerequisites can help to avoid installation problems, and when provided, always capture error messages that appear at the user interface.

6

WebSphere Enterprise Service Bus key concepts and related technologies

This chapter explains the key concepts of WebSphere Enterprise Service Bus and explores some of the most important related technologies. First we will give a short feature overview of the product and explain the key terms. Then, we will break down the product structure in a top-down fashion. In addition to that, the following complementary technology viewpoints will be discussed:

- The data view (Service Data Objects)
- The external interfaces view (Bindings)
- The system management view (Common Event Infrastructure)
- The deployment view

6.1 Product overview

WebSphere Enterprise Service Bus delivers an Enterprise Service Bus (ESB) infrastructure to enable connecting applications that have standards based interfaces (typically a Web service interface described in a WSDL file). It provides mechanisms to process request and response messages from service consumers and service providers connecting to the ESB.

WebSphere Enterprise Service Bus is the mediation layer that runs on top of the transport layer within WebSphere Application Server. As such, WebSphere Enterprise Service Bus provides prebuilt mediation functions and easy to use tools to enable rapid construction and implementation of an ESB as a value-add on top of WebSphere Application Server.

Figure 6-1 gives an overview of what WebSphere Enterprise Service Bus is - the components in the product, its features and functions associated with the product. If you start in the center of the picture, you can see that WebSphere Enterprise Service Bus is built on top of WebSphere Application Server. WebSphere Enterprise Service Bus leverages WebSphere Application Server Network Deployment qualities of service, with its clustering, failover, scalability, security and a built-in messaging provider.

Along with these qualities, WebSphere Enterprise Service Bus also includes a number of key WebSphere Application Server related features, including UDDI as a service registry, the Web services gateway, Tivoli® Access Manager, DB2 Universal Database, and Edge components.

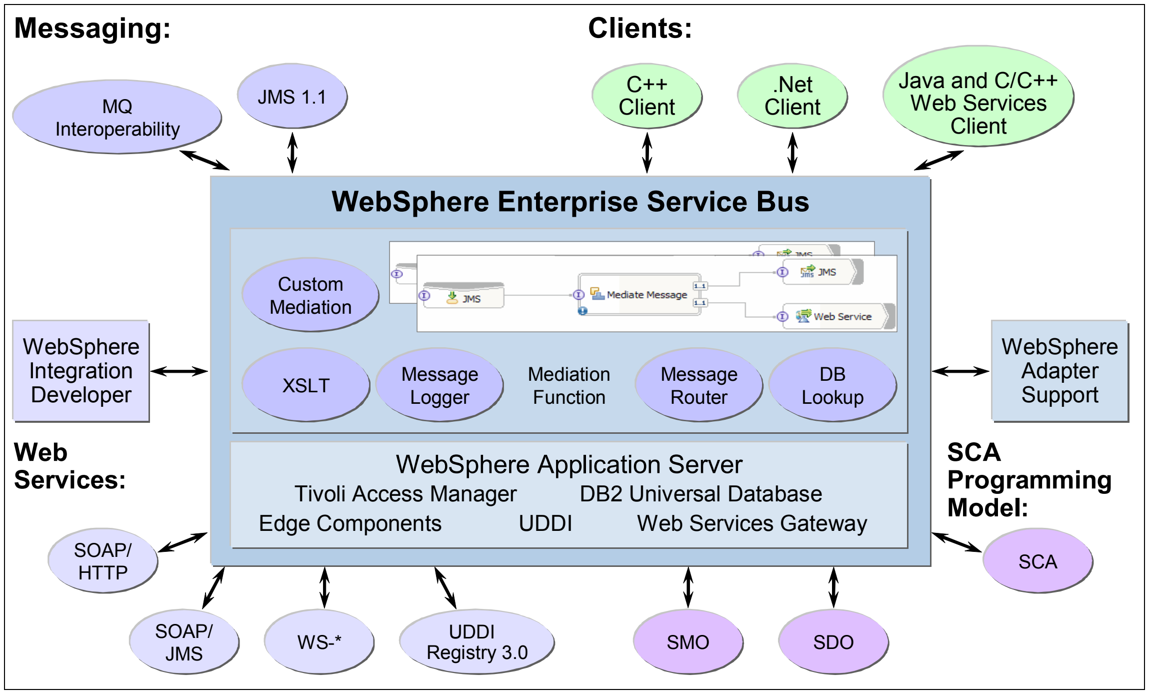


Figure 6-1 WebSphere Enterprise Service Bus at a glance

Moving outward in Figure 6-1 we can see the value that WebSphere Enterprise Service Bus adds to the application server:

- Providing built-in mediation functions, which can be used together to create integration logic for connectivity.
- The SCA programming model supports rapid development of mediation flow components.
- WebSphere Integration Developer is an easy to use tool that supports WebSphere Enterprise Service Bus.
- Leveraging WebSphere Application Server, WebSphere Enterprise Service Bus offers JMS messaging and WebSphere MQ interoperability for messaging, as well as a comprehensive clients package for connectivity.
- Support for J2EE Connector Architecture based WebSphere Adapters.

To implement an SOA properly, it is necessary to have a single invocation model and a single data model. Service Component Architecture (SCA) is this invocation model – every integration component is described through an interface. These services can then be assembled in a component assembly editor thus enabling a very flexible and encapsulated solution.

WebSphere Enterprise Service Bus introduces a new component type to the SCA model, namely the *mediation flow component*. From the perspective of SCA, a mediation flow component is not different to any other service component.

Business Objects are the universal data description. They are used as data objects passed between services and are based on the Service Data Object (SDO) standard. In WebSphere Enterprise Service Bus a special type of SDO is introduced, the *Service Message Object* (SMO).

Also part of the infrastructure is the *Common Event Infrastructure* (CEI) which is the foundation for monitoring applications. IBM uses this infrastructure throughout its product portfolio, and monitoring products from Tivoli as well as WebSphere Business Monitor exploit it. The event definition (Common Business Event) is being standardized through the OASIS standards body, so that other companies as well as customers can use the same infrastructure to monitor their environment.

6.2 Key terms in WebSphere Enterprise Service Bus

This section summarizes the key terms in the context of WebSphere Enterprise Service Bus introduced in this chapter. These key terms are defined in Table 6-1.

Table 6-1 Key terms relating to WebSphere Enterprise Service Bus

Term	**Explanation**
Mediation	A service request interception by an ESB. It typically centralizes logic like routing, transformation, and data handling.
Mediation module	The basic building block in WebSphere Enterprise Service Bus for creating mediations.
Export	Exposes the interfaces of an mediation module and contains the bindings.
Stand-alone reference	The external publishing of an interface for SCA clients only (without a WSDL description).
Import	Represent the service providers that are invoked by a mediation module.
Binding	The protocols and transports assigned to exports and imports.
Mediation flow component	The container for mediation logic inside a mediation module. It provides interfaces and uses references.

Term	Explanation
Interface	Interfaces define access points and are defined using WSDL.
Operation	Operations represent interactions that can be one-way (only input parameters) and two-way (input and output parameters)
Partner reference	The declaration of the referenced interfaces of an mediation flow component.
Wire	An association between components inside a mediation module and exports/imports/stand-alone references.
Mediation flow	The processing steps defined for each interface in form of a request flow and usually a response flow.
Mediation primitive	Units of message processing inside a mediation flow providing different terminals.
Service message object (SMO)	A data object that represents the context, the content and the header information of an application message created during a mediation flow.
Business object	Data type definitions (specified in XML schema) which can be used for input/output parameters.

6.3 Structure of WebSphere Enterprise Service Bus

This section explores the structure of WebSphere Enterprise Service Bus by working through the different layers of the product architecture in a top-down manner.

It describes the following:

- Mediations, service consumers and service providers
- Mediation modules
- Mediation flow components
- Mediation flows
- Mediation primitives

6.3.1 Mediations, service consumers and service providers

A service interaction in SOA defines both service consumers and service providers. The role of WebSphere Enterprise Service Bus is to intercept the requests of service consumers and fulfill additional tasks in mediations in order to support loose coupling. When the mediation completes, the relevant service provider(s) should be invoked. The mediation tasks include:

- Centralizing the routing logic so that service providers can be exchanged transparently
- Performing tasks like protocol translation and transport mapping
- Acting as a facade in order to provide different interfaces between service consumers and providers
- Adding logic to provide tasks such as logging

As shown in figure 6-2 mediations can not only customize the protocol and the details of a request, but they can also modify the results of the reply.

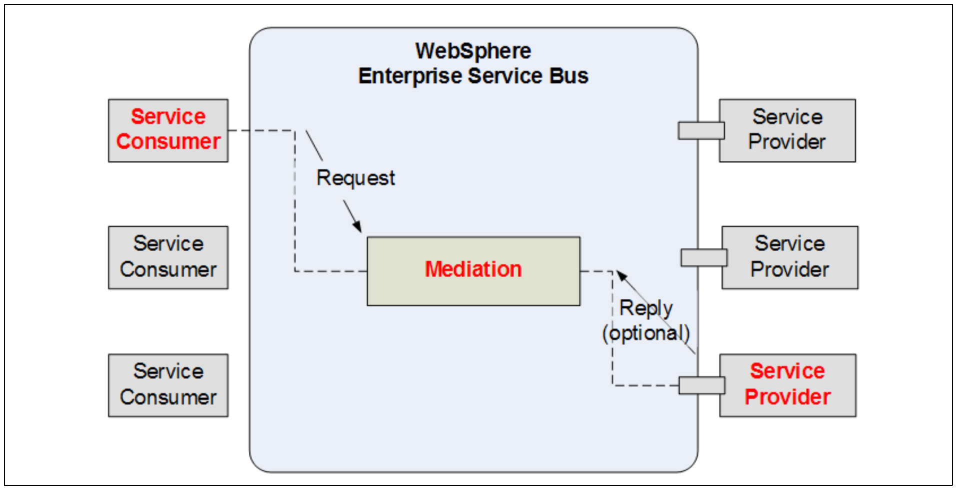


Figure 6-2 Enterprise Service Bus and mediations

WebSphere Enterprise Service Bus can interconnect a variety of different service consumers and providers using standard protocols including:

- JMS
- SOAP over HTTP (for Web services)
- SOAP over JMS (for Web services)

For back-end applications (such as SAP) several IBM WebSphere Adapters (based on JCA) are available.

WebSphere Enterprise Service Bus supports diverse messaging interaction models to meet your requirements, including the following models:

- One-way interactions
- Request-reply
- Publish/subscribe

6.3.2 Mediation modules

The *mediation module* is a new type of SCA component, which can process or mediate service interactions.

As illustrated in Figure 6-3 the mediation module is externalized or made available through an *export* which specifies the interfaces that are exposed. These are defined in a WSDL document. *Stand-alone references* provide the externalized interface only for SCA clients. They do not define a WSDL document, instead they specify the interface declaration in Java (called a *reference*).

The mediation module will typically invoke other service providers. These are declared with the creation of an *import*, which is representing an external service to be invoked.

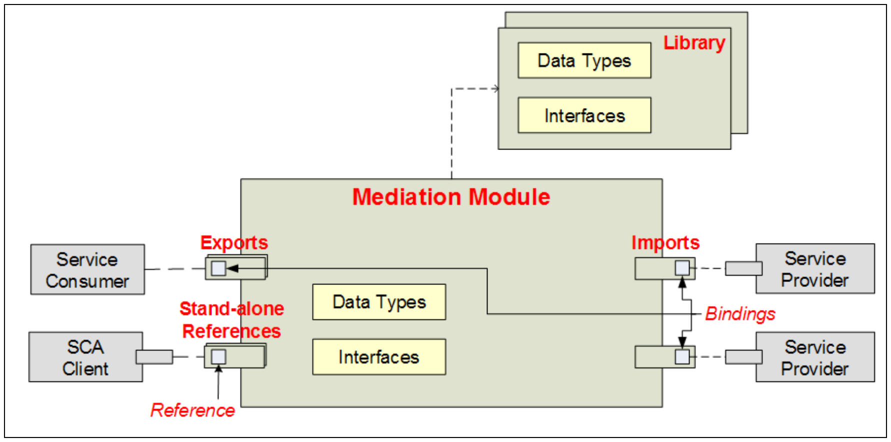


Figure 6-3 Mediation modules

For each export and import an interface needs to be specified. Each interface has multiple operations, which in turn can have multiple input and output parameters associated with either simple data types or business objects. A one-way operation has only input parameters.

Every export and import has to be associated with a *binding*. A binding identifies a specific type of invocation for a service consumer or provider. WebSphere Enterprise Service Bus supports several bindings:

- JMS binding leveraging the JMS V1.1 delivered in WebSphere Application Server V6 using the service integration bus
- Web services using SOAP/HTTP and SOAP/JMS
- JCA compliant WebSphere Adapters
- SCA bindings, which is the default binding used for communication between SCA modules.

  Note: Wiring of SCA components can been done either at development time within WebSphere Integration Developer or administrators can dynamically modify those bindings using the WebSphere Enterprise Service Bus administrative console to *rewire* component interactions (see 9.3.4, “Changing bindings” on page 266)

- Enterprise Java Beans (EJB), which are only valid for import bindings.

Finally, data types (business objects) and interfaces can be defined on the module level, but they can also be defined and referenced in *libraries* in order to centralize them.

6.3.3 Mediation flow components

Inside a mediation module there can be one *mediation flow component.* Mediation flow components offer one or more *interfaces* and use one or more *partner references*. Both get resolved assigning them to exports or imports via wires as shown in Figure 6-4.

Important: You should not try to compare the notions and semantics of components and interfaces of the Java programming language with the ones in WebSphere Enterprise Service Bus model, since this is not applicable in several cases.

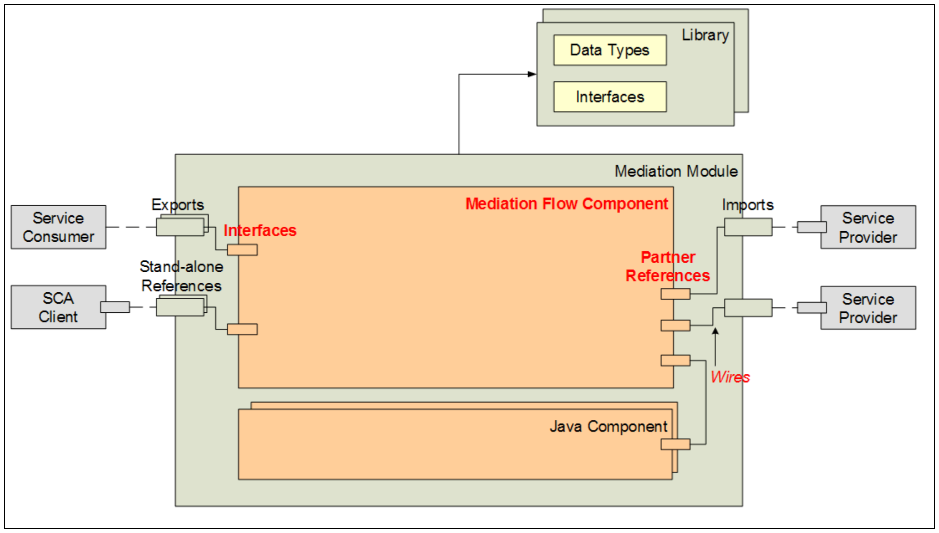


Figure 6-4 Mediation flow component

In addition to the mediation flow component inside a mediation module one or more Java components can be created using custom mediation implementations.

Restriction: WebSphere Integration Developer does not stop you from creating more than one mediation flow component per mediation module, but only one is allowed (as described in the product documentation). Therefore, there is a one-to-one relationship between a mediation module and a mediation flow component.

6.3.4 Mediation flows

Mediation flows (figure 6-5) contain the high-level mediation logic. This means the different processing steps of a request are declared in a graphical way. In WebSphere Enterprise Service Bus, the processing of requests is separated from processing of responses. Therefore, we distinguish between a *request flow* and a *response flow*. In both directions, logic can be added or modifications be applied.

Note: Mediation flows need to be defined *for every operation* that gets exposed via an export of a mediation module. For those operations which do not need any additional functionality to the wrapped interface you just wire them from input to input response.

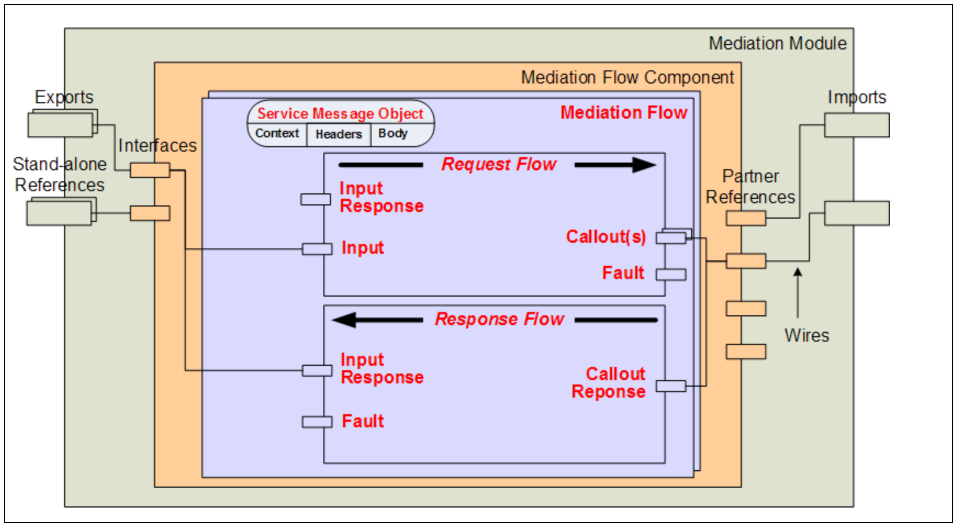


Figure 6-5 Mediation flows

Mediation flows consist of a sequence of processing steps that are executed when an input message is received. A *request flow* begins with a single *input* for the source operation and can have multiple *callouts*. If a message is to be returned to the source directly after processing, it can be wired to an *input response* in the request flow. If fault messages are defined in the source operation, an *input fault* is also created.

A *response flow* begins with one or more *callout responses* and ends with a single input response (and optionally a callout fault). Both a request flow and a response flow are associated with a mediation flow. The request flow can map data to a correlation context and the transient context.

In terms of the actual data WebSphere Enterprise Service Bus introduces the *Service Message Object (SMO)*. It is a special kind of a service data object that represents the content of an application message as it passes through a mediation flow component. As well as the payload in the body it contains context and header information, which can be accessed and acted upon inside the mediation flows.

6.3.5 Mediation primitives

Mediation primitives (figure 6-6) are the smallest building blocks in WebSphere Enterprise Service Bus and they are wired and configured inside mediation flows. They let you change the format, content or target of service requests, log messages, do database lookups, and so forth.

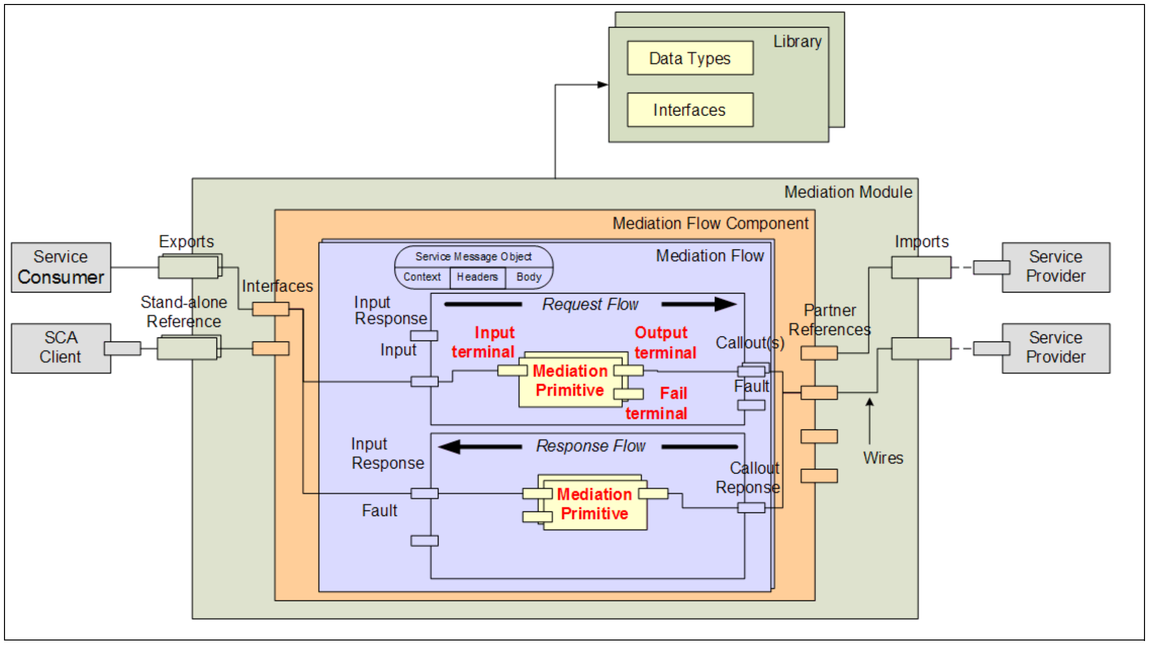


Figure 6-6 Mediation primitives (in the complete overview)

The following standard mediation primitives are provided with WebSphere Integration Developer and WebSphere Enterprise Service Bus V6.0.1:

- The *MessageLogger* primitive logs a copy of a message to a database for future retrieval or audit. The integration developer can customize the primitive by, for example, naming the database.
- The *DatabaseLookup* primitive retrieves values from a database to add them to a message.
- The *MessageFilter* primitive compares the content of a message to expressions configured by the developer, and routes the message to the next mediation primitive based on the result.
- The *XSLT* primitive transforms messages according to transformations defined by an XSL style sheet.
- The *Fail* primitive throws an exception and terminates the path through the mediation flow.
- The *Stop* primitive silently terminates the path through the mediation flow.
- The *Custom mediation* primitive allows the user to implement their own mediate method using Java. The Custom mediation, like the other primitives, receives a Service Message Object and returns a Service Message Object. It

can be used to perform tasks that cannot be performed by using the other mediation primitives.

Mediation primitives have three types of terminal:

- *In terminal:* All mediation primitives have an in terminal that can be wired to accept a message.
- *Out terminal*: Most mediation primitives have one or more out terminals that can be wired to propagate a message (exceptions are the stop and the fail primitive).
- *Fault terminal*: If an exception occurs during the processing of an input message, then the fail terminal propagates the original message, together with any exception information.

6.4 Related technologies

This section explores some of the accompanying features of WebSphere Enterprise Service Bus in more detail. It describes:

- Service message objects (SMO)
- WebSphere Enterprise Service Bus bindings
- Quality of service
- Common event infrastructure (CEI)
- Deployment of mediations

6.4.1 Service message objects (SMO)

Messages can come from a variety of sources, so the payload has to be able to carry a number of different types of messages. Mediation primitives need to be able to operate on these messages and SMO represents the common representation that is needed for that.

The kinds of messages handled by WebSphere Enterprise Service Bus include:

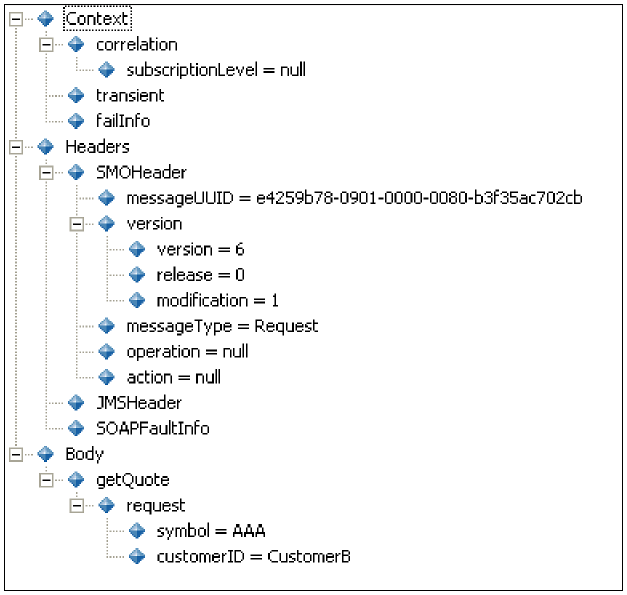
- SDO data object
- SDO data graph
- SCA component invocation message (request, reply or exception)
- SOAP message
- JMS message

The SMO model is extensible so could support other message types in the future such as COBOL structures. SMO extends SDO with additional information to support the needs of a messaging subsystem.

SMO structure

All SMOs have the same basic structure, defined by an XML schema. An SMO has three major sections. The *body* contains the application data (payload) of the message, particularly the input or output values of an operation. The *headers* contain the information relevant to the protocol used to send the message. The *context* covers the data specific to the logic of a flow or failure information. Figure 6-7 shows a sample SMO when calling the stock quote sample provided with WebSphere Enterprise Service Bus.

.



```
Context
  correlation
    subscriptionLevel = null
  transient
  failInfo
Headers
  SMOHeader
    messageUUID = e4259b78-0901-0000-0080-b3f35ac702cb
    version
      version = 6
      release = 0
      modification = 1
    messageType = Request
    operation = null
    action = null
  JMSHeader
  SOAPFaultInfo
Body
  getQuote
    request
      symbol = AAA
      customerID = CustomerB
```

Figure 6-7 Sample SMO

Data section

The data carried in the SMO body is the operation defined by the interface specification and the inputs/outputs/faults specified in the message parts set in the business object definition. This is shown in Figure 6-8.

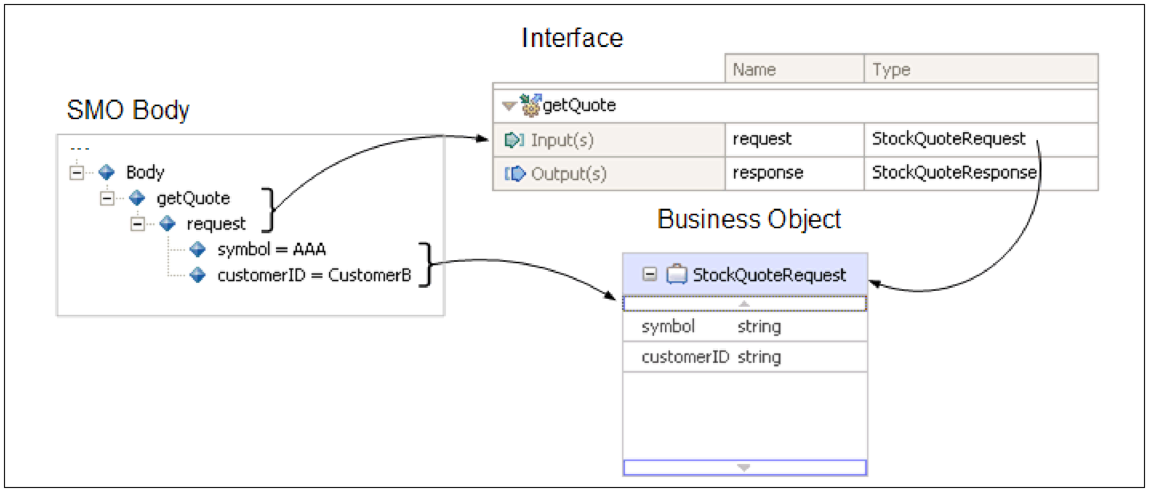


Figure 6-8 Content of the SMO body

Context section

The context includes the correlation and transient context information. Correlation is used to maintain data across a request/response flow, whereas transient maintains data only in one direction.

Both of these are used to pass application data between mediation primitives. They are described as business objects, which contain XML schema described data objects and are specified on the mediation flows input node properties.

The context also includes the *failInfo*, which is added to the SMO when a fault terminal flow is used. The information provided includes the *failureString* (nature of the failure), *origin* (mediation primitive in which the failure occurred), *invocationPath* (the flow taken through the mediation) and *predecessor* (previous failure).

Header section

The header section of a SMO contains the following supplemental information:

- SMOHeader: information about the message (message identifier, SMO version)
- JMSHeader: used when there is a JMS import or export binding
- SOAPHeader: used when there is a Web services import or export binding
- SOAPFaultInfo: contains information about SOAP faults
- Properties[]: arbitrary list of name value pairs (for example JMS user properties)

SMO manipulation

During the execution of mediation flows the active mediation primitives can access and manipulate the SMO. There are three different ways to access SMOs:

- XPath V1.0 expressions
  This is the primary mechanism used by all mediation primitives.
- XSL stylesheets
  They can be used by the XSLT mediation primitive and are the common way to modify the SMO type within a flow. It can also be used to modify the SMO without changing the type (using XSLT function and logical processing with XSL choose statements).
- Java code
  Using the Custom Mediation primitive you can access the SMO either using the generic DataObject APIs (commonj.sdo.DataObject, which is loosely typed) or the SMO APIs (com.ibm.websphere.sibx.smobo, strongly typed).

6.4.2 WebSphere Enterprise Service Bus bindings

Bindings identify a specific type of invocation for a service consumer or provider. Bindings can be applied to mediation module imports or exports. Exports let a mediation module offer a service to consumers. They define interactions between SCA modules and service consumers. Export bindings define the specific way that an SCA module is accessed by others.

Imports let a mediation module access external services (services that are outside the SCA module) in a transparent manner. Imports define interactions between SCA modules and service providers. Import bindings define the specific way that an external service is accessed.

WebSphere Enterprise Service Bus supports the following bindings:

- Web service binding

  Using a Web service binding on an export it exposes the module as a Web service. To invoke an external Web service an import with a Web service binding is used. This binding always uses SOAP messages and two transports are available:

  – SOAP/HTTP
  – SOAP/JMS
- SCA binding

  – SCA bindings connect SCA modules with each other.

  – This is the default binding.

- WebSphere Adapter binding
  - WebSphere Adapters enable interaction with Enterprise Information Systems (EIS).
  - The Enterprise Service Discovery tool can be used to create import and exports representing applications on EIS systems. To use EIS bindings a resource adapter is needed.
- Java Message Service (JMS) V1.1 binding
  - JMS can exploit various transport types, including TCP/IP and HTTP(S).
  - There are predefined JMS bindings that support JMS text messages containing Business Object (BO) XML. The predefined JMS bindings also support JMS object messages containing serialized Java Business Objects.
  - You can use JMS custom bindings to support other types of JMS messages. However, custom bindings require some coding to translate the message.
  - If you want a module to receive a JMS message from a queue or topic, you need to use an export with a JMS binding. If you want a module to send a JMS message, you use an import with a JMS binding.

Note: The Publish/Subscribe interaction model can be applied in WebSphere Enterprise Service Bus using the JMS binding.

- EJB bindings (only for imports)
  - An import component can have a stateless session EJB binding.

6.4.3 Quality of service

Qualifiers in SCA allow developers to place quality of service requirements on the SCA runtime. There are several different categories of qualifiers available in SCA. These are:

- Security
- Transactions (with ActivitySessions as a special type)
- Reliable Messaging

Each qualifier has a particular scope within the Service Component Definition Language (SCDL) specification for a SCA component where the qualifier can be added (interface, implementation, partner reference).

For example some qualifiers can be specified at the partner reference level, while others may only be valid at the interfaces or implementation level. Figure 6-9 shows the conceptual model for SCA service qualifiers.

.

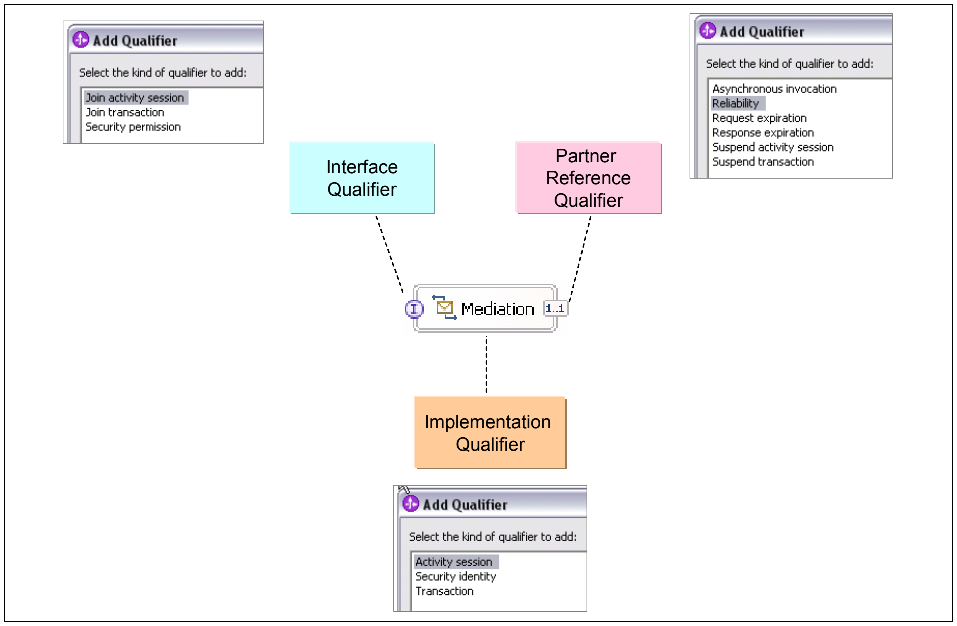


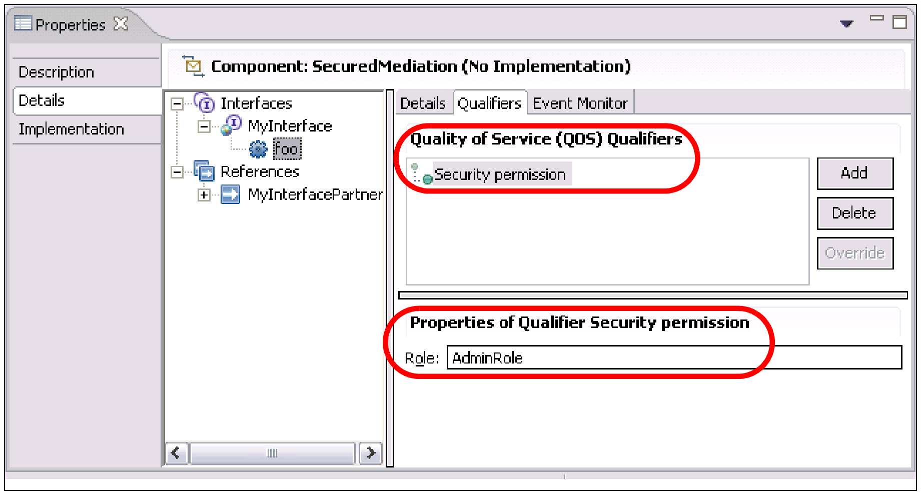
Figure 6-9 SCA quality of service qualifier model

In the following subsections we briefly describe the various qualifiers that are available and the valid scope for each will be examined.

Security

In WebSphere Integration Developer you specify security attributes for mediation flow components in the properties view at the boundaries and the implementation of an component.

At the interface level you can define the permission for every operation (Figure 6-10). At the mediation flow component implementation level you can define under which identity the component gets executed (initiating a role-change) as shown in Figure 6-11.



Component: SecuredMediation (No Implementation)

Quality of Service (QOS) Qualifiers

Security permission

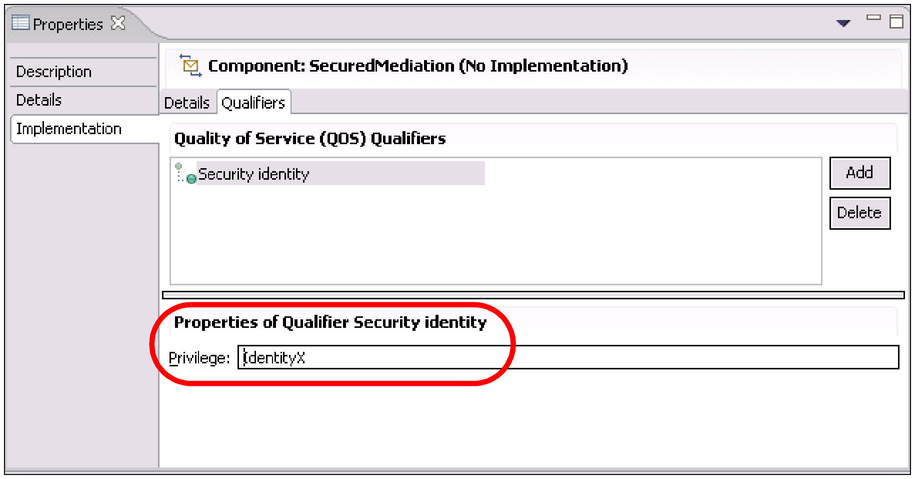
Properties of Qualifier Security permission

Role: AdminRole

Figure 6-10 Security permission qualifier on interfaces

Use the *security permission* qualifier to specify a role, which is a semantic grouping of permissions that a given type of users must have to use an operation in an interface. The identity of the caller must have this role in order to be permitted to call the interface or operation. If no security permission is specified, then no permissions are checked and all callers are permitted to call the interface or operation.

The *security identity* qualifier is a privilege specification that you can use to provide a logical name for the identity under which the implementation executes at run time (Figure 6-11). An implementation has to be created for this qualifier to be specified. If this qualifier is not specified, then the implementation executes under the identity of its caller. Alternatively it is executed under the hosting container's identity if no caller identity is present. Roles are associated with the identity and the roles dictate whether the implementation is authorized to invoke other components.



Component: SecuredMediation (No Implementation)

Quality of Service (QOS) Qualifiers

Security identity

Properties of Qualifier Security identity

Privilege: IdentityX

Figure 6-11 Security identity qualifier for mediation components

Depending on the bindings you have created, WebSphere Enterprise Service Bus will generate the relevant J2EE artifacts. In order to integrate remote clients (for example using the Web service security specifications) with the J2EE application infrastructure, a proper distributed security infrastructure needs to be built. For additional information of securing Web Services, see the redbook *WebSphere Version 6 Web Services Handbook Development and Deployment,* SG24-6461

Activity sessions

This qualifier determines if the components processing will be executed under an *activity session*, which provides an alternate unit-of-work scope to the one provided by global transaction contexts. An activity session context can have a longer lifetime global transaction context and can encapsulate global transactions.

Note: Activity sessions are an extension of J2EE introduced with WebSphere Application Server V5. See the Infocenter documentation for more information on this topic:

```
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websp
here.base.doc/info/aes/ae/welc6tech_as.html
```

You can specify the activity session qualifier at all three levels:

- Interface level

  Can optionally join a propagated (client) activity session.

- Implementation level

  For the implementation the qualifier specifies if the component can run under an established activity session. The default is that if an activity session has been propagated from the client, the runtime environment will dispatch methods from the component in the activity session. Otherwise, the component will not run under any activity session.

- Partner reference level

  By default, activity session context is always propagated to a target component when it is invoked using the synchronous programming model. If the client does not want a target component to federate with the client's activity session, further qualification of the partner reference is required using the *suspend activity session* qualifier.

Transactions

This qualifier determines the logical unit of work that the component processing executes. For a logical unit of work, all of the data modifications made during a transaction are either committed together as a unit or rolled back as a unit.

- On an interface level the *join transaction* qualifier determines if the hosting container will join any propagated transaction.
- On a implementation level the *transaction* qualifier can be set either to *global* (where multiple resource managers are required), *local (default)* (running in a local transaction) or *any* (dispatching the global transaction context if existent).

  Note: The different combinations of the interface and implementation qualifiers define the behavior for the target component. Not all combinations are allowed.

- For a partner reference you can specify the *Suspend transaction* qualifier, which can be set to *false* (so the synchronous invocations run completely within any global transaction) and *true* (where synchronous invocations occur outside any client global transaction).

In addition the *asynchronous invocation* determines if asynchronous invocations should occur as part of any client transaction. When set to *call* (default) the asynchronous invocations using the partner reference will occur immediately, whereas with *commit* the partner reference will be transacted as part of any client global transaction or extended local transaction which postpones the availability of the request.

Asynchronous reliability

To support asynchronous invocation of components, asynchronous reliability qualifiers can be specified for the partner reference only. They take effect when asynchronous programming calls are used by the client to invoke a service. The reliability qualifier specifications are:

- Reliability:

  The reliability qualifier determines the quality of an asynchronous message delivery. In general, better performance usually means less reliable message delivery. With an *assured* specification, the client application cannot tolerate the loss of a request or response message. With a *best effort* specification, the client application can tolerate the possible loss of the request or response message.

- Request expiration (milliseconds)

  Request expiration is the length of time after which an asynchronous request will be discarded if it has not been delivered, beginning from the time when the request is issued. Zero denotes an indefinite expiration.

- Response expiration (milliseconds)

  Response expiration is the length of time that the runtime environment must retain an asynchronous response or provide a callback, beginning from the time when the request is issued. Zero denotes an indefinite expiration.

6.4.4 Common event infrastructure (CEI)

The CEI is a core component of WebSphere Enterprise Service Bus leveraged from WebSphere Application Server and provides facilities for the runtime environment to persistently store and retrieve events from different programming environments. This section briefly introduces the basic event-related concepts:

- Common Event Infrastructure (CEI)
- Common Base Events (CBE)

Common Event Infrastructure

In WebSphere Enterprise Service Bus, the CEI is used to provide basic event management services, such as event generation, transmission, persistence, and consumption. CEI was developed to address industry-wide problems in exchanging events between incompatible systems, many of which employed different event infrastructures, event formats, and data stores.

Common Base Event

Although CEI provides an infrastructure for event management, it does not define the format of events. This is defined by the Common Base Event specification, which provides a standard XML-based format for business events, system events, and performance information. Application developers and administrators can use the Common Base Event specification for structuring and developing event types.

The key concept in the Common Base Event model is the *situation*, which is any occurrence that happens anywhere in the computing system, such as a user login or a scheduled server shutdown. The Common Base Event model defines a set of standard situation types, such as *StartSituation* and *CreateSituation*, that accommodate most of the situations that might arise.

In the Common Base Event model, an event is a structured notification that reports information related to a situation. An event reports three kinds of information:

- The situation that has occurred
- The identity of the affected component
- The identity of the component that is reporting the situation, which might be the same as the affected component

In the WebSphere Integration Developer editors the specification of event monitoring is based on the operation level.

6.4.5 Deployment of mediations

WebSphere Integration Developer creates J2EE artifacts which are stored in EAR files. Logically, mediation modules can be thought of as one entity. In reality, SCA modules are defined by a number of XML files (stored in one JAR file later on), which are the basis for the generation of the J2EE artifacts.

J2EE staging projects

For any given module project there will be up to four J2EE staging projects generated with naming conventions that are based on the modules project name (in the following called *MyModule*). In the Business Integration view of WebSphere Integration Developer you will not be able to see these projects. To view these you will need to change to another perspective such as the J2EE perspective.

A module may implement the following staging projects

- *MyModuleApp* - the enterprise application staging project

  Enterprise application projects contain artifacts and metadata for an entire enterprise application. It includes information such as the name of the EJB projects contained within the enterprise application, and the context root for the Web modules within the enterprise application.

- *MyModuleEJB* - the EJB staging project

  EJB projects contain artifacts and metadata for Enterprise Java Beans. This project holds generated EJBs that represent the runtime artifacts that make components. For example, an SCA export results in a generated stateless session EJB.

- *MyModuleEJBClient* - the EJB client staging project

  EJB Client projects contain artifacts that represent the client-side for the EJBs in the EJB projects. For example they include stubs for remote and home interfaces so that clients of EJBs can interact with the EJBs.

- *MyModuleWeb* - the dynamic Web staging project

  Dynamic Web projects contain artifacts that represent Web components such as servlets and JSPs. In particular the Web project contains a servlet that represents an HTTP router for inbound HTTP traffic.

Deployment of mediation modules

Mediation modules are created using WebSphere Integration Developer, and deployed to WebSphere Enterprise Service Bus inside an *EAR* (Enterprise Archive) file. Therefore, a mediation module is deployed to WebSphere Enterprise Service Bus in the same way you deploy any enterprise application.

Note: It is advisable to package a significant amount of mediation logic into one module, otherwise you might end up with an enormous number of enterprise applications on your server.

A high-level overview of deployment is shown in Figure 6-12. When a generated EAR is deployed to a server, it gets bound to several J2EE resources including data sources, JMS destinations, and J2EE Connector Architecture resource adapters.

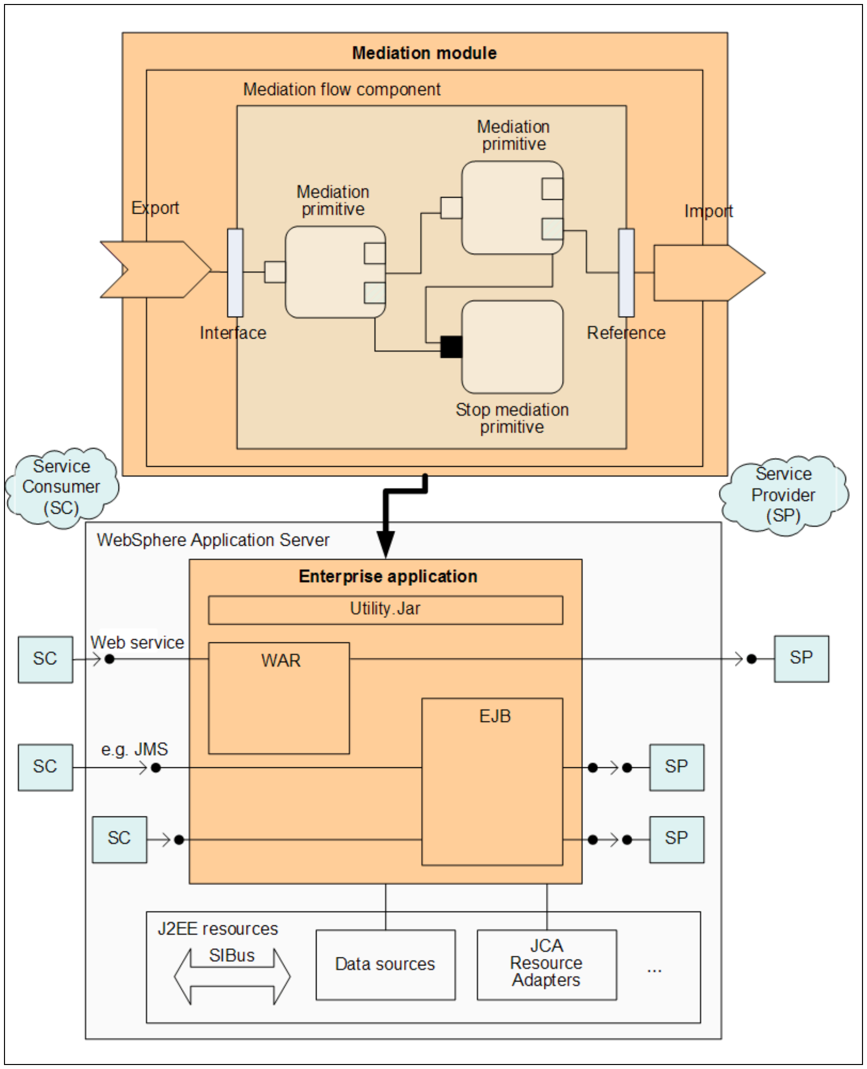


Figure 6-12 Deployment of mediation modules

7

WebSphere Integration Developer key concepts and common tasks

WebSphere Integration Developer is the development environment for building integrated business applications targeted for WebSphere Enterprise Service Bus and WebSphere Process Server. One of the primary purposes of WebSphere Integration Developer is to provide the appropriate tools to easily build and test SCA based applications.

This chapter discusses WebSphere Integration Developer key concepts and common tasks in terms of mediation module development for deployment to WebSphere Enterprise Service Bus.

Figure 7-1 presents an overview of the common tasks. It shows the main stages in the mediation module development process and provides assistance navigating this chapter.

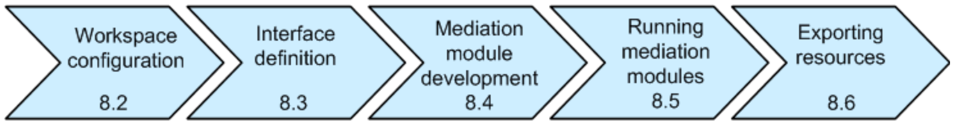


Figure 7-1 Common tasks

7.1 Key terms and concepts

WebSphere Integration Developer is built on the Rational Software Development Platform (RSDP), which is based on Eclipse 3.0 technology.

Each IBM product built on RSDP will coexist and share plugins and features with other RSDP based products. RSDP is installed once per system with the first product that is installed. As other products built on this platform are installed on the system only the necessary plugins are installed.

Note: For more information about Eclipse and tutorials visit `http://www.eclipse.org` and explore the Getting Started pages.

This section introduces some of the basic terms and concepts used in WebSphere Integration Developer. Many of these terms and concepts are common to all RSDP products.

7.1.1 User roles

Two user roles are associated with WebSphere Integration Developer:

- integration developer
- application developer

The integration developer is the primary role. It focuses on building service-oriented solutions. This user role expects the tooling to simplify and abstract advanced IT implementation details. The integration developer is familiar with basic programming constructs such as loops, conditions and string manipulation.

The application developer is knowledgeable in development platforms like J2EE, understands service-oriented architecture, Web services and Java. Application developers implement application specific business logic and expose it as a service.

This chapter focuses on tools used by the integration developer.

7.1.2 The workbench

When you first start a new workspace you will see the Welcome screen (Figure 7-2). From this screen you can access information such as the product overview, cheat sheets, tutorials, samples, migration information and Web resources.

Tip: If you close the Welcome screen you can access it again by selecting **Help** → **Welcome** from the menu bar.

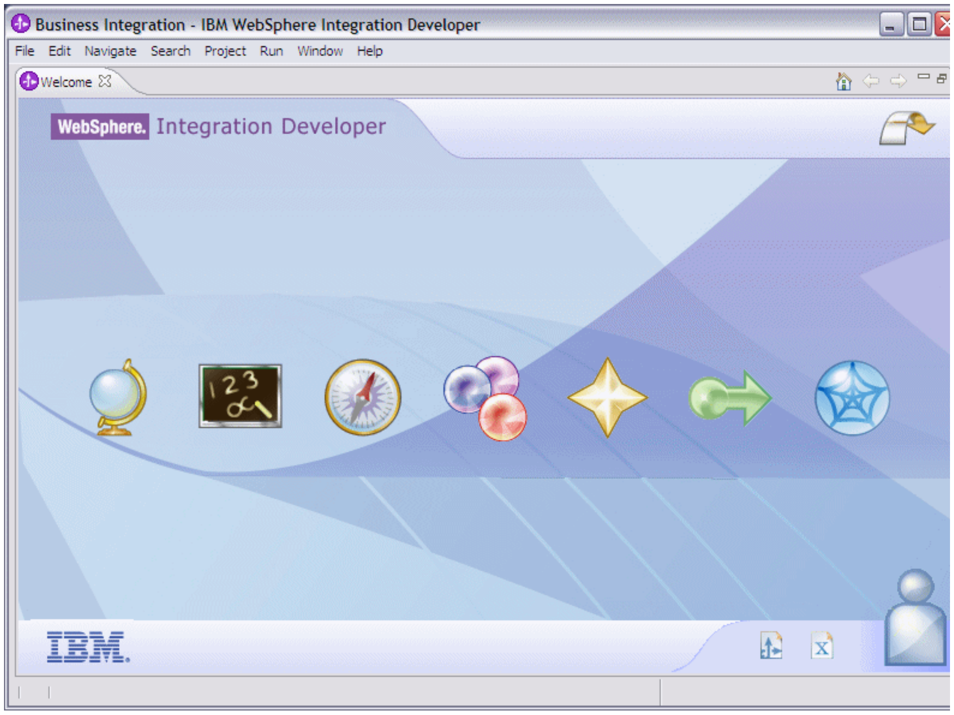

Business Integration - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Welcome
WebSphere. Integration Developer
IBM.

Figure 7-2 Welcome screen

Clicking the arrow labelled **Workbench** closes the Welcome screen and opens the *Business Integration perspective*.

The workbench (Figure 7-3) is where you will spend most of your time developing mediation modules. It offers the developer a choice of perspectives and an array of toolbars and menu items which are used to accomplish a variety of tasks. These are introduced and explained later in this chapter.

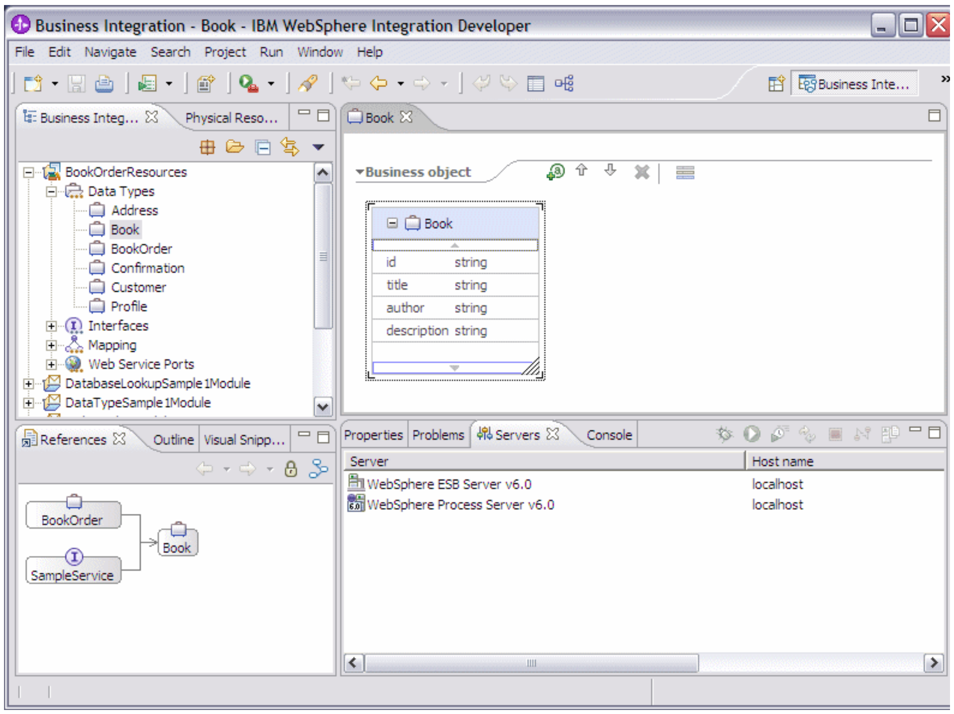


Figure 7-3 The workbench

7.1.3 Workspaces

A *workspace* is a directory where your work is stored. You can create many workspaces and choose which one to work on at any time. A common scenario is to have separate workspaces for different projects you may be working on. This lets you organize you work efficiently, keep backups of entire workspaces and share your workspace with other developers.

Tip: To switch workspaces select **File** → **Switch Workspace** from the menu bar.

7.1.4 Project types

There are two important project types when working with WebSphere Integration Developer: *module projects* and *shared libraries*.

Module project

A module project represents a single deployable unit and encapsulates SCA components, J2EE projects, Java projects and required libraries. When deploying to WebSphere Enterprise Service Bus your choice of module type is limited to *mediation* modules.

Shared library

A shared library is another type of business integration project. Unlike modules, libraries are not deployable units. Shared libraries hold resources that are shared between module projects. At run time, libraries are not shared but are deployed with the module that depends on it.

If you are deploying to WebSphere Enterprise Service Bus, you can only create two types of artifacts in a shared library: business objects and interfaces. The *Mapping* folder only applies to WebSphere Process Server projects.

Additionally, you can use shared libraries to hold WSDL bindings in a Web Service Bindings folder which is created when you copy WSDL files into your library.

Tip: Libraries can be added to the dependency list for a module from the Module Dependency editor. To open this editor right click on the module folder and select **Open Dependency Editor**.

7.1.5 Perspectives

A *perspective* is a role-based collection of views and editors.

Perspectives are very useful because they offer users the tools that are most needed to perform their current job. Perspectives are fully customizable; views and editors can be added, removed and rearranged.

Tip: To restore any perspective to its default layout select **Window** → **Reset Perspective** from the menu bar.

The primary WebSphere Integration Developer perspective is the *Business Integration perspective.* We will use this perspective almost exclusively because it contains all the tools we need to create, develop and manage business integration projects. Figure 7-3 on page 156 shows the Business Integration perspective on the workbench.

Another useful perspective is the Debug perspective which is used during testing to specify breakpoints and inspect variables and messages to determine and fix problems.

7.1.6 Views

A *view* is used to present information about a resource. Views are also used for navigating the information in the workspace.

Views might appear by themselves or stacked with other views in a tabbed notebook.

Business Integration view

This is the primary Business Integration perspective view. The Business Integration view is used to navigate workspace resources. It provides a logical grouping of resources and hides artifacts that are not essential for business integration development. It is initially by itself on the top left area of the Business Integration perspective (Figure 7-4).

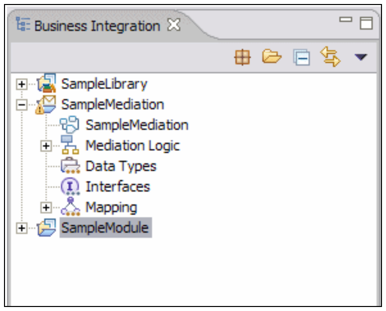


Figure 7-4 Business Integration view

Physical Resources view

The Physical Resources (Figure 7-5) view by default is not open in the Business Integration perspective. This view shows the physical resources that are hidden in the Business Integration view. For example, the individual SCA resources that make up the elements of your module. You can also use this view to learn more about the artifacts that are generated when creating integration modules.

Tip: You can use the **Show files** context menu from the Business Integration view to open the Physical Resources view.

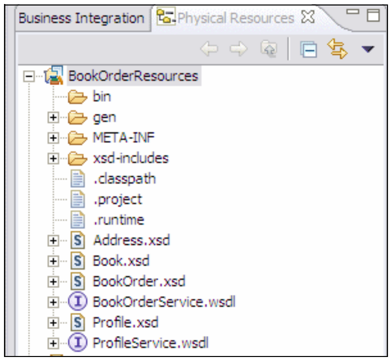


Figure 7-5 Physical Resources view

References view

The References view (Figure 7-6) is used in association with the Business Integration view. The contents of the References view is based on the artifact that is selected in the Business Integration view. For example, if a business object is selected in the Business Integration view, the References view will show other objects that are referenced by the selected object.

Tip: You can open and navigate to resources directly from the References view.

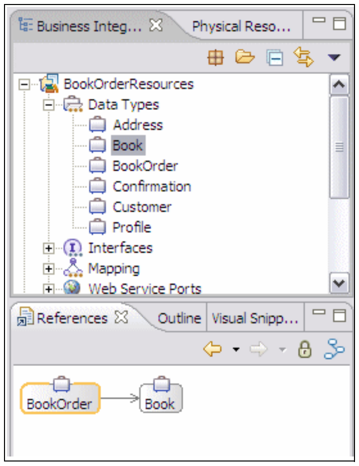


Figure 7-6 References view

Outline view

The Outline view (Figure 7-7) lets you navigate resources within a single module. This view has two modes: tree and overview. The tree mode shows resources grouped by resource type in expandable folders and lets you select elements. The outline view shows the full assembly diagram and lets you quickly scroll to a particular area of the assembly that might not be visible in the Assembly Diagram editor. A shaded marquee surrounds the portion of the diagram currently visible in the Assembly Diagram editor.

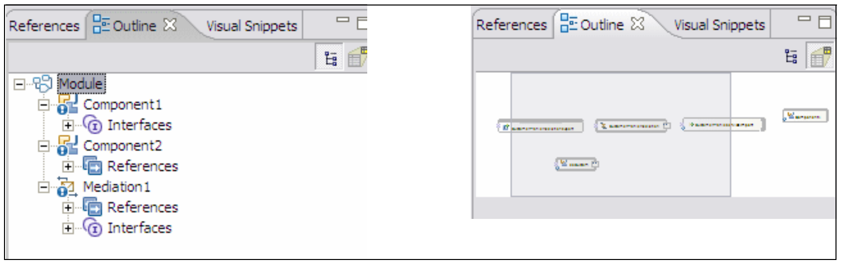


Figure 7-7 Outline view, tree and overview

Visual Snippets view

The Visual Snippets view (Figure 7-8) lists Java snippets that can be used to visually build code. Both standard snippets that come with the product and custom snippets are shown in this view and are available for drag and drop support in the various visual code editors.

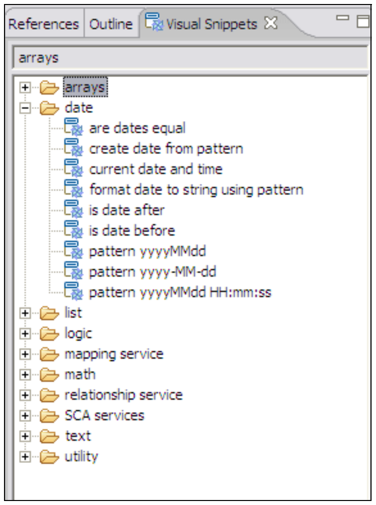
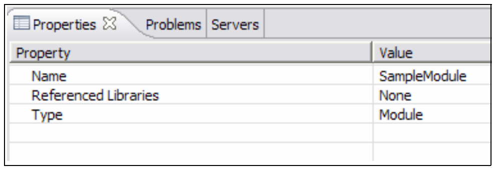


Figure 7-8 Visual Snippets view

Properties view

The Properties view (Figure 7-9) displays information about the currently selected object. It is stacked with the Problems and Servers view at the bottom right area of the workbench.

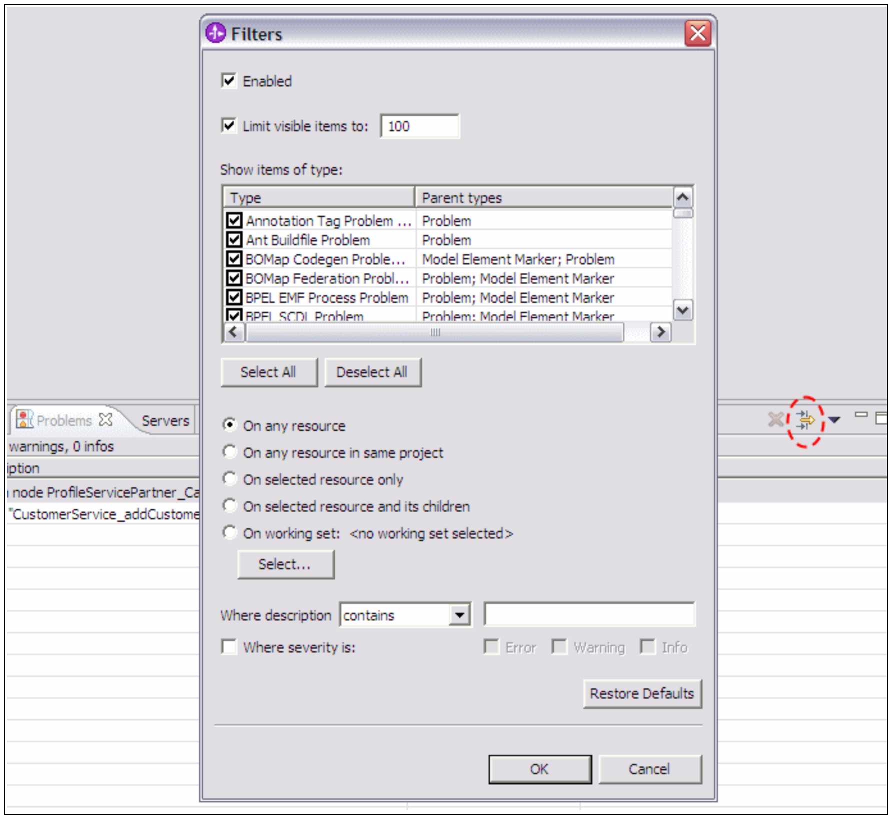


Property	Value
Name	SampleModule
Referenced Libraries	None
Type	Module

Figure 7-9 Properties view stacked in a tabbed notepad

Problems view

The Problems view displays all compilation errors and warnings. You can use filters to customize the amount and type of information shown (Figure 7-10).



[✓] Enabled

[✓] Limit visible items to: 100

Show items of type:

Type	Parent types
[✓] Annotation Tag Problem ...	Problem
[✓] Ant Buildfile Problem	Problem
[✓] BOMap Codegen Proble...	Model Element Marker; Problem
[✓] BOMap Federation Probl...	Problem; Model Element Marker
[✓] BPEL EMF Process Problem	Problem; Model Element Marker
[✓] BPEL SCDL Problem	Problem; Model Element Marker

(●) On any resource
( ) On any resource in same project
( ) On selected resource only
( ) On selected resource and its children
( ) On working set: <no working set selected>

Where description contains

[ ] Where severity is: [ ] Error [ ] Warning [ ] Info

warnings, 0 infos
iption
node ProfileServicePartner_Ca
"CustomerService_addCustome

Figure 7-10 Applying filters to the Problems view

Servers view

The Servers view lets you manage the integrated test environment. From this view you can start, stop and publish modules to your test server. Please refer to the section 7.5.2, “Managing test servers” on page 198 for more information about managing servers and the Servers view.

7.1.7 Editors

An *editor* is a tool to create and modify files. Depending on the type of file that you are editing, the appropriate editor opens in the center or main pane of the workbench.

For example, a text editor opens when you double click a text file and business objects open in the Business Object editor.

Note: An asterisk (*) preceding the object name on the editor tab indicates that the resource being edited has unsaved changes.

Assembly Diagram editor

Use this editor to compose your mediation module. Typically you drop into the canvas SCA components like Mediation Flow components, imports and exports, specify their interfaces and bindings, and wire them together using the Assembly Diagram editor (Figure 7-11).

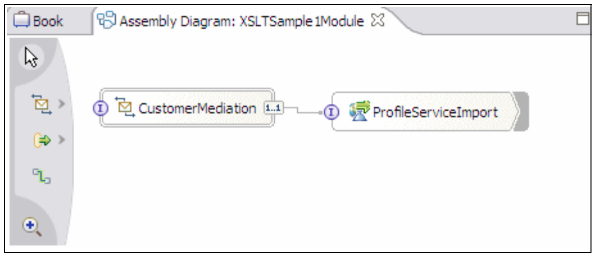
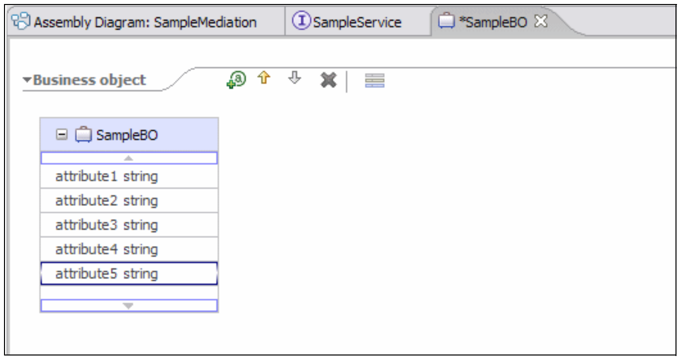


Figure 7-11 Assembly Diagram editor

Business Object editor

The Business Object editor (Figure 7-12) is used to build and edit business objects and business graphs.

Use this editor to add, delete and reorder attributes and to change the type of an attribute.

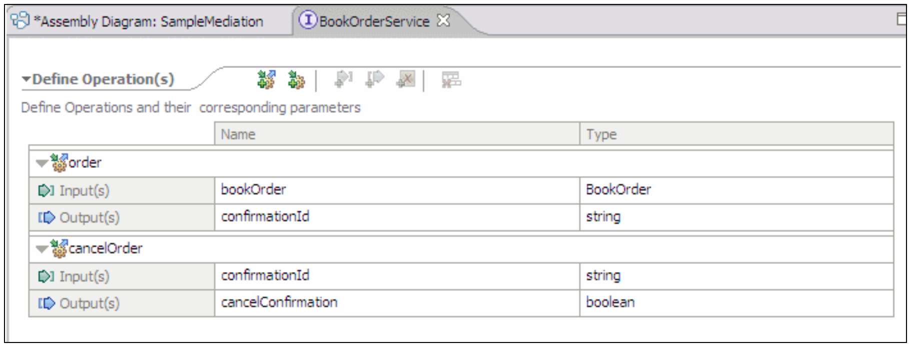


attribute1	string
attribute2	string
attribute3	string
attribute4	string
attribute5	string

Figure 7-12 Business Object editor

Interface editor

The Interface editor (Figure 7-13) is used to build WSDL Port Type interfaces used to define some SCA components. You use this editor to add and remove operations and specify operation's inputs and outputs.


*Assembly Diagram: SampleMediation
BookOrderService
Define Operation(s)
Define Operations and their corresponding parameters

	Name	Type
order		
Input(s)	bookOrder	BookOrder
Output(s)	confirmationId	string
cancelOrder		
Input(s)	confirmationId	string
Output(s)	cancelConfirmation	boolean

Figure 7-13 Interface editor

Visual Java Snippet editor

The Visual Java™ Snippet editor (Figure 7-14) is used to compose custom snippets visually. You can create your own custom visual snippets and add them to the snippet editor or you can use the standard snippets that come with the product.

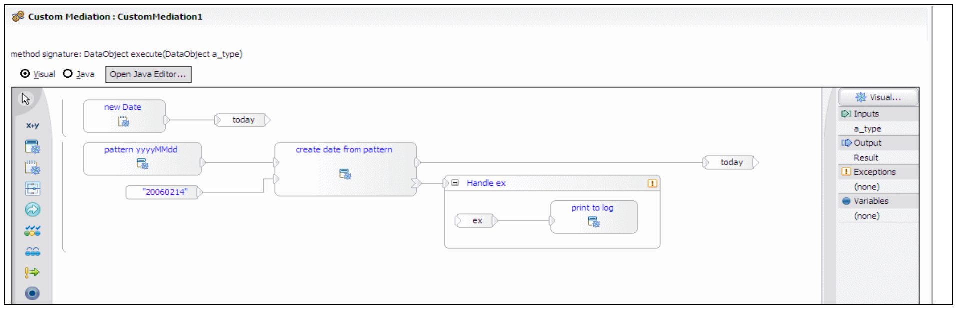


Figure 7-14 Visual Snippet editor

Tip: You can double click on a view or editor tab to maximize it. Double click it again to restore it.

7.1.8 Mediation module

A *mediation module* (Figure 7-15) is a Business Integration project. It is used to intercept and modify messages between service consumers (exports) and service providers (imports).

The mediation module contains exports, imports, a new type of SCA component called *mediation flow component*, and SCA Java components.

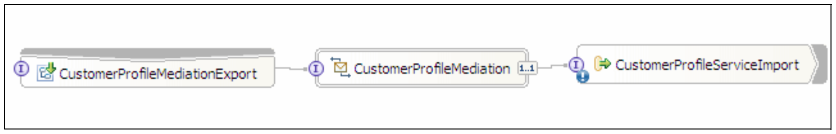
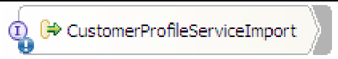
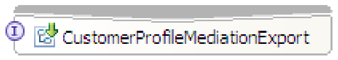
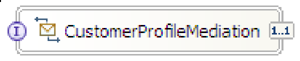
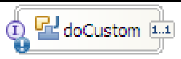


Figure 7-15 Mediation module

You will soon be able to recognize the different graphical representations for the mediation module elements. Refer to Table 7-1 to identify them easily on module assembly diagrams.

Table 7-1 Elements of a mediation module

Mediation module element	Symbol
Import	
Export	

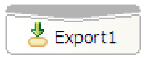
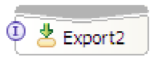
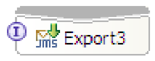
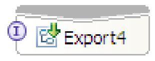
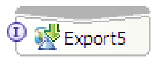
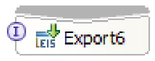
Mediation module element	Symbol
Mediation flow component	
SCA Java component	

7.1.9 Exports

An *export* represents a service consumer outside the scope of the module. Exports in mediation modules are like normal SCA Exports with all the supporting bindings including the default SCA, JMS and Web services.

Use Table 7-2 as a reference of Export icons and their bindings.

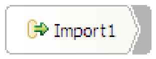
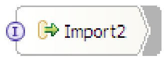
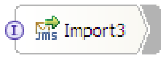
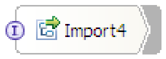
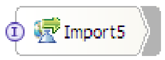
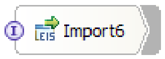
Table 7-2 Export icons

Export icon	Description
	Export with no interface and no binding
	Export with interface and no binding
	Export with interface and JMS binding
	Export with interface and SCA binding
	Export with interface and Web service binding
	Export with EIS binding

7.1.10 Imports

An *import* represents a service provider outside the scope of the module. Imports in mediation modules are like normal SCA Imports with all the supporting bindings including the default SCA, JMS and Web services. Use Table 7-3 as a reference of Import icons and their bindings.

Table 7-3 Import icons

Import icon	Description
 Import1	Import with no interface and no binding
 Import2	Import with interface and no binding
 Import3	Import with interface and JMS binding
 Import4	Import with interface and SCA binding
 Import5	Import with interface and Web service binding
 Import6	Import with EIS binding

7.1.11 Mediation flow components

A *mediation flow component* contains logic for how the message is processed between the input and output of the flow.

Functions like routing, transformation, augmentation, logging or any other custom processing of messages occur within the mediation flow component.

Note: Only one mediation flow component can exist in a mediation module.

7.1.12 Mediation primitives

Mediation *primitives* are building blocks used to build mediation flows. WebSphere Integration Developer supplies a set of built-in primitives and a Custom primitive used to execute user defined mediation logic.

Table 7-4 lists all primitives and their toolbar icon and description.

Table 7-4 Mediation primitive types

Mediation primitives	Symbol	Description
Message Logger	📋	To log message information to a database
Message Filter	✉	To filter messages selectively forwarding them on to output terminals based on a simple condition expression.
Database Lookup	🛢	To access information in a database and store it in the message
XSLT	📄	To manipulate or transform messages using XSL transformation
Stop	🟥	To stop a path in the flow without generating an exception
Fail	❗	To stop a path in the flow and generate an exception
Custom	⚙	For custom processing of a message. Uses a custom SCA Java component for custom message processing

7.2 Workspace configuration

This is the initial stage in the development process (Figure 7-16). This section introduces basic workspace configuration tasks.

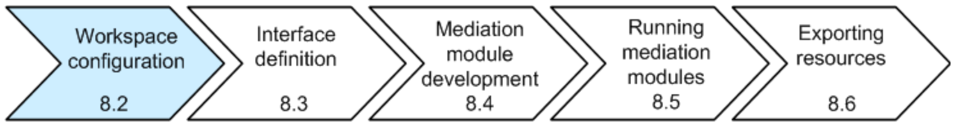
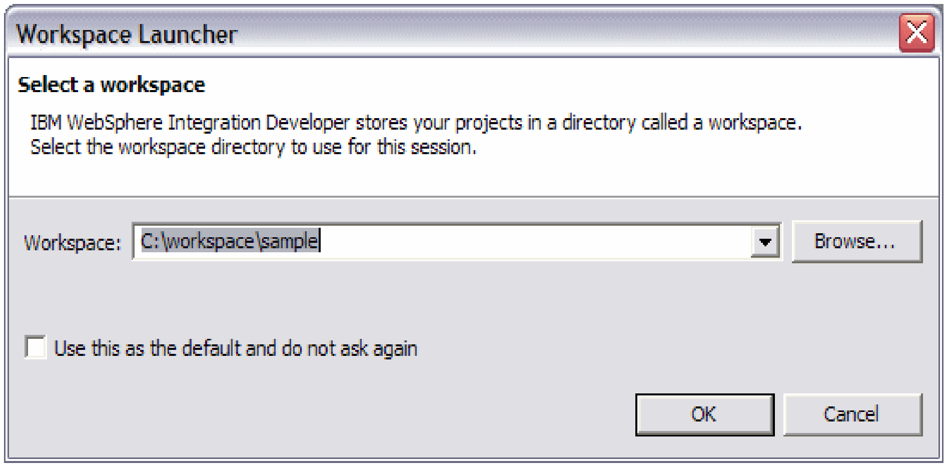


Figure 7-16 Workspace configuration stage

7.2.1 Creating the initial workspace

When WebSphere Integration Developer is launched, you will see a dialog that allows you to specify the workspace location (Figure 7-17).


Workspace Launcher

Select a workspace

IBM WebSphere Integration Developer stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

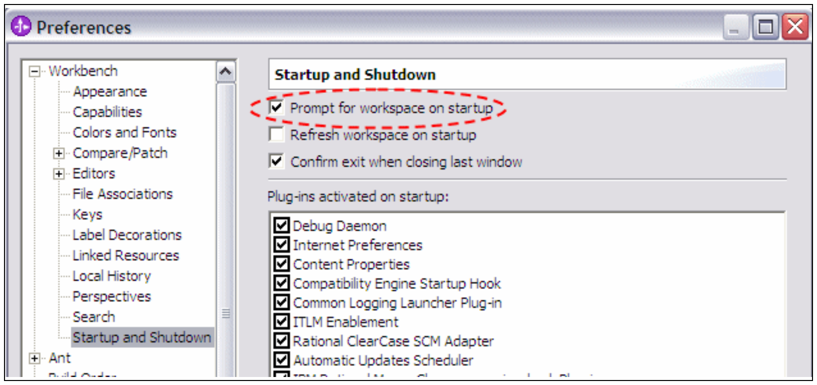
Workspace: C:\workspace\sample

☐ Use this as the default and do not ask again

Figure 7-17 Workspace launcher dialog

It is a good idea to have separate workspaces for projects which belong together. This dialog lets you choose an existing workspace or create a new one. If the directory specified does not exist, a new workspace will be created.

Tip: If you enable the Use this as default and do not ask again checkbox but want the Workspace Launcher to start prompting again, go to **Window** → **Preferences** → **Workbench** → **Startup and Shutdown** and check the option called **Prompt for workspace on startup** (Figure 7-18).



Startup and Shutdown

- [✓] Prompt for workspace on startup
- [ ] Refresh workspace on startup
- [✓] Confirm exit when closing last window

Plug-ins activated on startup:

- [✓] Debug Daemon
- [✓] Internet Preferences
- [✓] Content Properties
- [✓] Compatibility Engine Startup Hook
- [✓] Common Logging Launcher Plug-in
- [✓] ITLM Enablement
- [✓] Rational ClearCase SCM Adapter
- [✓] Automatic Updates Scheduler

Figure 7-18 Prompt for workspace on startup checkbox

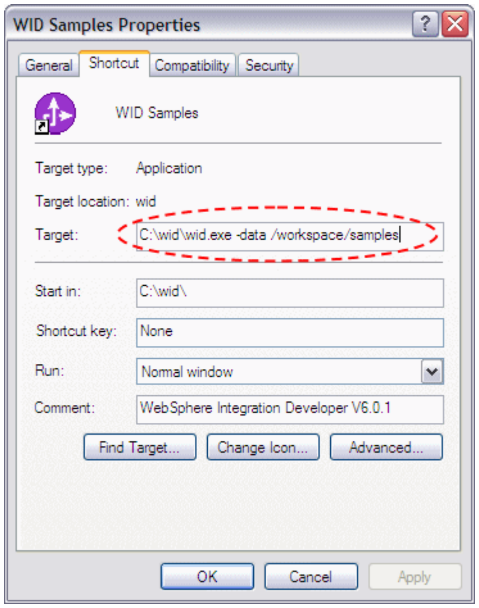
Attention: On Windows systems, because of path length restrictions, keep the workspace path as short as possible.

7.2.2 Configuring desktop shortcuts

A convenient way to launch workspaces is to have dedicated desktop shortcuts, each associated with a different workspace:

1. Create a copy of your WebSphere Integration Developer desktop shortcut.
2. Rename the new shortcut.
3. Right-click the shortcut and select **Properties** from the context menu.
4. In the Target field, after the executable name append `-data` followed by the workspace path, as illustrated in Figure 7-19.

Tip: You can use Java style paths for the workspace location.


WID Samples Properties

General | Shortcut | Compatibility | Security

WID Samples

Target type: Application

Target location: wid

Target: C:\wid\wid.exe -data /workspace/samples

Start in: C:\wid\

Shortcut key: None

Run: Normal window

Comment: WebSphere Integration Developer V6.0.1

Find Target... | Change Icon... | Advanced...

OK | Cancel | Apply

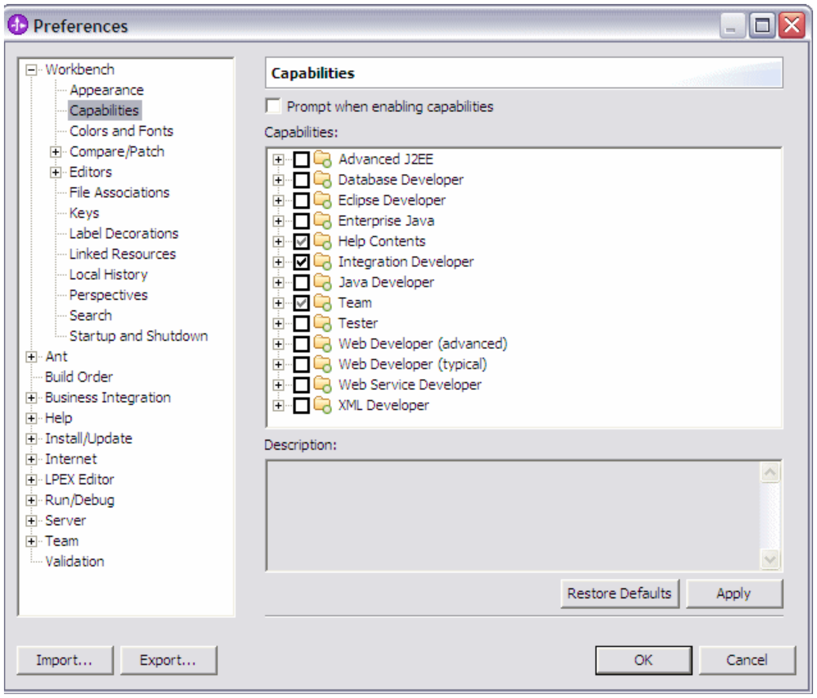
Figure 7-19 Desktop shortcut properties

7.2.3 Capabilities

Capabilities are way to hide certain product features based on the user role. For example, Web services tools and wizards can be hidden by disabling the Web services capability.

By using Capabilities the user interface is simplified by only displaying the features that are most relevant to the current role.

Capabilities that are not enabled can be enabled the first time the feature is accessed. Capabilities can also be enabled from the menu bar by selecting **Window** → **Preferences**, expand **Workbench** and select **Capabilities** (Figure 7-20).



Capabilities:
- ☐ Advanced J2EE
- ☐ Database Developer
- ☐ Eclipse Developer
- ☐ Enterprise Java
- ☑ Help Contents
- ☑ Integration Developer
- ☐ Java Developer
- ☑ Team
- ☐ Tester
- ☐ Web Developer (advanced)
- ☐ Web Developer (typical)
- ☐ Web Service Developer
- ☐ XML Developer

Figure 7-20 Capabilities

Note: Capabilities are associated with a given workspace and it is important to be aware that it might be necessary for you to turn on certain capabilities in order to make sure that the features you typically use during your developing activities are visible to you.

7.3 Interface definition

In preparation to creating mediations you may want to define or import the interfaces your mediation modules will reference and expose (Figure 7-21).

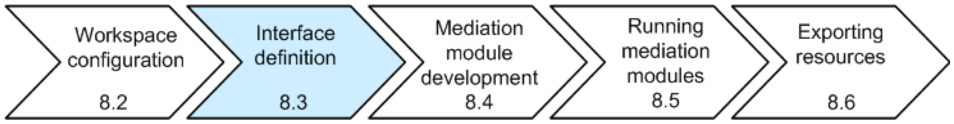


Figure 7-21 Interface definition stage

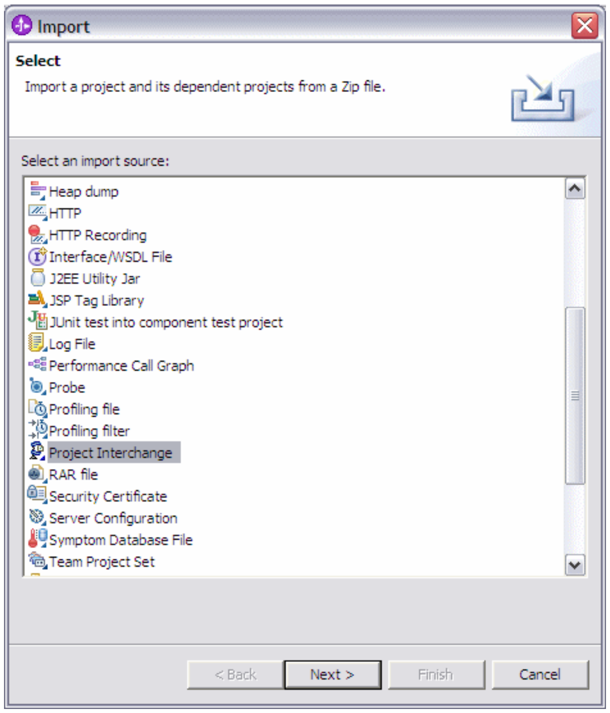
Although the tooling is not prescriptive about the order in which these common tasks are performed, you should have your interfaces defined before you move on to developing mediation modules.

The interfaces, along with the data types they refer to, may already be defined, in which case you can *import* them as a shared library into your workspace.

This section describes the basics of importing workspace resources using the Project Interchange file format, working with shared libraries, and the tools and editors used to define interfaces and business objects (data types).

7.3.1 Importing a Project Interchange file

1. From the menu bar select **File** → **Import**
2. In the Import dialog box select **Project Interchange** (Figure 7-22).


Import

Select

Import a project and its dependent projects from a Zip file.

Select an import source:

- Heap dump
- HTTP
- HTTP Recording
- Interface/WSDL File
- J2EE Utility Jar
- JSP Tag Library
- JUnit test into component test project
- Log File
- Performance Call Graph
- Probe
- Profiling file
- Profiling filter
- Project Interchange
- RAR file
- Security Certificate
- Server Configuration
- Symptom Database File
- Team Project Set

Figure 7-22 Choosing Project Interchange import source

3. Click **Next**.
4. On the Import Project Interchange Contents dialog, click the **Browse** button and navigate to the Project Interchange zip file's location in the file system.
5. Select the projects you wish to import and click **Finish** (Figure 7-23).

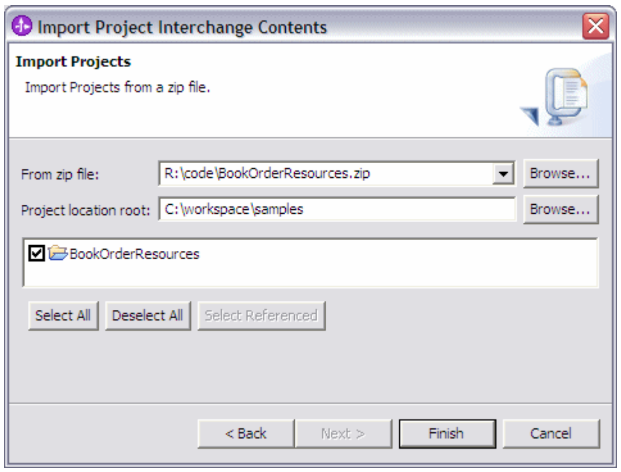


Figure 7-23 Importing Project Interchange contents

7.3.2 Working with shared libraries

A shared library is a special type of Business Integration project that holds resources shared between modules.

Note: Only business objects and interfaces can exist in a WebSphere Enterprise Service Bus shared library. The mapping folder only applies to WebSphere Process Server projects. An additional Web Service Bindings folder is created automatically if you store WSDL files in a library.

Creating a new library

1. To create a new shared library right-click the Business Integration view and select **New** → **Library** from the context menu.
2. Name the library and click **Finish**.

Adding libraries to a mediation module dependency list

1. To add a dependent library to a module double click on the modules' top level project folder to open the Module Dependency editor
2. Click **Add** on the libraries section and select the library to add (Figure 7-24).
3. Click **OK**.

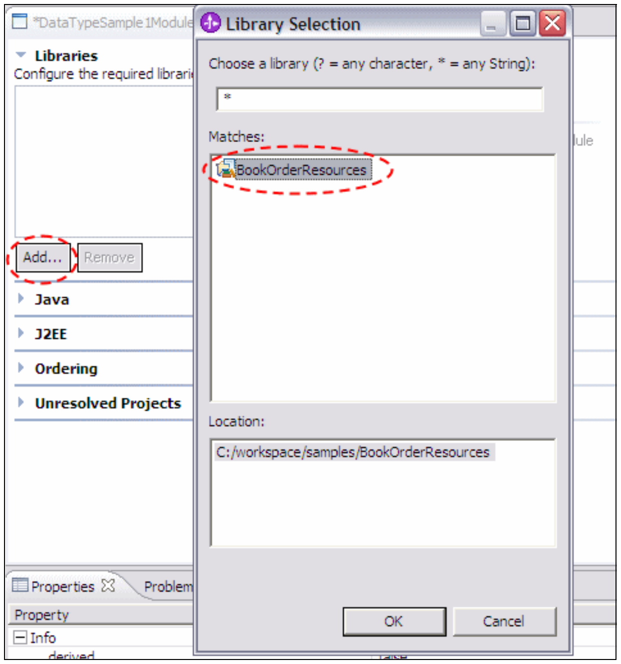


Figure 7-24 Adding a library to a module dependency list

4. The library is now listed under Configured libraries. Click on the library and verify that the Deploy with Module checkbox under Advanced is checked (Figure 7-25). At runtime a library is not shared but deployed with each module that depends on it.

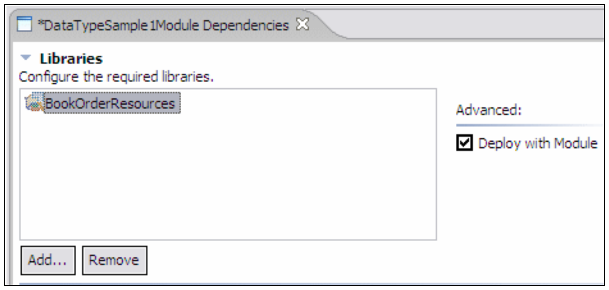

*DataTypeSample1Module Dependencies
Libraries
Configure the required libraries.
BookOrderResources
Advanced:
☑ Deploy with Module
Add...
Remove

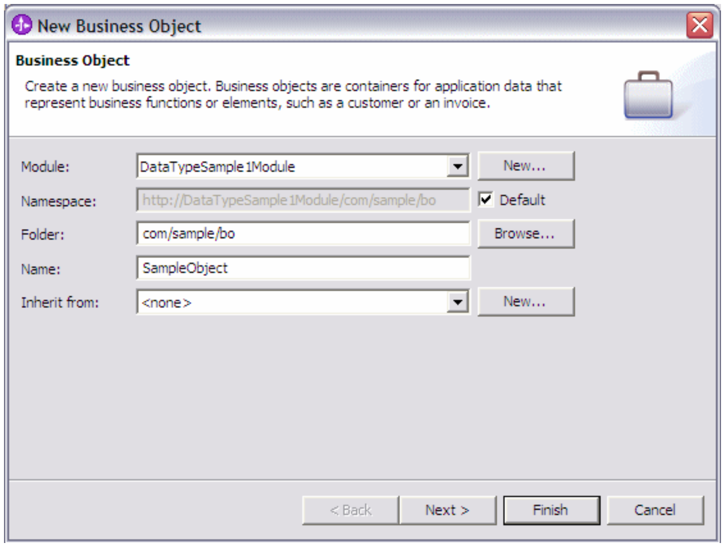
Figure 7-25 Library configured and deployed with module

7.3.3 Modeling business objects

Business objects (data types) can be created in mediation modules or shared libraries. If the business object is to be shared between modules then it should be created in a library.

Creating a business object

1. To create a new business object right-click your module or library and select **New → Business Object**.
2. Specify the business object name, verify the module name and optionally define a folder name. If the Default checkbox by the Namespace field is checked the folder name will become part of the new object's namespace (Figure 7-26).


New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module:	DataTypeSample1Module	New...
Namespace:	http://DataTypeSample1Module/com/sample/bo	✓ Default
Folder:	com/sample/bo	Browse...
Name:	SampleObject	
Inherit from:	<none>	New...

Figure 7-26 Defining the new business object

3. Click **Finish.** A new business object opens in the Business Object editor. Business objects are created in the Data Types folder in your module or library (Figure 7-27).

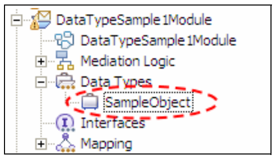


Figure 7-27 New business object

Adding business object attributes

1. Right-click the object and select **Add attribute** from the context menu. Alternatively click the **Add an attribute to a business object** button on the business object editor toolbar (Figure 7-28).

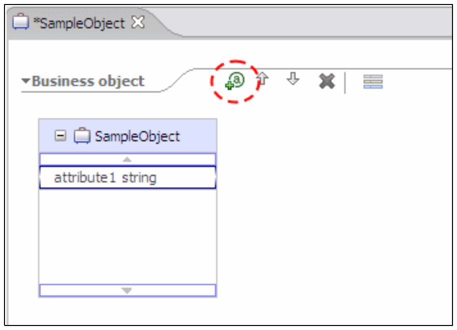


Figure 7-28 Add attribute tool

2. Attributes are created with a default name of attribute1 and a default type of string. Add as many attributes as you need. The attribute names will keep incrementing to attribute2, and so forth.
3. Rename the attributes by overtyping their names.

Business object attribute types

1. To change attribute types, click on its type field (initially string) and select a new type from the context list (Figure 7-29).

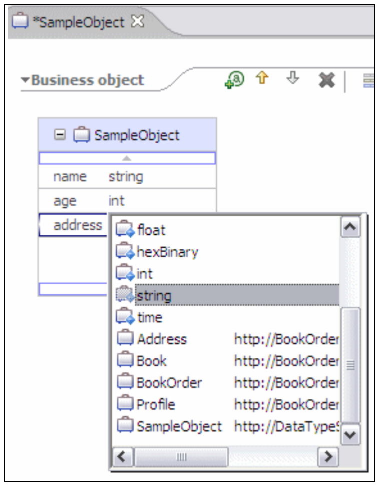


Figure 7-29 Changing an attribute's type

2. From the type list you can choose simple types like int and float, or any complex data type (business object) defined in the module or a library configured in the Module Dependency editor. A complex type field is shown as a link to the actual business object (Figure 7-30).

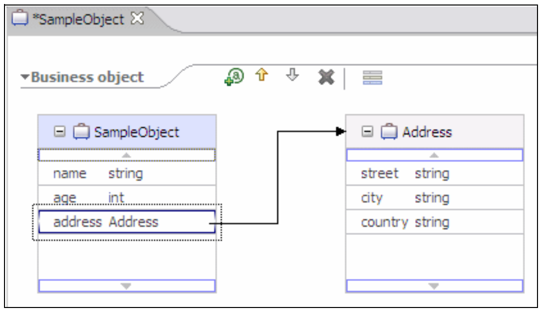


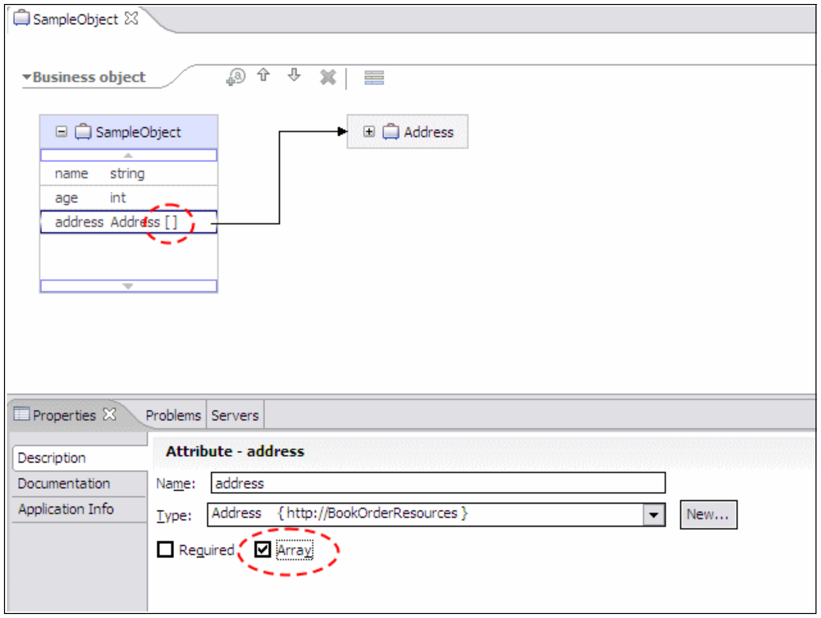
Figure 7-30 Complex attribute type

Business object attribute properties

1. To inspect and modify attribute properties select the attribute and click the Properties view.

   From the Properties view you can change the attribute’s name and type. You can also specify if the attribute’s value is mandatory and make the attribute an *array*. Click on the Array checkbox (Figure 7-31) to make the attribute an array.

   Depending on the attribute, further type checking can be performed. For example, ranges and enumerations can be defined for integers and strings.



name	string
age	int
address	Address [ ]

Attribute - address

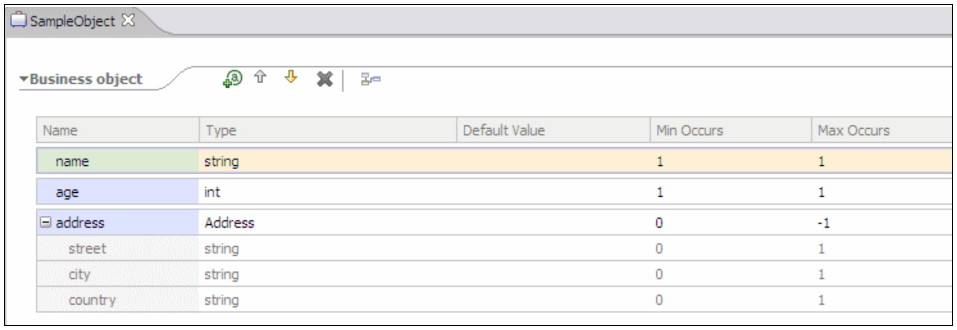
Name: address

Type: Address { http://BookOrderResources }

Required [✓ Array]

Figure 7-31 List of addresses

Tip: You can view basic properties of all attributes by clicking the **Table view** button in the business object editor toolbar (Figure 7-32).



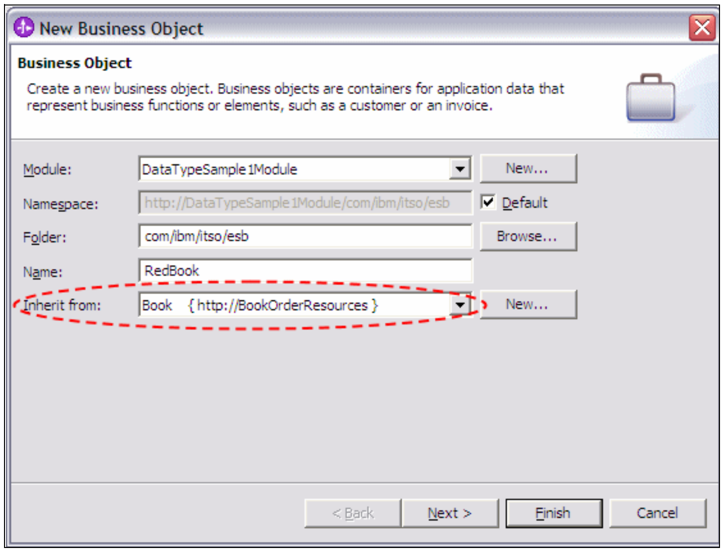
Name	Type	Default Value	Min Occurs	Max Occurs
name	string		1	1
age	int		1	1
address	Address		0	-1
street	string		0	1
city	string		0	1
country	string		0	1

Figure 7-32 Table view of a business object

Using supersets

You can define a business object as a *superset* of another. To do this you need to *inherit* your new object from another one. You can define object inheritance at object creation time or later on the object Properties view.

On the New Business Object wizard, use the **Inherit from** drop down list to specify the parent object (Figure 7-33).


New Business Object

Business Object

Create a new business object. Business objects are containers for application data that represent business functions or elements, such as a customer or an invoice.

Module: DataTypeSample1Module

Namespace: http://DataTypeSample1Module/com/ibm/itso/esb ✓ Default

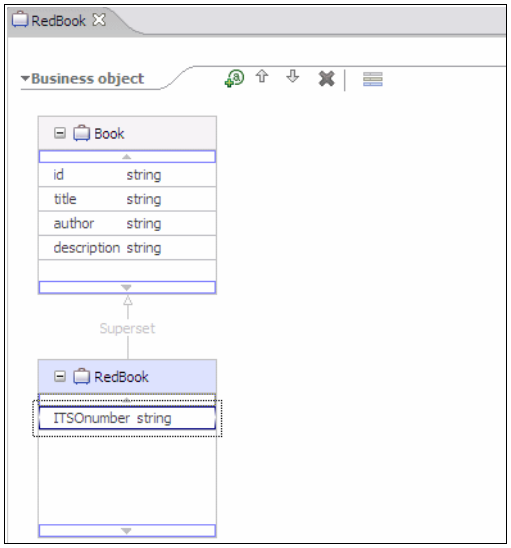
Folder: com/ibm/itso/esb

Name: RedBook

Inherit from: Book {http://BookOrderResources}

Figure 7-33 Creating a superset business object

You can add attributes to specialize the new object. For example the RedBook business object is a superset of Book. It inherits all attributes from the Book business object and defines the RedBook specific ITSOnumber (Figure 7-34).



Book

id	string
title	string
author	string
description	string

RedBook

ITSOnumber	string

Figure 7-34 Business object inheritance

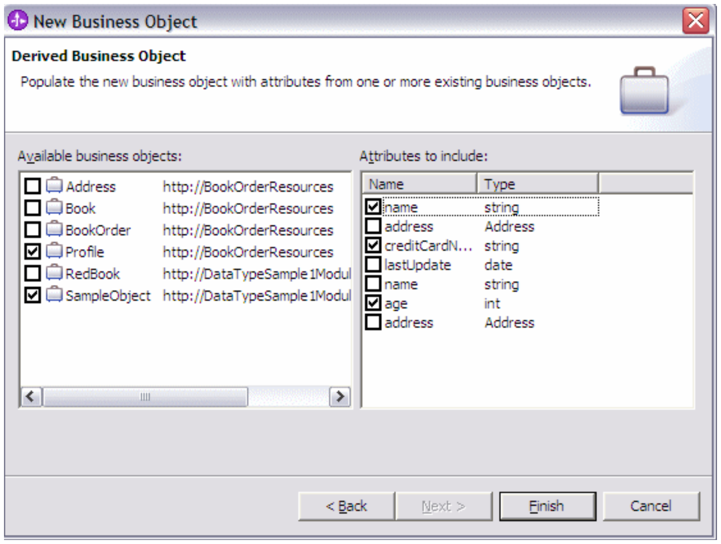
Deriving business object attributes

Another way to reuse attributes from existing business objects is to specify *derived* attributes in your new object. This method, rather than inheriting from an existing object, lets you copy selected attributes from one or more objects into your object.

This is a quick way to copy attributes from other objects when creating your new object.

1. To do this click **Next** (in the previous examples we clicked **Finish**) after completing the first page on the new business object wizard.

   In this example (Figure 7-35) we are creating a new object which includes the name and creditCardNum attributes from the Profile object in the BookOrderResources library and the age attribute from the SampleObject created in “Creating a business object” on page 176.



Derived Business Object

Populate the new business object with attributes from one or more existing business objects.

Available business objects:

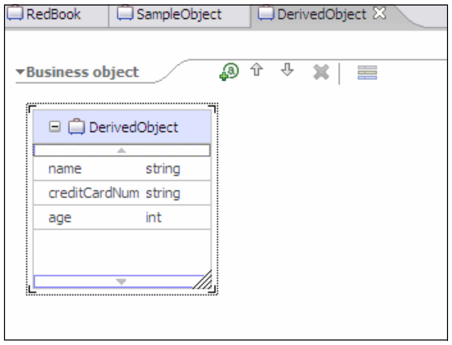
☐	Address	http://BookOrderResources
☐	Book	http://BookOrderResources
☐	BookOrder	http://BookOrderResources
☑	Profile	http://BookOrderResources
☐	RedBook	http://DataTypeSample1Modul
☑	SampleObject	http://DataTypeSample1Modul

Attributes to include:

	Name	Type
☑	name	string
☐	address	Address
☑	creditCardN...	string
☐	lastUpdate	date
☐	name	string
☑	age	int
☐	address	Address

Figure 7-35 Deriving attributes

2. Click **Finish** and the new object will be created (Figure 7-36). You can add more attributes to it as normal.



name	string
creditCardNum	string
age	int

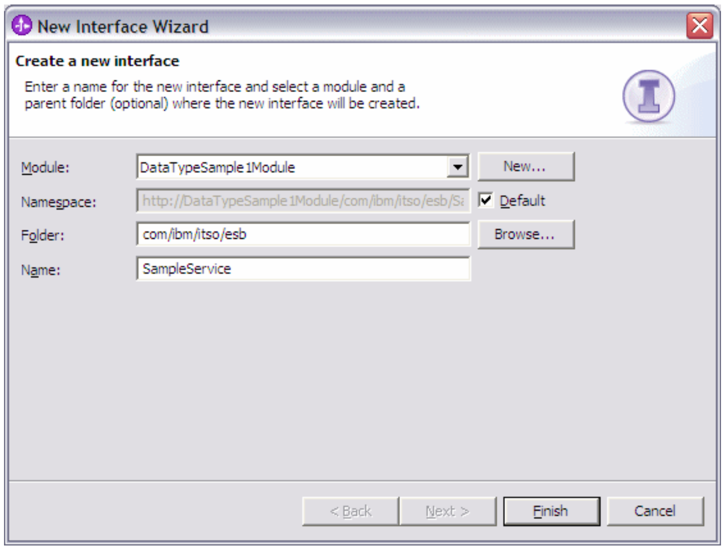
Figure 7-36 Derived business object

Important: Deriving business object attributes is a way to populate the new business object with attributes from one or more existing business objects. There is no connection between the new object and the objects used to copy attributes from.

7.3.4 Defining interfaces

Interfaces can be created in mediation modules or shared libraries. If the interface is to be shared between modules then it should be created in a shared library.

1. To create a new interface right-click the module or library and select **New** → **Interface** from the context menu.
2. Name the interface and optionally specify a folder which will be used as part of the namespace if default namespaces are used (Figure 7-37).


New Interface Wizard

Create a new interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module: DataTypeSample1Module

Namespace: http://DataTypeSample1Module/com/ibm/itso/esb/Sa [✓ Default]

Folder: com/ibm/itso/esb

Name: SampleService

Figure 7-37 New interface wizard

3. Click **Finish** and the new interface is created in the Interfaces folder of your module or library. The Interface editor opens automatically and you will use its toolbar to add operations and operation parameters (Figure 7-38).

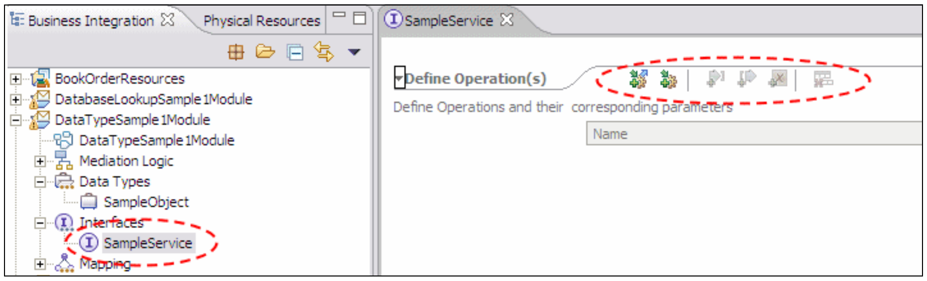


Figure 7-38 New interface in interface editor

Adding one way operations

1. Use the **Add One Way Operation** button in the Interface editor toolbar to add one way operations (Figure 7-39).
2. Rename the operation as required.

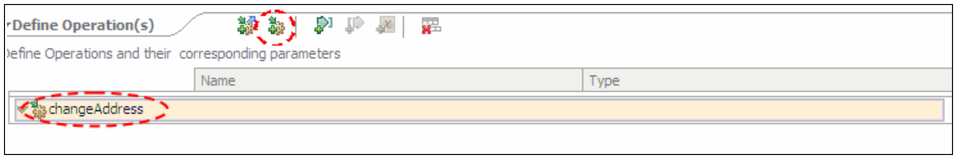
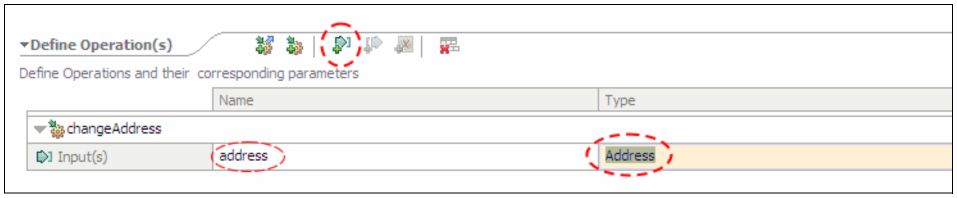


Figure 7-39 Add one way operation

Adding operation input parameters

1. Use the **Add Input** button in the Interface editor toolbar to add input parameters (Figure 7-40).
2. Change the parameter name by typing over it and set the parameter type by clicking on it and selecting a type from the list.

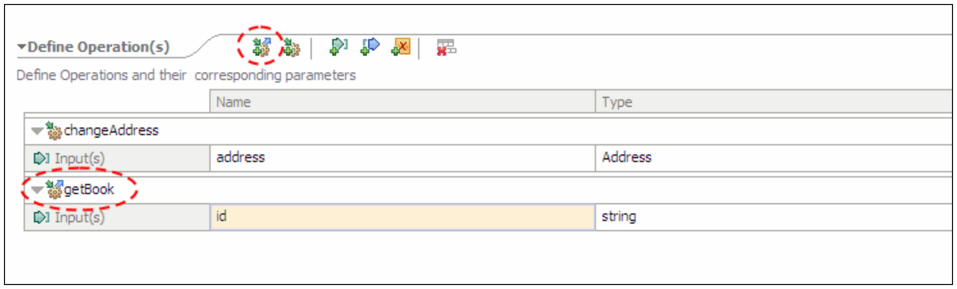


Name	Type
changeAddress	
Input(s) address	Address

Figure 7-40 Add operation input parameter

Adding request response operations

1. Use the **Add Request Response Operation** button (Figure 7-41).
2. Rename the operation by typing over its name.
3. Add required input parameters as normal.



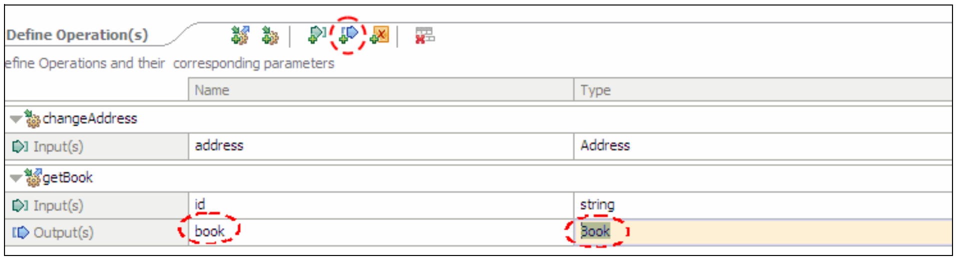
Name	Type
changeAddress	
Input(s) address	Address
getBook	
Input(s) id	string

Figure 7-41 Adding a request response operation

Adding operation output parameters

1. Add an output parameter using the **Add Output** button (Figure 7-42).

2. Rename the output parameter and select its type from the list of available types.


Define Operation(s)

efine Operations and their corresponding parameters

	Name	Type
changeAddress		
Input(s)	address	Address
getBook		
Input(s)	id	string
Output(s)	book	Book

Figure 7-42 Add operation output parameter

Note: Use the **Delete** tool to remove inputs, outputs and operations.

7.4 Mediation module development

This stage is where the actual mediation module development takes place (Figure 7-43).

In this section we discuss mediation modules, mediation flow components and the mediation flow editor. We also explain imports and exports.

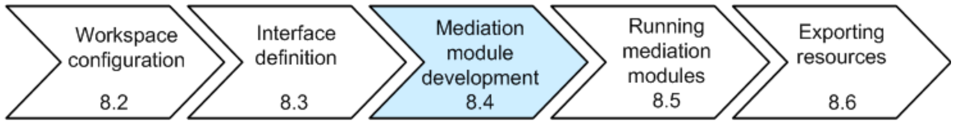
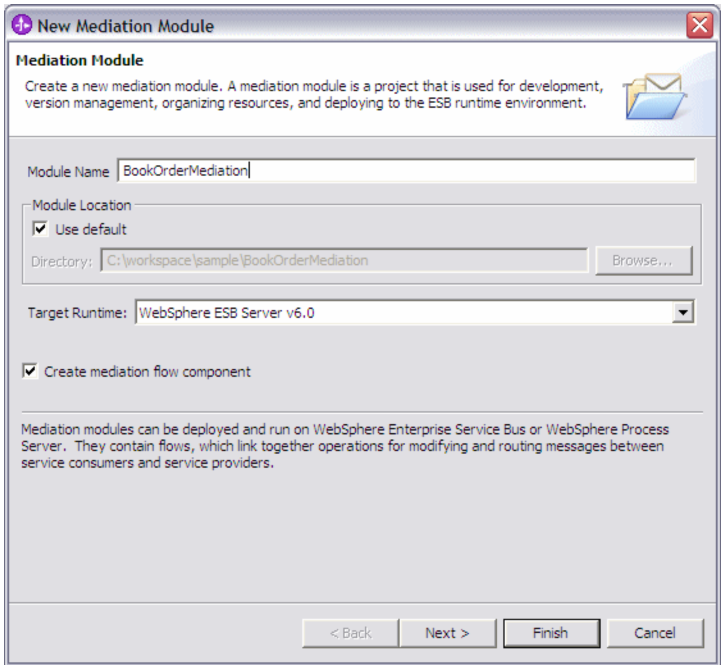

Workspace configuration 8.2
Interface definition 8.3
Mediation module development 8.4
Running mediation modules 8.5
Exporting resources 8.6

Figure 7-43 Mediation module development stage

7.4.1 Creating a new mediation module

1. Right-click the Business Integration view and select **New** → **Mediation Module**.
2. On the first page of New Mediation Module wizard give the module a name and leave the default values for all other items, making sure the target runtime is set to **WebSphere ESB Server** and a mediation flow component is created (Figure 7-44). Click **Next**.



Create a new mediation module. A mediation module is a project that is used for development, version management, organizing resources, and deploying to the ESB runtime environment.

Module Name: BookOrderMediation

Module Location
✓ Use default
Directory: C:\workspace\sample\BookOrderMediation

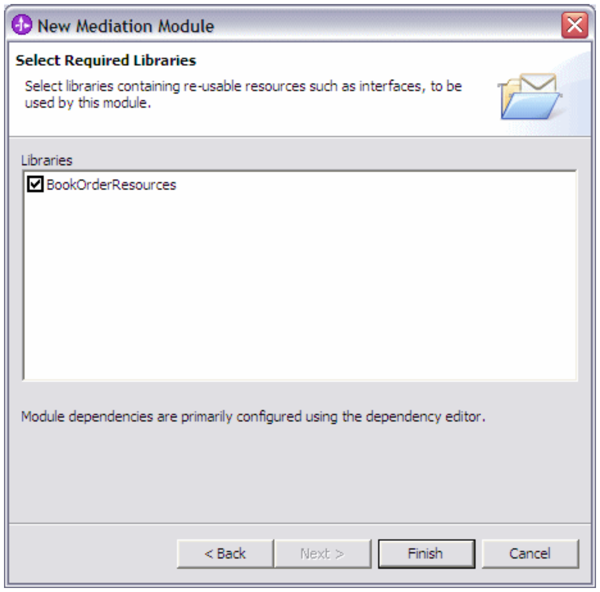
Target Runtime: WebSphere ESB Server v6.0

✓ Create mediation flow component

Mediation modules can be deployed and run on WebSphere Enterprise Service Bus or WebSphere Process Server. They contain flows, which link together operations for modifying and routing messages between service consumers and service providers.

Figure 7-44 New mediation module wizard

3. The last New Mediation Module wizard page (Figure 7-45) lets you specify any number of shared libraries that you want to refer to within your mediation module. Shared libraries included in this page will be deployed as part of the module.


New Mediation Module

Select Required Libraries

Select libraries containing re-usable resources such as interfaces, to be used by this module.

Libraries

☑ BookOrderResources

Module dependencies are primarily configured using the dependency editor.

Figure 7-45 Selecting libraries

4. Click the **Finish** button and the new mediation module will be created and displayed expanded in the Business Integration view (Figure 7-46).

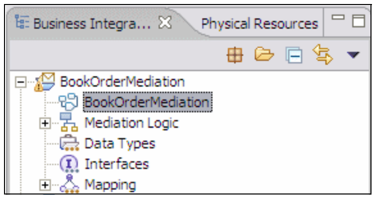


Figure 7-46 New mediation module

5. Double click on the module project (the top level folder) to open the Module Dependency editor (Figure 7-47). Under Libraries you should see any library that was added to the module at creation time.

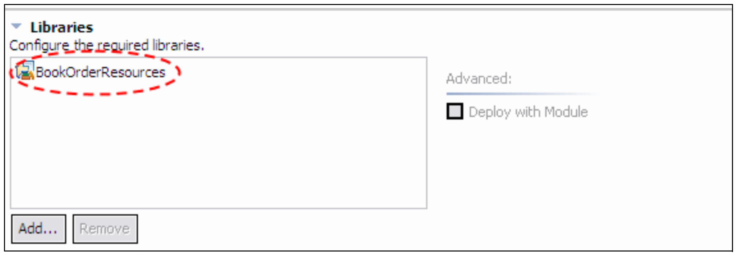


Figure 7-47 Module dependencies

7.4.2 Creating a new mediation flow component

Mediation flow components are shown in the mediation module assembly diagram and a single mediation flow component is created by default when creating the module as seen in 7.4.1, “Creating a new mediation module” on page 185.

To manually create a mediation flow component, in the assembly diagram editor click the **Mediation Flow** tool and then click the canvas to drop the new component (Figure 7-48).

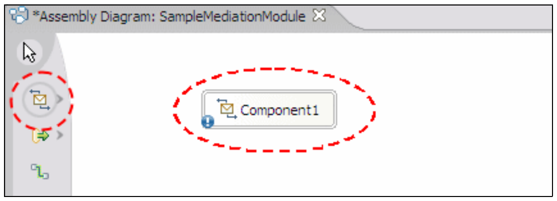


Figure 7-48 New mediation flow component

Note: Only one mediation flow component can be added and implemented in the mediation module.

Adding an interface to a mediation flow component

1. Select the mediation flow component, the corners on the component become blue dots and a bubble toolbar appears over it. Select the **Add Interface** tool. Alternatively right click on the mediation flow component and select **Add** → **Interface** from the context menu.
2. Select the required interface from the dialog box and click **OK**. The added interface will be displayed on the left side of the mediation flow component (Figure 7-49).

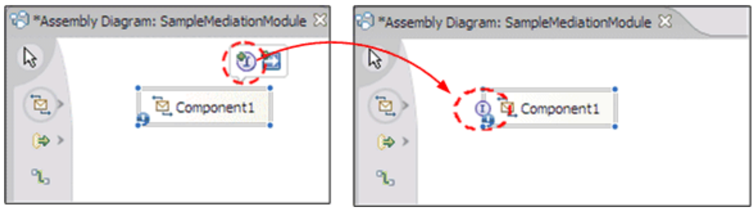

*Assembly Diagram: SampleMediationModule
Component1
*Assembly Diagram: SampleMediationModule
Component1

Figure 7-49 Adding interface to mediation flow component

Adding a reference to a mediation flow component

1. This is the same as adding an interface but selecting the **Add Reference** tool from the bubble toolbar or **Add** → **Reference** from the context menu (Figure 7-50).
2. Choose the interface your reference is to be associated with and click **OK**. The reference is added to the right side of the mediation flow component.

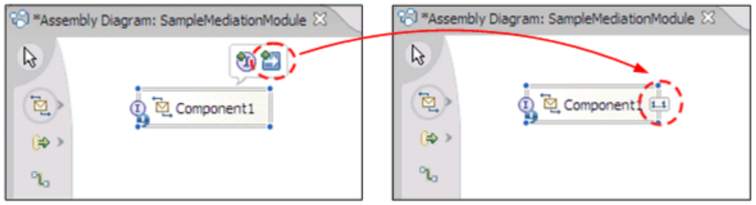

*Assembly Diagram: SampleMediationModule
Component1
*Assembly Diagram: SampleMediationModule
Component1

Figure 7-50 Add reference to mediation flow component

Implementing mediation flow components

1. A mediation flow component is created with no implementation. To implement the mediation flow right click on the component and select **Generate Implementation** from the context menu (Figure 7-51).

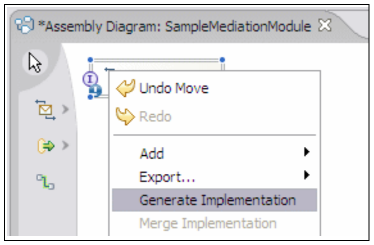


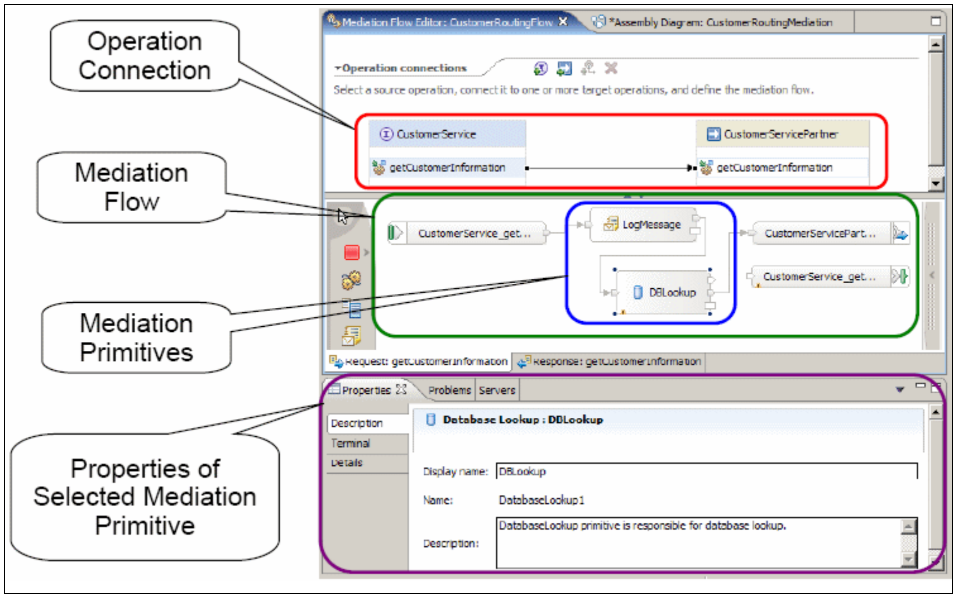
Figure 7-51 Generating mediation flow implementation

2. Select the folder where to generate the implementation and click **OK**. The Mediation Flow editor opens.

The Mediation Flow editor

Figure 7-52 shows the Mediation Flow editor with its three main panels, the Operation connections, the mediation flow and the Properties view.

We use this editor to connect operations, add mediation primitives to the mediation flow and wire mediation primitive terminals with requests and responses.


Operation Connection
Mediation Flow
Mediation Primitives
Properties of Selected Mediation Primitive
Mediation Flow Editor: CustomerRoutingFlow
*Assembly Diagram: CustomerRoutingMediation
Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
CustomerService
CustomerServicePartner
getCustomerInformation
getCustomerInformation
CustomerService_get...
LogMessage
DBLookup
CustomerServicePart...
CustomerService_get...
Request: getCustomerInformation
Response: getCustomerInformation
Properties
Problems
Servers
Description
Terminal
Details
Database Lookup : DBLookup

Display name: DBLookup

Name: DatabaseLookup1

Description: DatabaseLookup primitive is responsible for database lookup.

Figure 7-52 The Mediation Flow Editor

Wiring interface/reference operations

To wire operations, on the Operation connections panel, drag the source operation on the interface to the target operation on the reference (Figure 7-53).

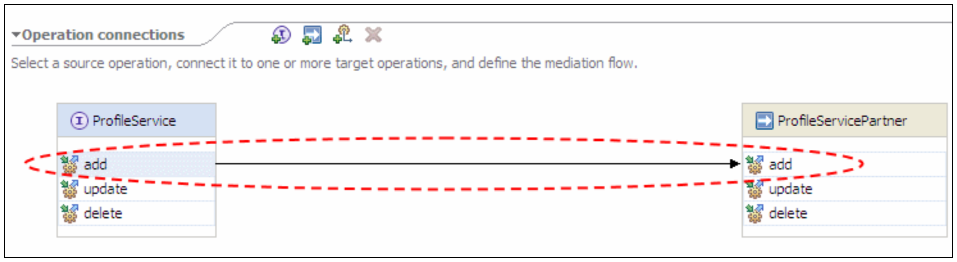

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
ProfileService
add
update
delete
ProfileServicePartner
add
update
delete

Figure 7-53 Connecting operations

Adding mediation primitives to the canvas

Use the toolbar on the left hand side of the mediation flow pane to select the required primitive and drop it into the canvas (Figure 7-54).

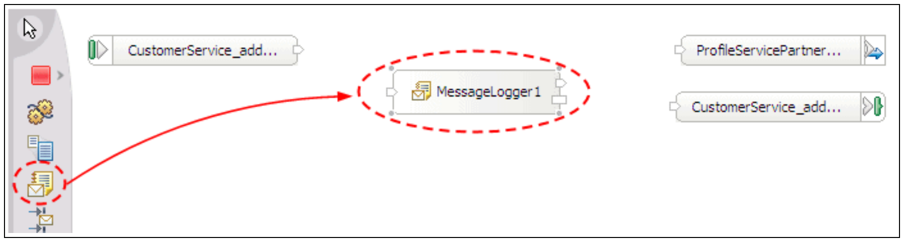


Figure 7-54 Adding a mediation flow primitive

Wiring a mediation flow (request/response)

Wiring the mediation flow defines the sequence in which mediation primitives are executed and assigns their terminal's message type.

Request flow

1. First we add a mediation primitive to the canvas. In this case we are adding a Message Logger. At this point the terminals on the primitive have no assigned type (Figure 7-55).

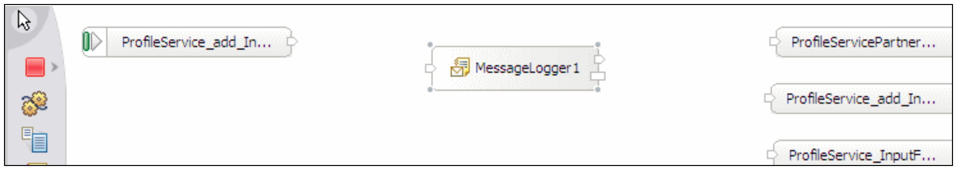


Figure 7-55 MessageLogger primitive added

2. Next we wire the input node of your mediation flow to the input terminal of your mediation primitive. The input node represents the entry point to the mediation flow component on the request flow. At this point the input terminal is assigned the message type (Figure 7-56).

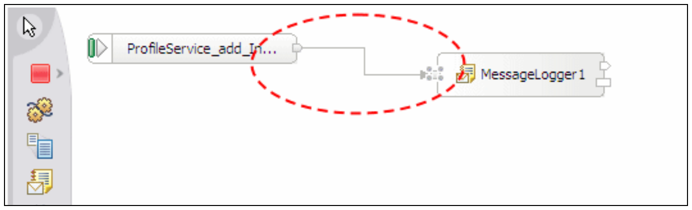


Figure 7-56 Wiring the input terminal

3. Now wire the output terminal from the mediation primitive to the callout node (Figure 7-57). The callout node represents the service provider. At this point the output terminal is assigned the message type.

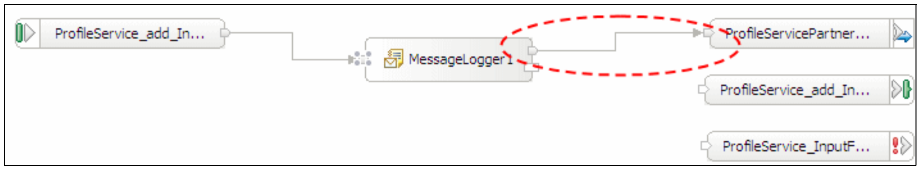


Figure 7-57 Wiring the output terminal

Response flow

1. Click on the **Response** tab on the mediation flow pane to wire the response flow.
2. Connect the callout response node to the mediation primitive input terminal. The callout response node represents the entry point to the mediation flow component on the response flow.
3. Connect the mediation primitive's output terminal to the input response node. The input response node represents the service consumer. All terminals are assigned a message type once they are wired (Figure 7-58).

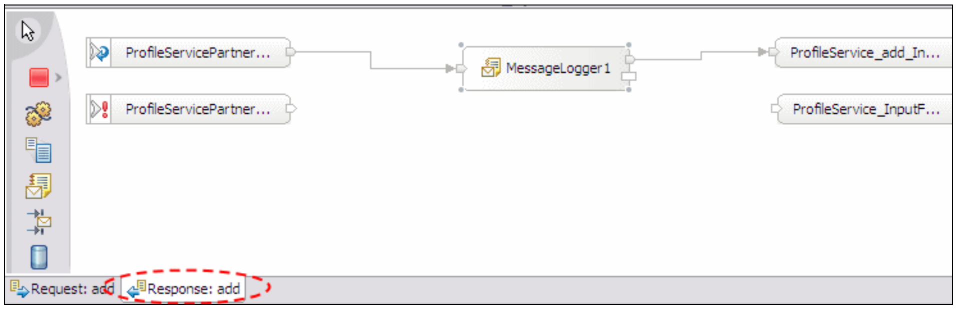


Figure 7-58 Wiring the response flow

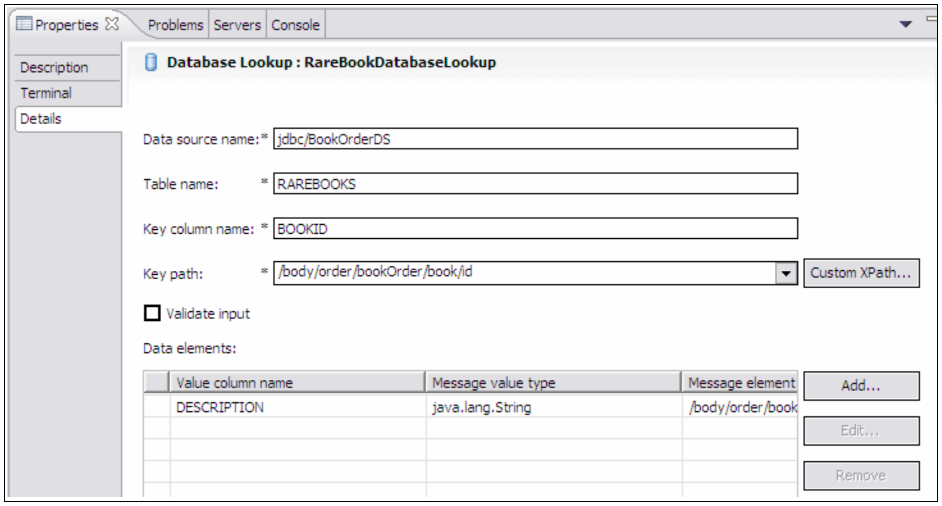
Mediation primitive properties

Use the Properties view to change a mediation primitive's properties.

You can use the Description tab to change a mediation primitive's display name and description. The Terminal tab lets you change the message type for the in, out and fail terminals.

Use the Details tab to change properties that are specific to the mediation primitive type, for example, for an XSLT mediation primitive you use the Details tab of the Properties view to assign the message root, edit and regenerate XSL, or pick an existing XSL file.

In the case of a database lookup mediation primitive, use the Details tab to specify the data source name, table name, key column name and the location in the message of the key value (Figure 7-59).



Database Lookup : RareBookDatabaseLookup

Data source name:* jdbc/BookOrderDS

Table name: * RAREBOOKS

Key column name: * BOOKID

Key path: * /body/order/bookOrder/book/id

☐ Validate input

Data elements:

Value column name	Message value type	Message element
DESCRIPTION	java.lang.String	/body/order/book

Figure 7-59 Database lookup mediation primitive's properties

7.4.3 Working with exports and imports

This section explains the basics of exports, imports, their interfaces, references and bindings.

Creating exports

Exports can be generated complete with interface and bindings from an existing SCA component, such as a mediation flow component.

However, you will not always have a mediation flow component to export. Maybe you want a service consumer request to pass through the bus for added flexibility, like protocol and transport mapping, without performing any mediation logic.

For those cases, you need to create and define the export manually.

1. Select the **Export** tool from the Assembly Diagram editor toolbar (Figure 7-60).

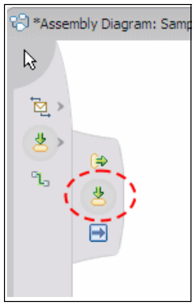


Figure 7-60 Export tool

2. Click on the Assembly Diagram canvas to create the Export component (Figure 7-61).

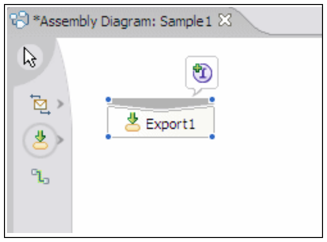


Figure 7-61 Export created

3. The export will be selected and the Add Interface tool will be visible, if not make sure the export is selected. Click the **Add Interface** tool.

> **Tip:** You can always right-click the export and select **Add Interface**.

4. Now you need to decide which binding to use.
   a. Right-click the export and hover the mouse over the **Generate Binding** context menu item.
   b. Select a binding, for example **SCA Binding**.
5. At this point your export component has an interface and can be invoked over SCA.

Creating exports from mediation flow components

To expose a mediation flow component as a service that can be invoked by other modules and clients you must *export* it.

1. Right-click the component and select **Export** then select your binding choice from the context menu (Figure 7-62). This will add the export to the assembly diagram and wire it to the mediation flow component.

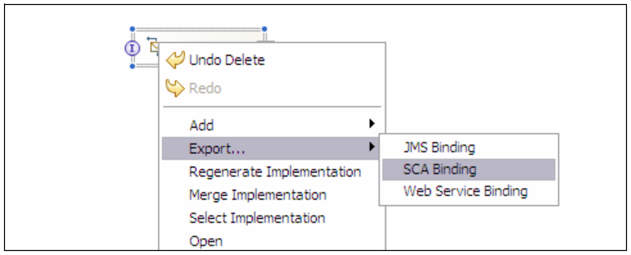


Figure 7-62 Generating export with SCA bindings

2. The export is generated on the assembly diagram and can be also located in the Business Integration view (Figure 7-63).
3. At this point your mediation flow component can be invoked over SCA.

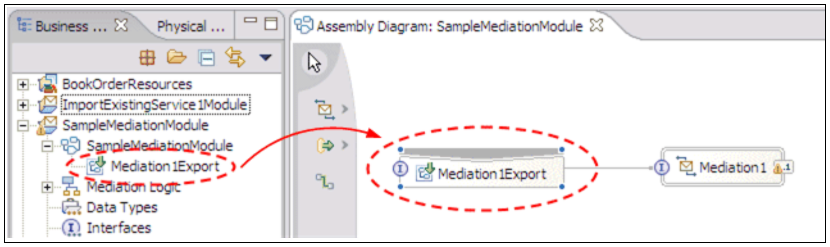


Figure 7-63 Generating mediation component export

Tip: The export listed in the Business Integration view can be dragged and dropped into the canvas of another module's assembly diagram creating an import.

Creating imports

An import represents a service outside of our module.

Imports can be created automatically from another module's export or a WSDL service definition, however we can also manually create imports in our assembly diagram.

1. Select the **Import** tool from the assembly diagram toolbar (Figure 7-64).

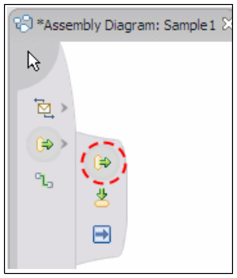


Figure 7-64 Import tool

2. Click on the assembly diagram to create the import (Figure 7-65).

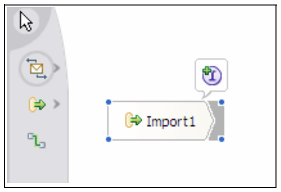


Figure 7-65 Import created in assembly diagram

3. Add an interface to the import by using the **Add Interface** tool or the context menu.
4. Generate bindings using the context menu:
   a. Right-click the import and hover the mouse over the **Generate Binding** context menu item.
   b. Select a binding, for example **SCA Binding**.

Creating imports from existing services

For the mediation flow component to invoke services, the service providers need to be imported into the assembly diagram as *imports*.

We can create imports automatically by dragging and dropping external exports or WSDL service definitions into our assembly diagrams.

1. Consider a mediation flow component already implementing an interface and wired to an export. This mediation now needs to be associated to a service provider.
2. In the Business Integration view we locate the service provider export (Figure 7-66).

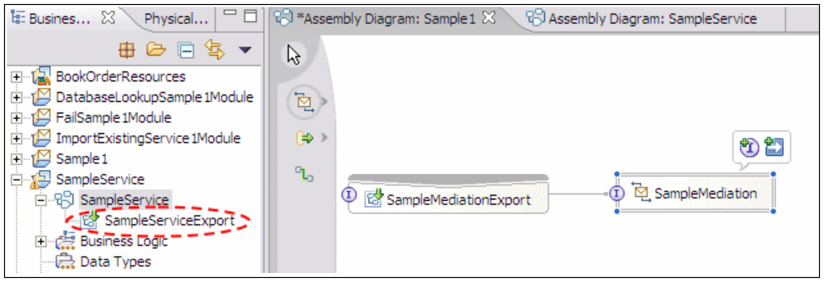


Figure 7-66 Locating the target Export

3. Now we drag this export into our diagram and we choose to create an import with SCA binding at the Component Creation dialog. This action creates an import component in the diagram (Figure 7-67).

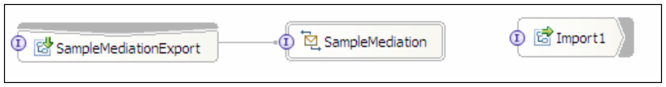


Figure 7-67 Import created

4. Rename this import and drag a wire from the mediation flow component to the import. Note that this action creates a matching reference on the mediation flow component (Figure 7-68).

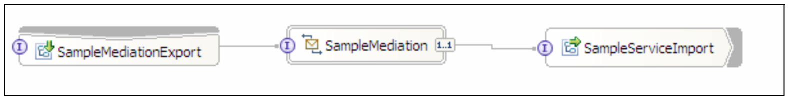


Figure 7-68 Import wired and reference created

5. At this point the basic structure of a mediation flow module is complete.

7.5 Running mediation modules

This section is about building, running and testing mediation modules (Figure 7-69).

We discuss the basics of building and cleaning projects, managing test servers, publishing mediation modules and working with the integration test client.

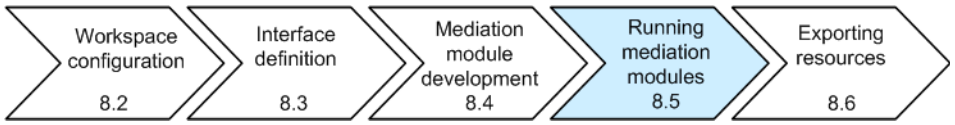

Workspace configuration 8.2
Interface definition 8.3
Mediation module development 8.4
Running mediation modules 8.5
Exporting resources 8.6

Figure 7-69 Mediation module running stage

7.5.1 Building and cleaning projects.

Build automatically

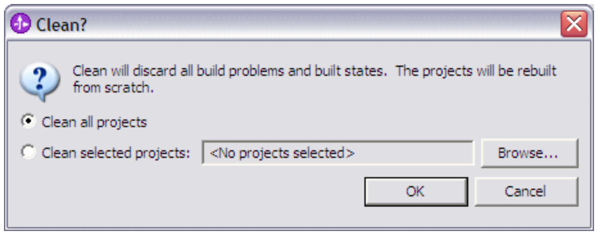
Your development environment is by default configured to build automatically when you make any code changes, check out code from a source repository or import projects from an external source like a Project Interchange file.

This option can be disabled by selecting **Project** → **Build Automatically** from the menu bar.

Cleaning the workspace

You can force a clean build by cleaning your workspace. Performing a workspace *clean* deletes all derived artifacts and staging projects, forcing an automatic build and regeneration to occur.

1. Select **Project** → **Clean...** from the menu bar.
2. Select **Clean all projects** and click **OK** (Figure 7-70).


Clean?

Clean will discard all build problems and built states. The projects will be rebuilt from scratch.

(●) Clean all projects

( ) Clean selected projects: <No projects selected>

Figure 7-70 Cleaning the workspace

7.5.2 Managing test servers

This section discusses the details of the test environment included in IBM WebSphere Integration Developer, how to setup different server configurations and the tools used to publish applications.

We also discuss how the development environment works with the server configurations.

Server profiles

A server profile is a configuration that describes the runtime environment and includes all of the files required by the server at runtime. Creating profiles enables multiple servers to be configured from a single install of WebSphere Enterprise Service Bus.

There are three types of profile:

- Stand alone - This profile hosts mediation modules and is the default WebSphere Enterprise Service Bus profile type.
- Managed node - A managed node performs the same function as a stand-alone server but all administrative tasks for this profile are managed by a deployment manager.
- Deployment manager - Used to administer all managed nodes in a multi-node, multi-machine group, known as a cell.

For information on configuring each of these server profile types see Chapter 5, “Setting up the runtime environment” on page 85.

Server configuration modes

The workspace does not contain the test environment configuration information. Instead, pointers are created to the test environment.

Three WebSphere Integration Developer server configuration modes are supported:

- Local test environments
- Local separate installations of WebSphere Enterprise Service Bus
- Remote test environments

Local test environment (default)

At installation time you have the option of installing the integrated test environment and associated profiles. In our install we created both the WebSphere Process Server and WebSphere Enterprise Service Bus profiles. See 4.3.1, “Installing WebSphere Integration Developer” on page 61.

Each workspace that you start will have a pointer to these profiles. The server profile is independent of the workspace and all you have in the workspace is essentially a pointer to the profile.

This means that you may see applications show up in the test server that are in different workspaces.

Local separate installation

You can also use a separate installation of the runtime as your test environment.

If you have installed a separate instance of WebSphere Enterprise Service Bus or WebSphere Process Server on your local machine, you can create a new workspace server configuration within WebSphere Integration Developer that points at the profile of your choice.

Remote test environment

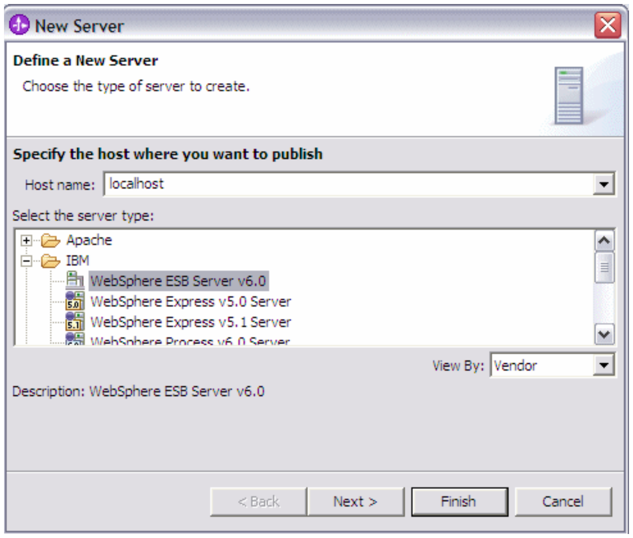
When configuring a test environment, the server can be either a local integrated server or a remote server. Once the server itself is installed and configured, the server definition within WebSphere Integration Developer is very similar for local and remote servers.

Creating a new server configuration

Local test environment

These are the steps to recreate your default local test server configuration.

1. Right-click the Servers view and select **New** → **Server**.
2. Next you will need to define the new server. Leave the host name as `localhost` and select WebSphere ESB Server v6.0 (Figure 7-71).


New Server

Define a New Server

Choose the type of server to create.

Specify the host where you want to publish

Host name: localhost

Select the server type:

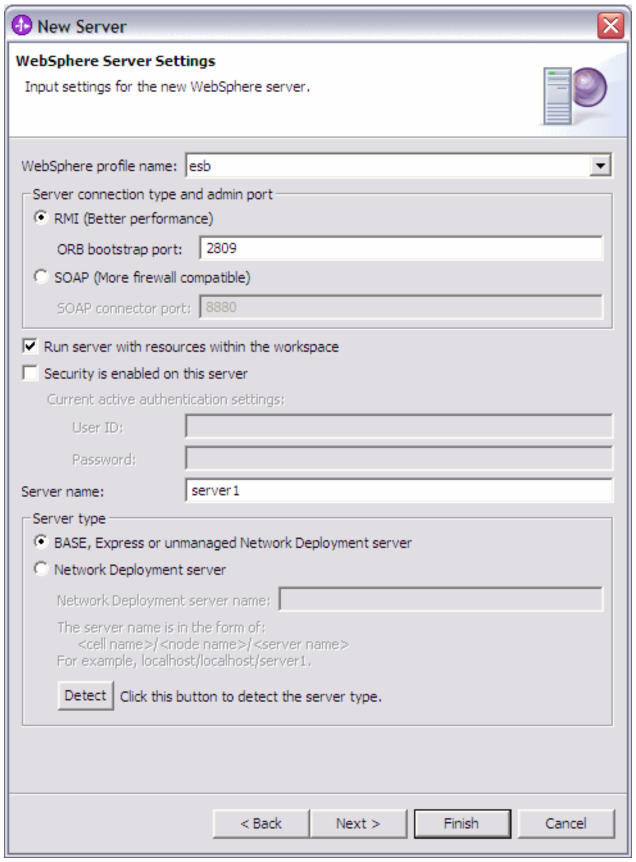
Apache
IBM
WebSphere ESB Server v6.0
WebSphere Express v5.0 Server
WebSphere Express v5.1 Server
WebSphere Process v6.0 Server

View By: Vendor

Description: WebSphere ESB Server v6.0

Figure 7-71 Define new sever

3. Next define the new server settings to use the esb profile (Figure 7-72).


New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: esb

Server connection type and admin port

(●) RMI (Better performance)

ORB bootstrap port: 2809

( ) SOAP (More firewall compatible)

SOAP connector port: 8880

[✓] Run server with resources within the workspace

[ ] Security is enabled on this server

Current active authentication settings:

User ID:

Password:

Server name: server1

Server type

(●) BASE, Express or unmanaged Network Deployment server

( ) Network Deployment server

Network Deployment server name:

The server name is in the form of:
<cell name>/<node name>/<server name>
For example, localhost/localhost/server1.

Detect Click this button to detect the server type.

< Back Next > Finish Cancel

Figure 7-72 New server settings

4. The next step lets you add workspace projects to your new server configuration. Select projects from the available projects pane and click **Add** to add them to the Configured Projects pane. When you are done click **Finish** (Figure 7-73).

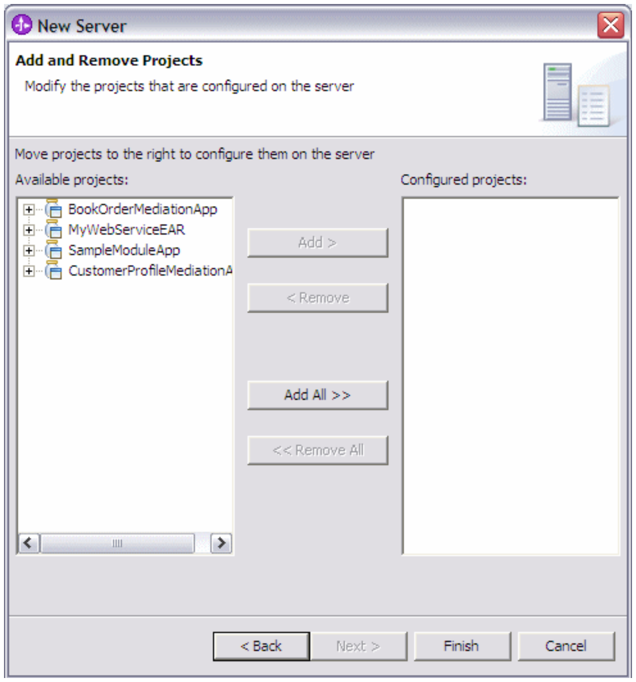
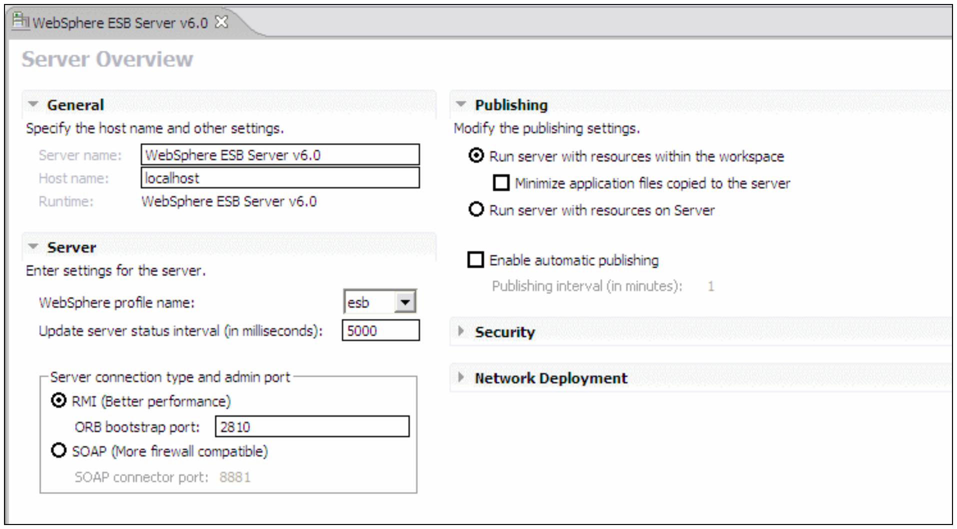


Figure 7-73 Add projects to new server configuration

5. You can now double click on the server configuration to open the Server Configuration editor and review the server settings (Figure 7-74).


WebSphere ESB Server v6.0

Server Overview

General

Specify the host name and other settings.

Server name: WebSphere ESB Server v6.0
Host name: localhost
Runtime: WebSphere ESB Server v6.0

Server

Enter settings for the server.

WebSphere profile name: esb
Update server status interval (in milliseconds): 5000

Server connection type and admin port
[✓ RMI (Better performance)]
ORB bootstrap port: 2810
SOAP (More firewall compatible)
SOAP connector port: 8881

Publishing

Modify the publishing settings.

[✓ Run server with resources within the workspace]
Minimize application files copied to the server
Run server with resources on Server

Enable automatic publishing
Publishing interval (in minutes): 1

Security

Network Deployment

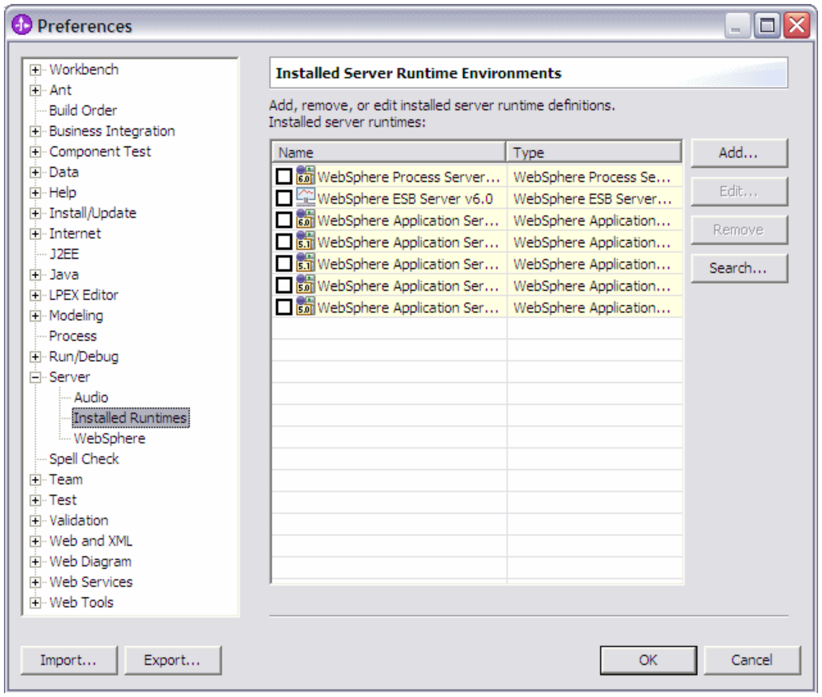
Figure 7-74 Server configuration details

Tip: A running server will continue to run, even after exiting WebSphere Integration Developer. To avoid this, check the box called **Terminate server on workbench shutdown** and save the configuration.

Local separate installation

To add a local installation of WebSphere Enterprise Service Bus to your WebSphere Integration Developer server configurations, first you need to add the server to the Installed Runtimes.

1. From the menu select **Window** → **Preferences**, then expand **Servers** and select **Installed Runtimes** (Figure 7-75).



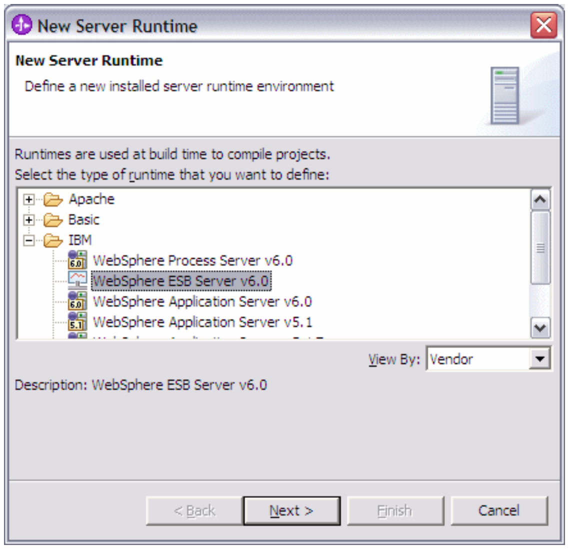
Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.
Installed server runtimes:

Name	Type
☐ WebSphere Process Server...	WebSphere Process Se...
☐ WebSphere ESB Server v6.0	WebSphere ESB Server...
☐ WebSphere Application Ser...	WebSphere Application...
☐ WebSphere Application Ser...	WebSphere Application...
☐ WebSphere Application Ser...	WebSphere Application...
☐ WebSphere Application Ser...	WebSphere Application...
☐ WebSphere Application Ser...	WebSphere Application...

Figure 7-75 Installed runtimes list

2. Click the **Add** button and on the next screen choose **WebSphere ESB Server v6.0**, then click **Next** (Figure 7-76).


New Server Runtime

New Server Runtime

Define a new installed server runtime environment

Runtimes are used at build time to compile projects.
Select the type of runtime that you want to define:

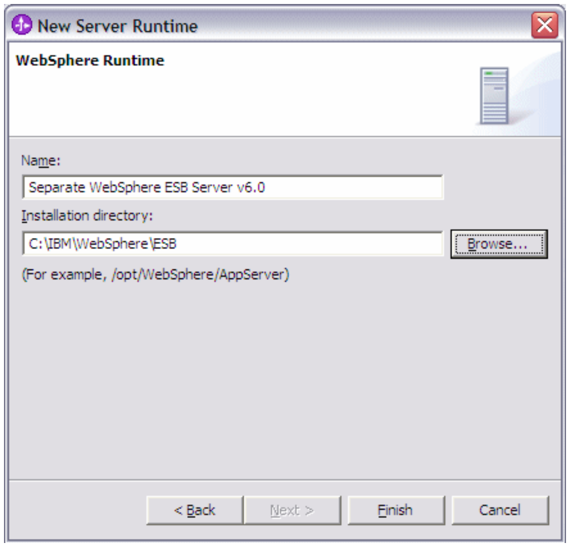
- Apache
- Basic
- IBM
  - WebSphere Process Server v6.0
  - WebSphere ESB Server v6.0
  - WebSphere Application Server v6.0
  - WebSphere Application Server v5.1

View By: Vendor

Description: WebSphere ESB Server v6.0

Figure 7-76 New server runtime

3. On the following screen give the server a name of your choice and locate the separate runtime's install root directory (Figure 7-77).


New Server Runtime

WebSphere Runtime

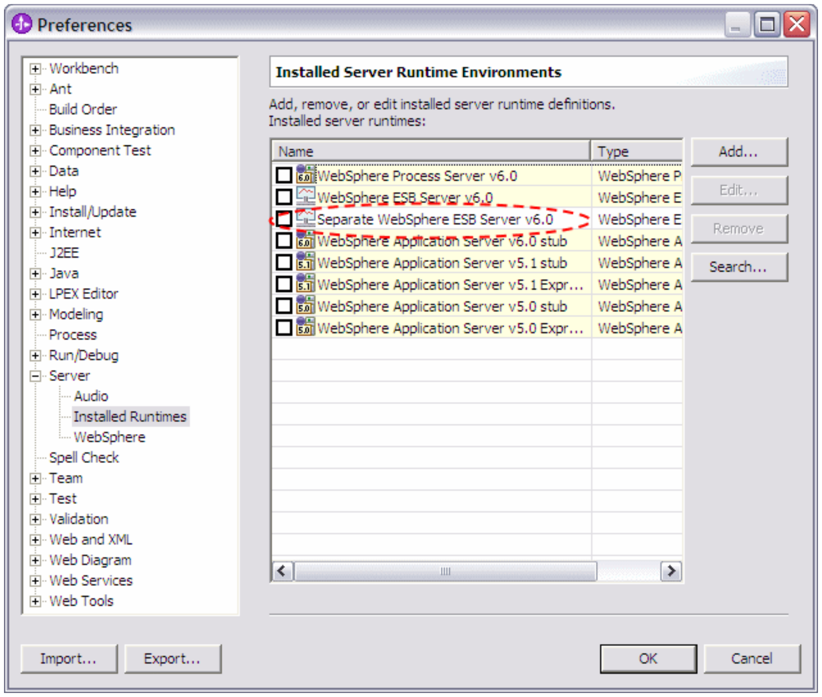
Name:
Separate WebSphere ESB Server v6.0

Installation directory:
C:\IBM\WebSphere\ESB

(For example, /opt/WebSphere/AppServer)

Figure 7-77 New server runtime name and location

4. Click **Finish** and your new server runtime will be part of the installed runtime environments (Figure 7-78).



Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.
Installed server runtimes:

Name	Type
WebSphere Process Server v6.0	WebSphere P
WebSphere ESB Server v6.0	WebSphere E
Separate WebSphere ESB Server v6.0	WebSphere E
WebSphere Application Server v6.0 stub	WebSphere A
WebSphere Application Server v5.1 stub	WebSphere A
WebSphere Application Server v5.1 Expr...	WebSphere A
WebSphere Application Server v5.0 stub	WebSphere A
WebSphere Application Server v5.0 Expr...	WebSphere A

Figure 7-78 New runtime added

5. Now you can repeat the steps as for adding a new local server configuration with the following differences:

   While defining a new server, when you select WebSphere ESB Server v6.0 as the server type, now you have a choice of runtimes, including the one just added to the installed server runtime environments (Figure 7-79).

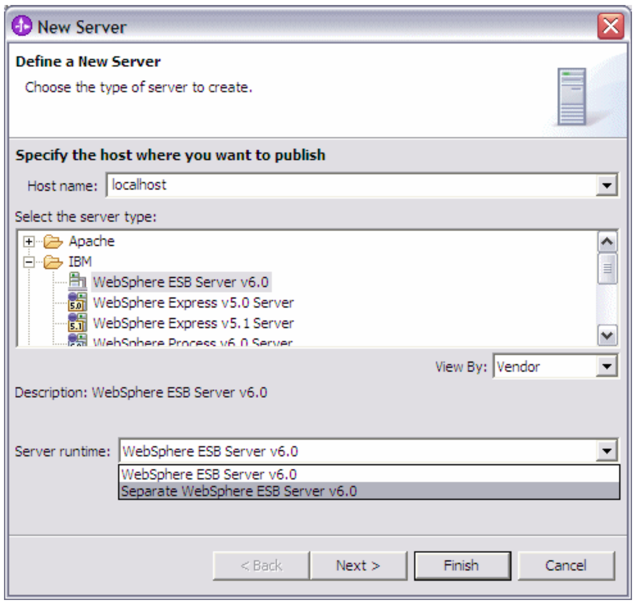
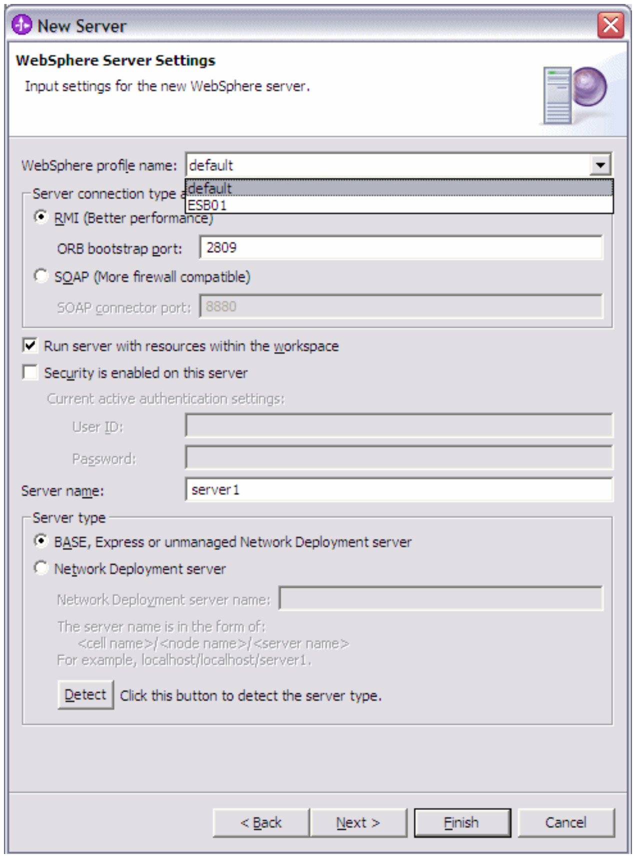

New Server
Define a New Server
Choose the type of server to create.
Specify the host where you want to publish
Host name: localhost
Select the server type:
Apache
IBM
WebSphere ESB Server v6.0
WebSphere Express v5.0 Server
WebSphere Express v5.1 Server
WebSphere Process v6.0 Server
View By: Vendor
Description: WebSphere ESB Server v6.0
Server runtime: WebSphere ESB Server v6.0
WebSphere ESB Server v6.0
Separate WebSphere ESB Server v6.0
< Back
Next >
Finish
Cancel

Figure 7-79 Server runtime choice

6. Select the separate WebSphere Enterprise Service Bus runtime and click **Next**. You will now have a choice of profiles from the profiles created for this separate runtime (Figure 7-80).



New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: default

default

ESB01

Server connection type and port
- (●) RMI (Better performance)
  - ORB bootstrap port: 2809
- ( ) SOAP (More firewall compatible)
  - SOAP connector port: 8880

[✓] Run server with resources within the workspace

[ ] Security is enabled on this server

Current active authentication settings:
- User ID:
- Password:

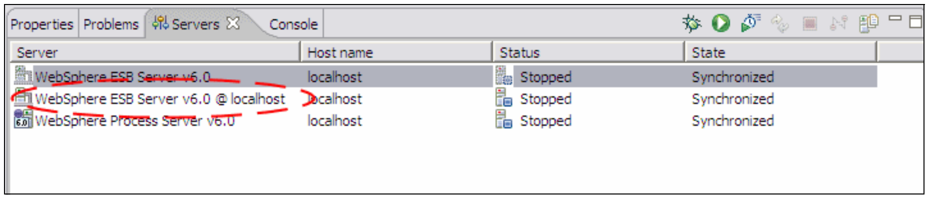
Server name: server1

Server type
- (●) BASE, Express or unmanaged Network Deployment server
- ( ) Network Deployment server
  - Network Deployment server name:
  - The server name is in the form of: <cell name>/<node name>/<server name> For example, localhost/localhost/server1.

Detect Click this button to detect the server type.

Figure 7-80 Separate runtime profile choice

7. Choose a profile and click **Next**. At the next screen optionally add workspace projects to the new server configuration and click **Finish**. The new server configuration will be listed on the Servers view (Figure 7-81).



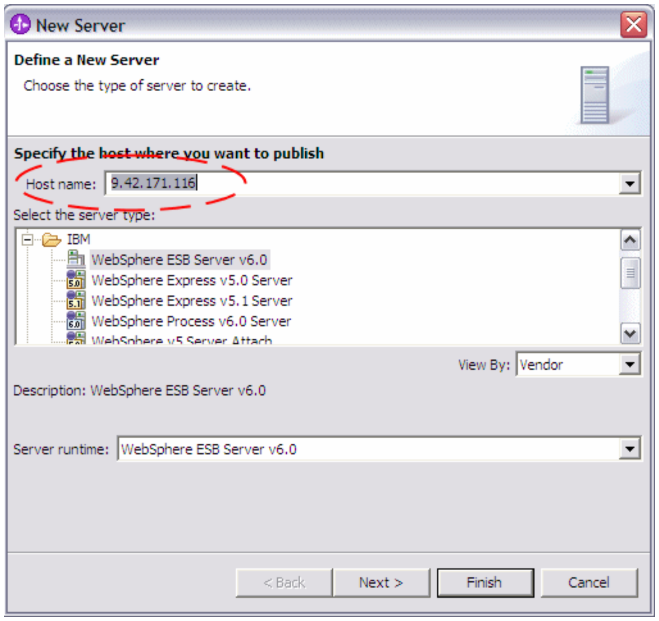
Properties | Problems | Servers | Console

Server	Host name	Status	State
~~WebSphere ESB Server v6.0~~	localhost	Stopped	Synchronized
WebSphere ESB Server v6.0 @ localhost	localhost	Stopped	Synchronized
~~WebSphere Process Server v6.0~~	localhost	Stopped	Synchronized

Figure 7-81 New server configuration

Remote test environment

1. The steps for adding a remote test environment are the same as for adding a locally installed runtime with the exception that you need to specify the host name of the remote machine (Figure 7-82).



Specify the host where you want to publish

Host name: 9.42.171.116

Select the server type:

IBM
- WebSphere ESB Server v6.0
- WebSphere Express v5.0 Server
- WebSphere Express v5.1 Server
- WebSphere Process v6.0 Server
- WebSphere v5 Server Attach

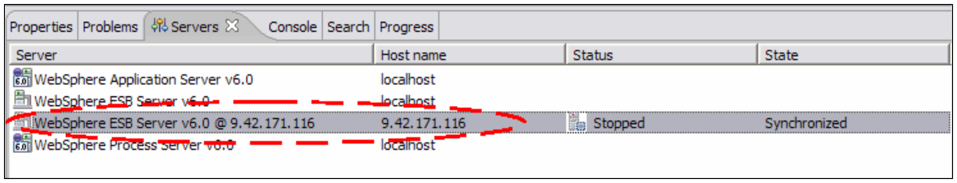
View By: Vendor

Description: WebSphere ESB Server v6.0

Server runtime: WebSphere ESB Server v6.0

Figure 7-82 Adding a remote server configuration

2. Once the server is created it will be listed on the Servers view. Note that the hostname/IP address is part of the server configuration name (Figure 7-83).



Server	Host name	Status	State
WebSphere Application Server v6.0	localhost		
WebSphere ESB Server v6.0	localhost		
WebSphere ESB Server v6.0 @ 9.42.171.116	9.42.171.116	Stopped	Synchronized
WebSphere Process Server v6.0	localhost		

Figure 7-83 Remote server configuration

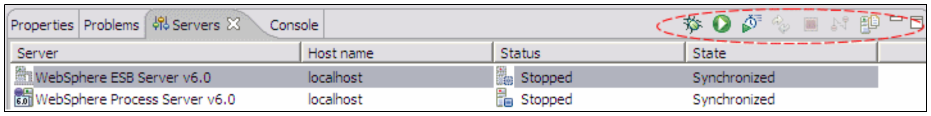
Note: After adding a separate local or remote server it is important to open the server configuration editor and review the server connection type and admin port settings. For remote servers we recommend you use the more reliable SOAP connector.

Commands to manage test servers

Once the server is configured, there are a few key commands to be aware of used to manage the test servers.

- Debug: only available for local test servers.
- Start: only available for local test servers.
- Restart: available on all active servers.
  - Can restart in different modes (normal, debug, and profile).
- Stop

Tip: Most commonly used server commands can be accessed from the Servers view toolbar (Figure 7-84)



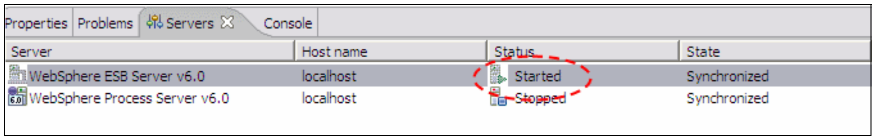
Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Stopped	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

Figure 7-84 Servers view toolbar

Starting and stopping the server

Starting the server

1. In the Servers view, right-click the server you want to start and select **Start**.
2. The console view comes to the foreground and displays logging information.
3. Wait until the `Server server1 open for e-business` message appears in the console and the server status in the Servers view is Started (Figure 7-85).



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Started	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

Figure 7-85 Server status

Note: You might see a system error about the system not being able to find the file cell-wbi.xml. This error can be safely ignored.

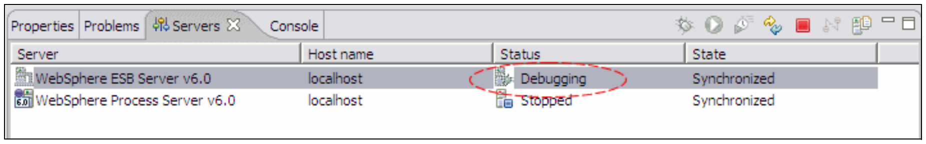
Stopping the server

1. In the Servers view, right click on the server you want to stop and select **Stop**
2. Wait until the server status is Stopped in the Servers view.

Starting the server in debug mode

If you want to debug code deployed to your test server you need to start the server in *debug mode*.

1. Click the **Debug** tool on the Servers view toolbar or right-click the server and select **Debug**.
2. Once the operation completes the server status in the Servers view should be Debugging (Figure 7-86).



Server	Host name	Status	State
WebSphere ESB Server v6.0	localhost	Debugging	Synchronized
WebSphere Process Server v6.0	localhost	Stopped	Synchronized

Figure 7-86 Server started in debug mode

Tip: If the server is already running, a quick way to switch to debug mode is to right-click the server and select **Restart** → **Debug**.

For more information about debugging, refer to 8.2, “Debugging tools” on page 233.

Running the administrative console

You can run the administrative console for a running server from within WebSphere Integration Developer.

1. Right-click the server and select **Run administrative console** from the context menu.
2. Click the **Log In** button to enter the console. Security is not enabled in the test environment.
3. Scroll to the bottom of the welcome page and use the Task filtering selector to apply available filters (Figure 7-87). For example, the *Application Integration* filter does not include tasks to manage servers. If you need to modify port numbers you need the server tasks available to you:
   a. Click **Server and Bus**.
   b. Click **Apply**.

Tip: You can change the filter at any time by going back to the welcome page. Click the **Welcome** link at the top of the administrative console menu.

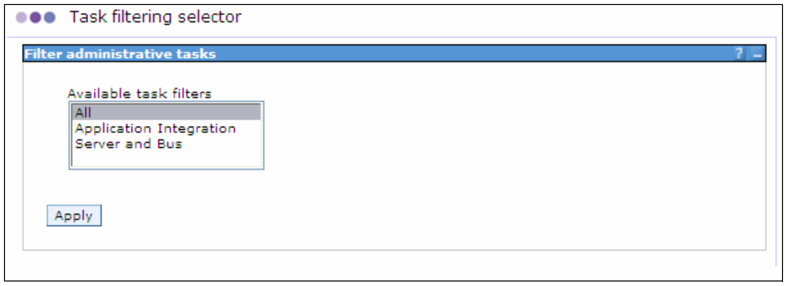


Figure 7-87 Filter administrative tasks

For more information about administering WebSphere Enterprise Service Bus resources see Chapter 9, “Administering WebSphere Enterprise Service Bus” on page 251.

7.5.3 Deploying mediation modules

In order to test your mediation modules you must run them on the test server. This section describes how to add and remove projects to the test server.

Adding projects to the test server

1. Right-click the server and select **Add and remove projects**
2. Add the required projects from the Available projects pane to the Configured projects pane (Figure 7-88). Add projects individually by selecting them and clicking the **Add >** button. You can add all workspace projects by using the **Add All >>** button.
3. Once the list is complete click **Finish**.

Note: Calling up the Add and Remove Projects dialog is a convenient way to check which modules are published to the test server, as they show up in the Configured projects pane.

However, you cannot use this method to see projects that were published to the test server from a different workspace. If in doubt, run the administrative console and verify which applications are running on the test server.

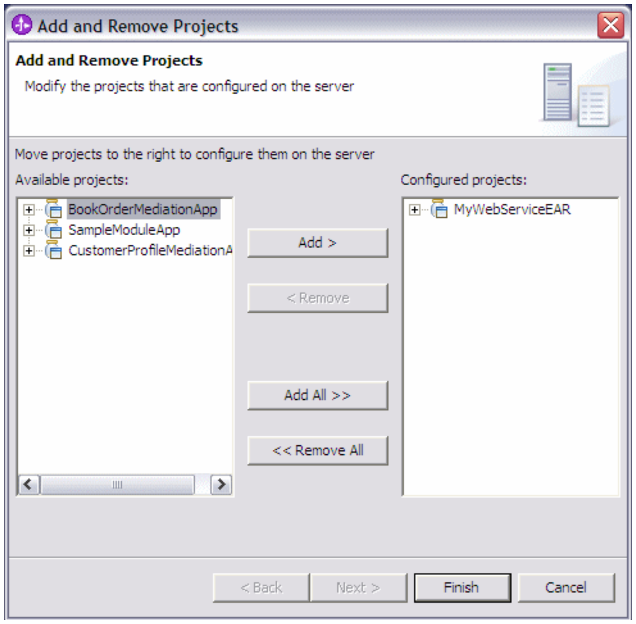


Figure 7-88 Add and remove projects

Removing projects from the test server

To remove projects from the test server follow the same steps as for adding projects but use the **< Remove** or **<< Remove All** buttons.

Important: It is a good practice to always remove all projects and stop the server before switching workspaces or exiting WebSphere Integration Developer. Projects that you add from a given workspace are not visible from another one but those modules are still installed and will be started when the the server starts.

7.5.4 Testing mediation modules

Once the module is deployed or published to the test server you can test it with the Integration Test Client.

Typically you will perform module tests and component tests.

Module test

In the Business Integration view, right-click the module and select **Test** → **Test Module**. This will launch the Integration Test Client with all emulation disabled.

Component test

In the module assembly diagram, right-click the mediation flow component and select **Test Component**. This will launch the Integration Test Client with emulators configured to emulate any component references so you can test the component in isolation.

For more infomation about testing, refer to 8.1, “Testing tools” on page 220.

7.6 Exporting resources

This section describes the basics of exporting workspace resources (Figure 7-89).

We discuss exporting resources to Project Interchange files and to EAR files.

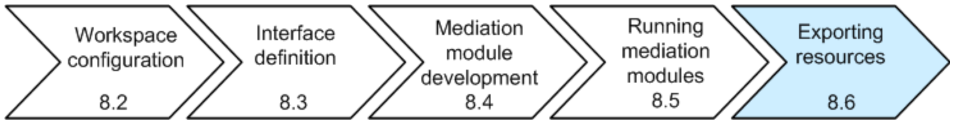


Figure 7-89 Exporting resources stage

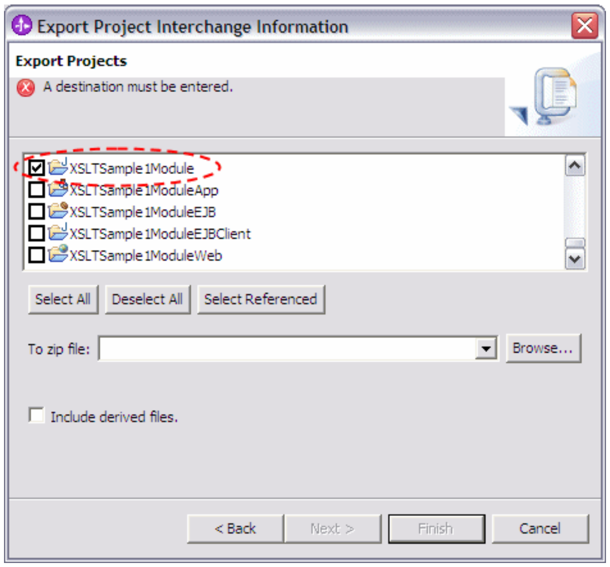
Project Interchange files are typically used by developers to share modules and libraries between workspaces.

Enterprise Archive (EAR) files are deployable units to be installed and run on target runtimes.

7.6.1 Exporting to Project Interchange

1. From the menu bar select **File** → **Export**
2. On the Export Project Interchange Information dialog, select the module you wish to export (Figure 7-90).

Note: There are four staging projects associated with each Business Integration module: an Enterprise Application, EJB, EJBClient and Web project. These are generated by WebSphere Integration Developer and need not be exported.



A destination must be entered.

- [✓] XSLTSample1Module
- [ ] XSLTSample1ModuleApp
- [ ] XSLTSample1ModuleEJB
- [ ] XSLTSample1ModuleEJBClient
- [ ] XSLTSample1ModuleWeb

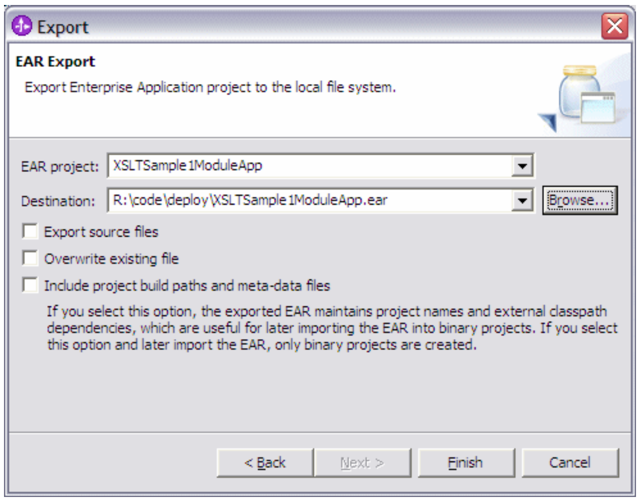
Figure 7-90 Export module to Project Interchange file

3. Click the **Browse** button and navigate to the target location in the file system.
4. Choose a filename and click **Save**.
5. Click **Finish**.

7.6.2 Exporting enterprise applications

To deploy a mediation module anywhere other than your integrated test environment, you need to export the module as an EAR file.

1. To do this, right-click the module, select **Export**, choose **EAR file** from the selection of export destinations and click **Next**.
2. On the EAR Export dialog, select the EAR project to export and a destination for the EAR file. You can optionally export source code and workspace metadata with the EAR file (Figure 7-91). Click **Finish**.
3. This creates an EAR file which will include the selected module and its required libraries.


Export

EAR Export

Export Enterprise Application project to the local file system.

EAR project: XSLTSample1ModuleApp

Destination: R:\code\deploy\XSLTSample1ModuleApp.ear

☐ Export source files

☐ Overwrite existing file

☐ Include project build paths and meta-data files

If you select this option, the exported EAR maintains project names and external classpath dependencies, which are useful for later importing the EAR into binary projects. If you select this option and later import the EAR, only binary projects are created.

Figure 7-91 EAR Export

Part 3

Administration and testing

8

Testing, debugging and problem determination

This chapter discusses the techniques and tools available to test the artifacts you develop for WebSphere Enterprise Service Bus and to perform troubleshooting in the runtime environment. We will look at the test tools in WebSphere Integration Developer and techniques for isolating code problems in development. We will also discuss how you can perform problem determination in the more advanced test stages and in the production environment, using the capabilities of the WebSphere Enterprise Service Bus runtime. We will cover the following areas:

- Integration Test Client
- Web Services Explorer
- TCP/IP Monitor
- Integration debugger
- Problem determination tools and techniques

8.1 Testing tools

The role of the integration developer naturally includes responsibilities for unit testing the components being developed in WebSphere Integration Developer. We review three of the key tools available to support this activity in the sections that follow:

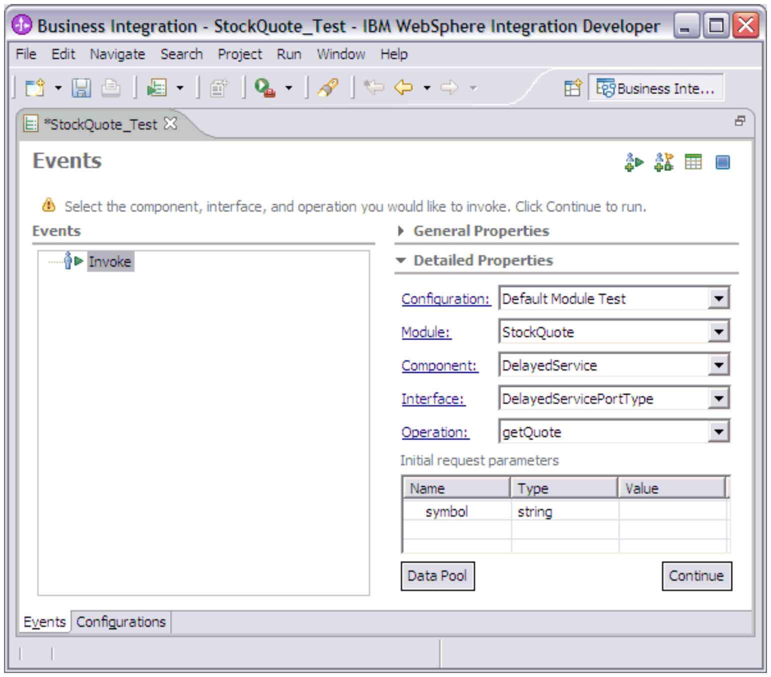
- Integration Test Client
- Web Services Explorer
- TCP/IP Monitor

8.1.1 Integration Test Client

This section discusses the Integration Test Client available in WebSphere Integration Developer, which is the recommended tool to test your mediation components.

Testing modules

To test an entire mediation module, right click on a mediation module and select **Test -> Test Module**. This will launch the Integration Test Client (Figure 8-1 on page 221).


Business Integration - StockQuote_Test - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Business Inte...
*StockQuote_Test
Events

Select the component, interface, and operation you would like to invoke. Click Continue to run.

Events
Invoke
General Properties
Detailed Properties

Configuration: Default Module Test
Module: StockQuote
Component: DelayedService
Interface: DelayedServicePortType
Operation: getQuote

Initial request parameters

Name	Type	Value
symbol	string	

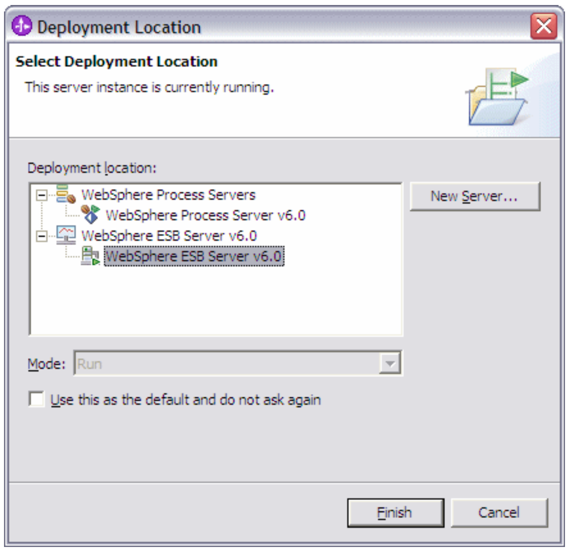
Data Pool
Continue
Events Configurations

Figure 8-1 Integration Test Client

When the test client starts, pay particular attention to the selected Component under Detailed Properties. You can also specify which interface and operation to invoke from the component.

You enter the data to send in a request message in the Initial request parameter section. When you have entered the data, click **Continue**.

At this point you will be asked to specify a deployment location (Figure 8-2 on page 222). This determines which server will be used to run the test. In our case we selected the WebSphere Enterprise Service Bus server and clicked **Finish**.


Deployment Location

Select Deployment Location

This server instance is currently running.

Deployment location:

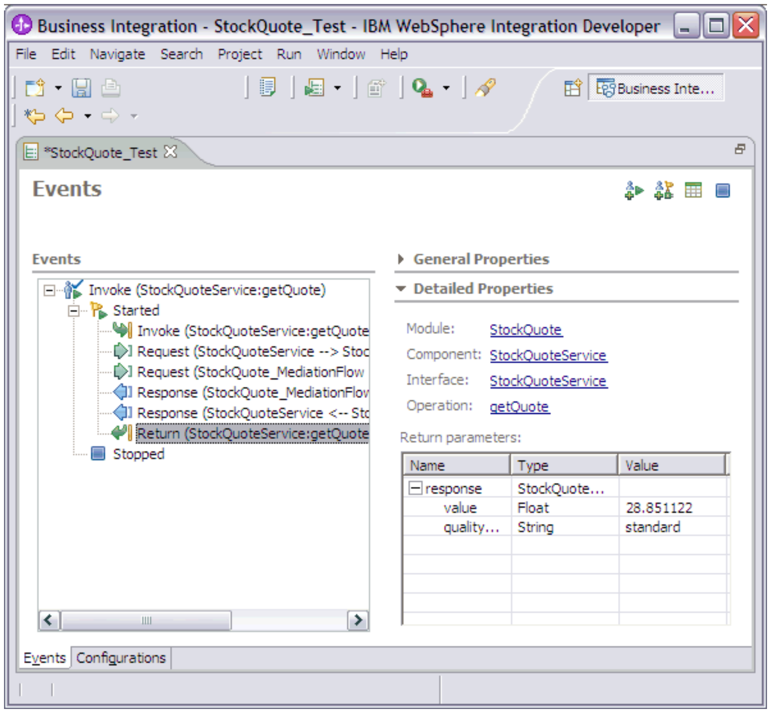
- WebSphere Process Servers
  - WebSphere Process Server v6.0
- WebSphere ESB Server v6.0
  - WebSphere ESB Server v6.0

Mode: Run

Use this as the default and do not ask again

Figure 8-2 Deployment location

The Integration Test Client will invoke the operation you specified. You can follow the request and response messages generated during the test in the Events section (Figure 8-3 on page 223). Highlight any event to see the message data used.


Business Integration - StockQuote_Test - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Business Inte...
*StockQuote_Test
Events
Events | Configurations

Events

```
⊟ Invoke (StockQuoteService:getQuote)
   ⊟ Started
        Invoke (StockQuoteService:getQuote
        Request (StockQuoteService --> Stoc
        Request (StockQuote_MediationFlow
        Response (StockQuote_MediationFlov
        Response (StockQuoteService <-- Stc
        Return (StockQuoteService:getQuote
     Stopped
```

General Properties

Detailed Properties

Module:	StockQuote
Component:	StockQuoteService
Interface:	StockQuoteService
Operation:	getQuote

Return parameters:

Name	Type	Value
⊟ response	StockQuote...	
value	Float	28.851122
quality...	String	standard

Figure 8-3 Integration Test Client

If we want to run the test a second time, simply click the **Invoke** button in the upper right portion of the pane.

Other testing functions

The Integrated Test Client provides four buttons for testing, as shown in Table 8-1 on page 223.

Table 8-1 Test Client Buttons

Button Name / Function	Icon
Invoke Generates an Invoke event in the Events area	▶
Attach Attaches the Integration Test Client directly to a test configuration module	⚑

Button Name / Function	Icon
Data Pool Opens the data pool editor, which enables you to view, edit, select, and use the saved data pool values	▦
Stop Generates a Stopped event in the Events area and detaches the integration test client from the server	■

The Invoke button sends an event to the selected component in order to initiate the test.

If you do not need an event to start your test, but rather will drive it from an external source, for example put a message on a JMS queue, or make a Web service request, then the Attach facility is useful. When you attach the test client to your configuration module, the client will show all the events processed after the external invocation.

If the operation you are testing has a fairly large number of attributes on the request message, you may want to use the Data Pool to save them after you have entered them once. Also, if you switch to the Configurations tab, shown in Figure 8-4 on page 225, at the bottom of the Test Client, you can save the test configuration, and load it later to speed your testing.

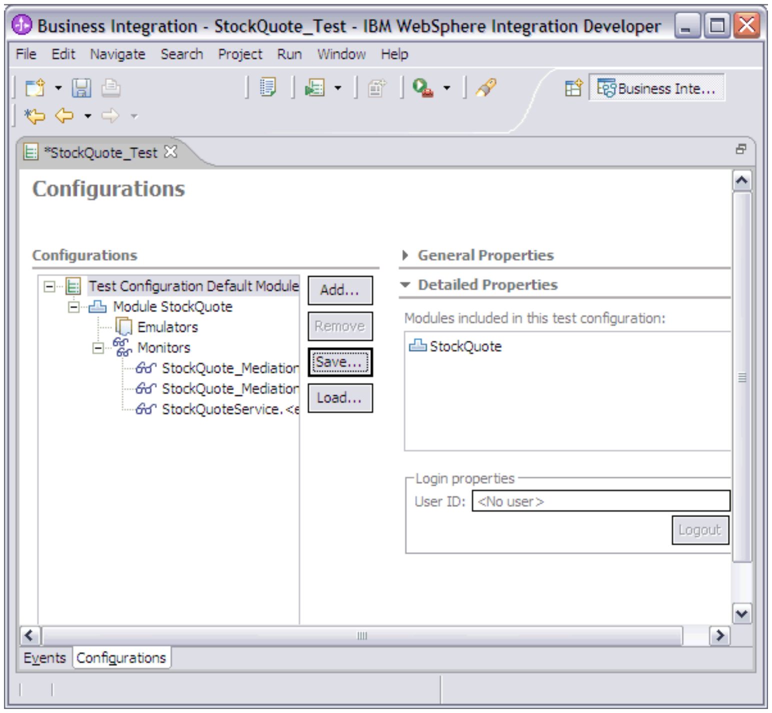


Figure 8-4 Test Client configuration tab

Emulation

Another useful function of the Test Client is the capability to *emulate* components in your module. When a component is emulated, the Test Client will intercept the message as it flows to the component, display the input parameters, and provide a form for entering the output parameters, allowing you to continue your test. This is valuable when other components are not yet developed, or when you want to focus your testing on one component specifically, possibly driving various code paths by varying the output that you enter.

There is a way to launch the Test Client that will add emulators to the test configuration automatically. While in the assembly diagram editor, if you select a component, right-click and select **Test Component**, emulators are added for any references in the component under test (Figure 8-5 on page 226).

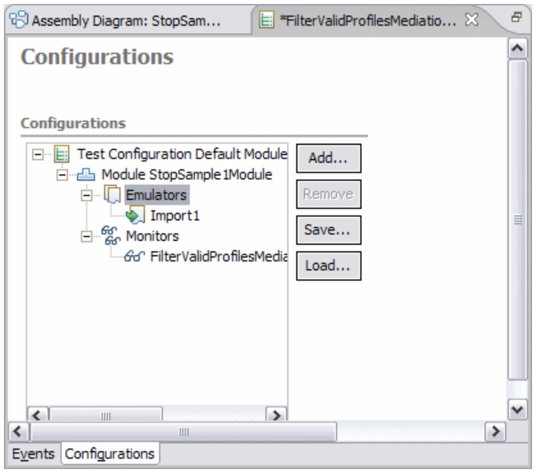


Figure 8-5 Emulating components

The configuration editor also allows you to add and remove emulators, and to configure programmatic emulators as well.

8.1.2 Web Services Explorer

The Web Services Explorer in WebSphere Integration Developer allows you to invoke Web services using SOAP over HTTP, and view the SOAP request and response messages used in this Web services interaction.

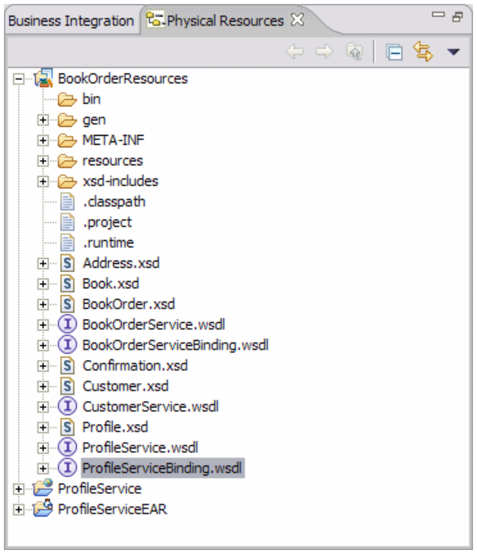
We examine these capabilities in this section using the ProfileService Web service example.

Note: In order to follow along with the step-by-step instructions in this section you will need to have prepared a WebSphere Integration Developer workspace with the necessary resources as described in Chapter 10, “Preparing for the development examples” on page 271.

To test the Web Services Explorer, perform the following:

1. Deploy the ProfileService Web service to the server.
   a. Switch to the Servers view.
   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
   c. Add ProfileServiceEAR.
   d. Click **Finish**.

2. In the Business Integration view, right click on the **BookOrderResources** project and select **Show Files**. This opens the Physical Resources view.
3. Select **ProfileServiceBinding.wsdl** as shown in Figure 8-6 on page 227.



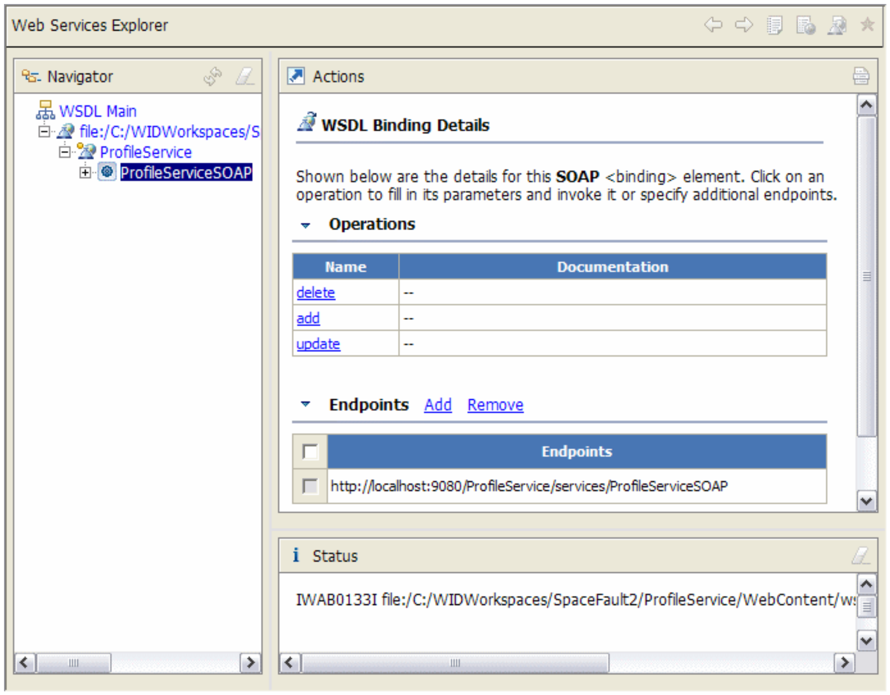
```
BookOrderResources
    bin
    gen
    META-INF
    resources
    xsd-includes
    .classpath
    .project
    .runtime
    Address.xsd
    Book.xsd
    BookOrder.xsd
    BookOrderService.wsdl
    BookOrderServiceBinding.wsdl
    Confirmation.xsd
    Customer.xsd
    CustomerService.wsdl
    Profile.xsd
    ProfileService.wsdl
    ProfileServiceBinding.wsdl
ProfileService
ProfileServiceEAR
```

Figure 8-6 BookOrderResources

4. Right-click and select **Web Services** -> **Test with Web Services Explorer**.

> **Note:** You must enable Web Service Development in your workspace capabilities in order for the Web Services menu item to appear.

5. The Web Services Explorer is launched, and the WSDL file is parsed. You can see from Figure 8-7 that the explorer has listed the Web service endpoint as well as the operations defined in the WSDL.



WSDL Binding Details

Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
delete	--
add	--
update	--

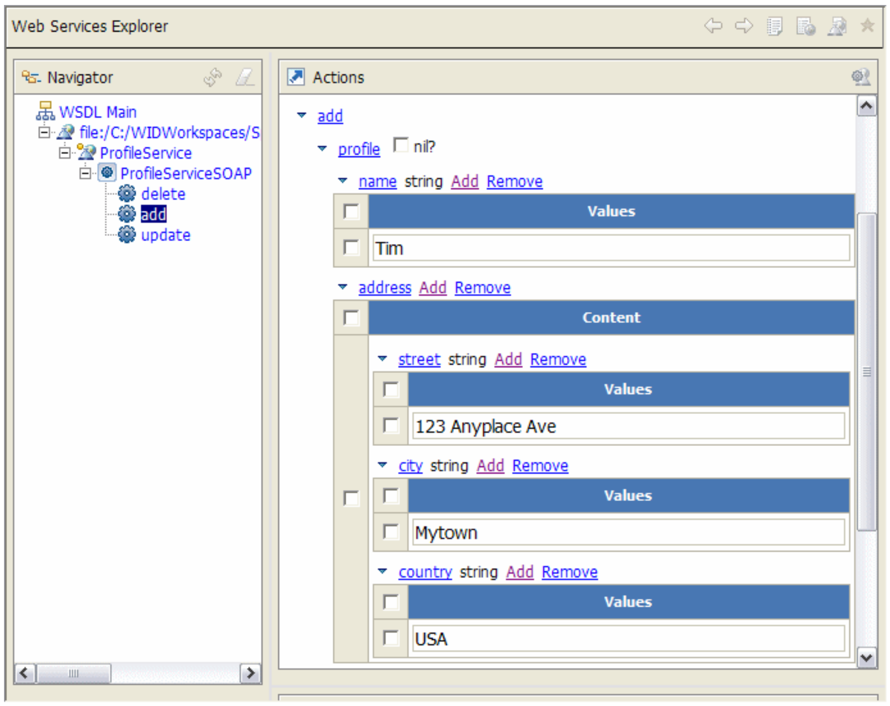
Endpoints Add Remove

Endpoints
http://localhost:9080/ProfileService/services/ProfileServiceSOAP

IWAB0133I file:/C:/WIDWorkspaces/SpaceFault2/ProfileService/WebContent/w

Figure 8-7 Web Services Explorer opened on ProfileService

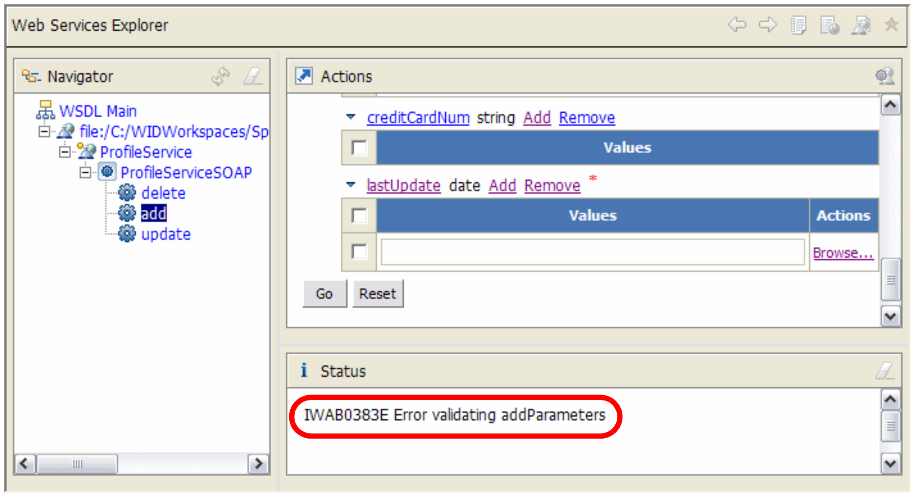
6. Click on the **Add** link to initialize an add operation to the Web service.
7. For each attribute that you want to set in the Web service request, click on the **Add** link next to the attribute and enter an appropriate value, as shown in Figure 8-8 on page 229.



add
profile nil?
name string Add Remove
Values
Tim
address Add Remove
Content
street string Add Remove
Values
123 Anyplace Ave
city string Add Remove
Values
Mytown
country string Add Remove
Values
USA

Figure 8-8 Setting attributes on the request

8. Click on the **Add** link next to the lastUpdate attribute, but leave the field blank.
9. Click **Go** and notice the error message that displays in the Status portion of the page (Figure 8-8 on page 229).

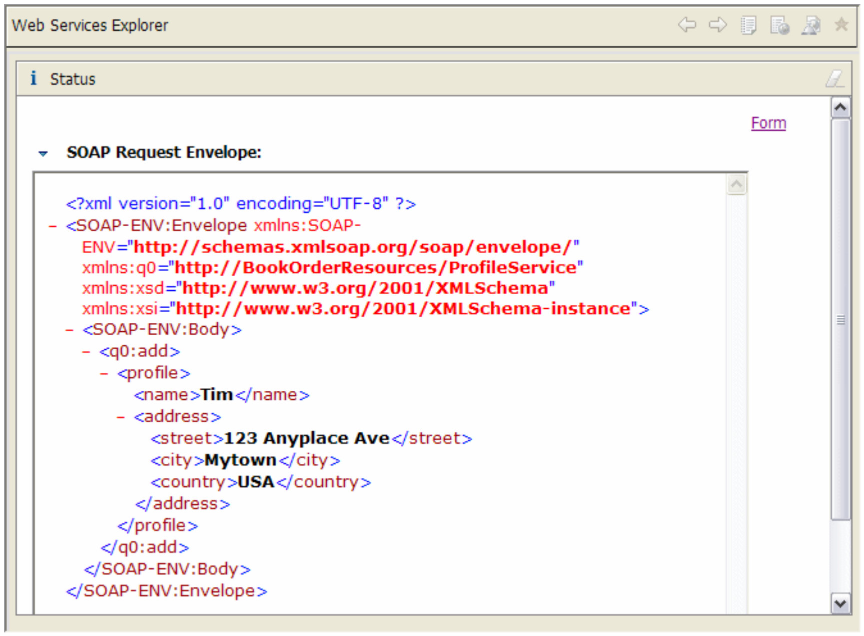


```
IWAB0383E Error validating addParameters
```

Figure 8-9 Understanding validation errors

10. The explorer has used the XML schema definition of the request message to validate the inputs, and has displayed a red asterisk next to the field that was in error. In this case, the attribute is defined as a Date field and so cannot be blank.
11. Check the empty value for lastUpdate, then click on the **Remove** link next to lastUpdate to remove the attribute from the request message. The schema allows for this attribute to be absent from the request.
12. Click **Go** to execute the Web service.
13. You can view the content of the request SOAP message sent to the Web service and the response SOAP messages returned from the Web service by clicking **Source** in the Status view (Figure 8-10 on page 231).

> **Note:** You can double click the task bar of the Actions pane or the Status pane to maximize them.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:q0="http://BookOrderResources/ProfileService"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <SOAP-ENV:Body>
    - <q0:add>
      - <profile>
          <name>Tim</name>
        - <address>
            <street>123 Anyplace Ave</street>
            <city>Mytown</city>
            <country>USA</country>
          </address>
        </profile>
      </q0:add>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Figure 8-10 SOAP request

14. Remove ProfileServiceEAR from the server.

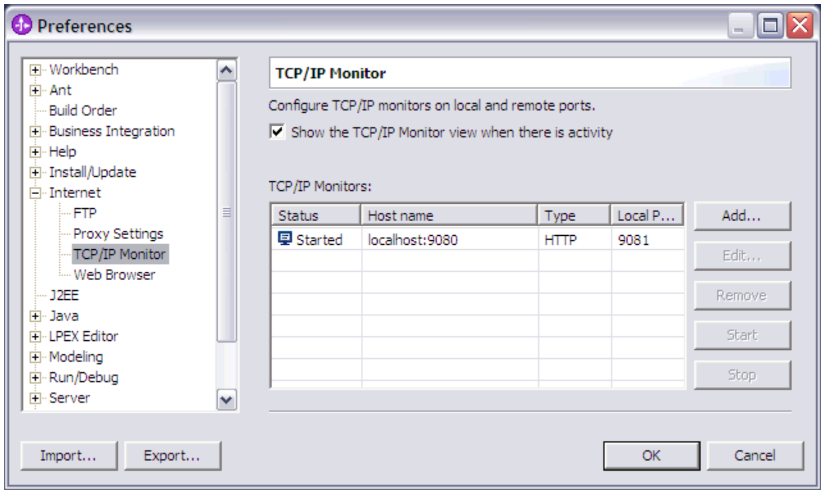
8.1.3 TCP/IP Monitor

The TCP/IP Monitor allows you to view the content of TCP/IP messages as they flow across a network. This is often particularly useful for monitoring HTTP messages, including SOAP/HTTP Web service message.

Additionally, the TCP/IP Monitor is used to redirect requests to an alternative port. For example, we built a mediation module which used an Import component to access a Web service. Although the Web service was deployed to the test server which was listening for HTTP requests on port 9080, the Import component had been built to request the Web service at port 9081, causing our test to fail. We used the TCP/IP Monitor to listen for requests on port 9081, and forward them to port 9080.

To perform this, we completed the following:

1. Select **Window** -> **Preferences**, expand **Internet** and select **TCP/IP Monitor** (Figure 8-11 on page 232)



TCP/IP Monitor

Configure TCP/IP monitors on local and remote ports.

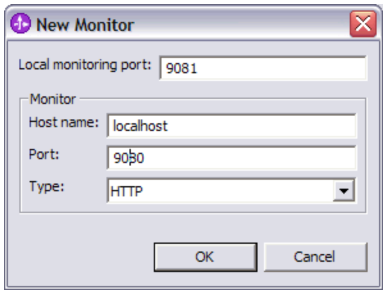
✓ Show the TCP/IP Monitor view when there is activity

TCP/IP Monitors:

Status	Host name	Type	Local P...
Started	localhost:9080	HTTP	9081

Figure 8-11 Configuring the TCP/IP monitor

2. To add a port to listen on, click **Add**.
3. The New Monitor dialog shown in Figure 8-12 on page 232 is displayed. To configure the monitor to listen on port 9081 and forward to port 9080, perform the following:
   a. Set Local monitoring port to `9081`.
   b. Set Host name to `localhost`.
   c. Set Port to `9080`.
   d. Set Type to `HTTP`.
   e. Click **OK**.



Local monitoring port: 9081

Monitor

Host name: localhost

Port: 9080

Type: HTTP

Figure 8-12 Creating a new TCP/IP monitor

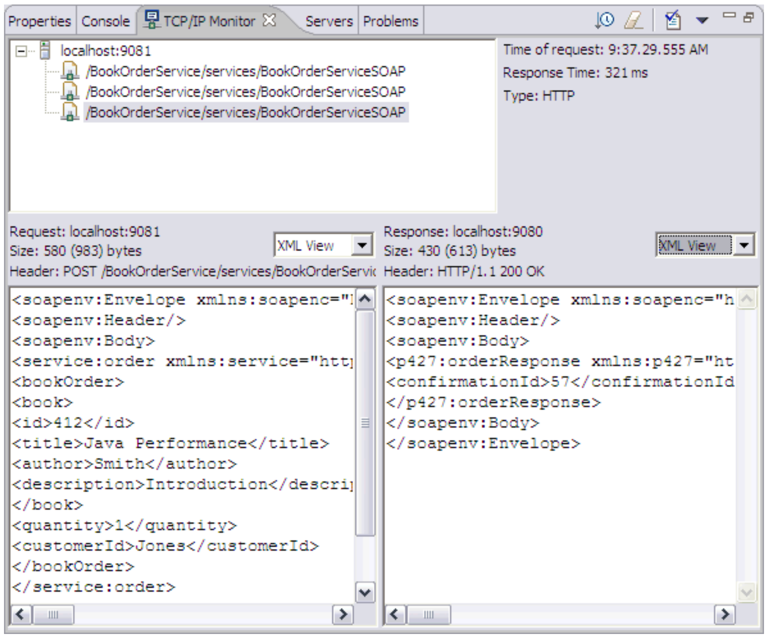
4. Once you have configured the monitor, select it and click the **Start** button.

In addition to forwarding TCP/IP messages, the TCP/IP Monitor shows the content of the messages as they pass through the TCP/IP Monitor, and the responses (if any) to these messages.

When a message passes through the TCP/IP Monitor, the TCP/IP Monitor view will become visible. To manually view it, select **Window** -> **Show View** -> **Other** -> **Debug**, select **TCP/IP Monitor** and click **OK**.

To view the content of a TCP/IP interaction, highlight it in the TCP/IP Monitor view. The request and response messages will be visible (Figure 8-13).

.



```
localhost:9081
    /BookOrderService/services/BookOrderServiceSOAP
    /BookOrderService/services/BookOrderServiceSOAP
    /BookOrderService/services/BookOrderServiceSOAP
```

Time of request: 9:37.29.555 AM
Response Time: 321 ms
Type: HTTP

Request: localhost:9081
Size: 580 (983) bytes
Header: POST /BookOrderService/services/BookOrderServic

```
<soapenv:Envelope xmlns:soapenc=":
<soapenv:Header/>
<soapenv:Body>
<service:order xmlns:service="htt]
<bookOrder>
<book>
<id>412</id>
<title>Java Performance</title>
<author>Smith</author>
<description>Introduction</descri]
</book>
<quantity>1</quantity>
<customerId>Jones</customerId>
</bookOrder>
</service:order>
```

Response: localhost:9080
Size: 430 (613) bytes
Header: HTTP/1.1 200 OK

```
<soapenv:Envelope xmlns:soapenc="h
<soapenv:Header/>
<soapenv:Body>
<p427:orderResponse xmlns:p427="ht
<confirmationId>57</confirmationId
</p427:orderResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 8-13 TCP/IP Monitor details

8.2 Debugging tools

In this section we describe the Integration debugger that is part of WebSphere Integration Developer and how it can be used to debug your mediation modules.

8.2.1 Integration debugger

The Integration debugger can be used with the test server in your WebSphere Integration Developer workspace, or can be used to debug a mediation module on a remote server. We focus on the first usage, as it should be the most prevalent. The InfoCenter has details on setting up to debug a remote server. See:

```
http://publib.boulder.ibm.com/infocenter/rtnl0600/index.jsp?topic=/org.ecli
pse.jdt.doc.user/concepts/cremdbug.html
```

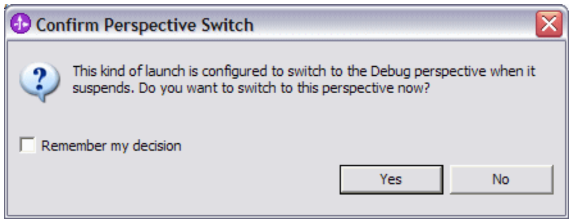
If you have any experience using the debugger that ships with Rational Application Developer, you will find the Integration debugger to be very intuitive, but you will notice some very important enhancements that make debugging mediation flows very straightforward.

8.2.2 Setting up to use the debugger

There are two key tasks required to set up for a debugger session:

1. You must start your test server in Debug mode, which can be done from the pop-up menu on the server or using the debug icon in the server pane of the Business Integration perspective.
2. You should set breakpoints on the components you are debugging. This can be done from the mediation flow editor by right clicking the component where you would like to add the breakpoint and selecting **Add Breakpoint** from the context menu. To remove a breakpoint, use the same context menu and select **Remove Breakpoint**.

Once the server is started in debug mode, and your breakpoints are set, you can drive your test case. The Integration Test Client is a good way to do this. When the debugger process gains control, a pop-up is displayed asking if you want to switch to the debug perspective, as shown in Figure 8-14 on page 234.



This kind of launch is configured to switch to the Debug perspective when it suspends. Do you want to switch to this perspective now?

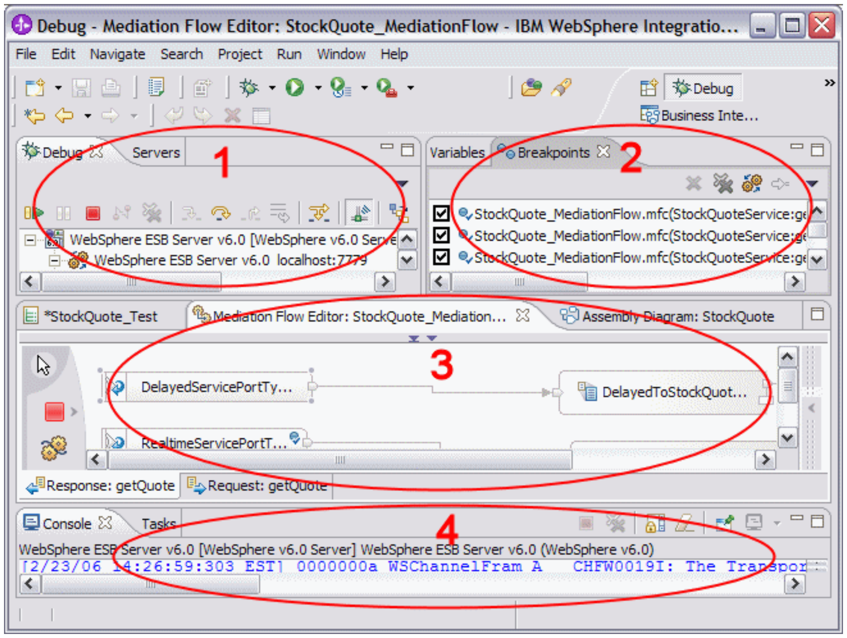
Remember my decision

Figure 8-14 Switching to the Debug perspective

8.2.3 Overview of the Debug perspective

Looking first at the Debug perspective as a whole, you can see four main sections of the window, which are numbered in Figure 8-15 on page 235 to match the following:

1. Debug / Servers view - this pane has tabs for the Debug view and the Servers view. The latter, you are already familiar with, as it is the same view that exists in the Business Integration perspective. The Debug view is used to control the execution of component instances and alter their state at runtime.
2. Breakpoint / Variable view - in this view, the Breakpoint tab lists all the breakpoints that have been set. You can disable and enable the breakpoints here, or remove them all together. The Variable view displays all the variables, messages and associated values for a component.
3. Component view - this view is the same view as the upper right pane in the Business Integration perspective. In the Debug perspective, it can be used to trace execution through a mediation flow, add and remove breakpoints from the components in the mediation view, and review the execution status by component.
4. Console view - this is the same view as is available in the Business Integration perspective, and is convenient to have in the Debug perspective as you can view messages displayed by the server and the application during the test.


Debug - Mediation Flow Editor: StockQuote_MediationFlow - IBM WebSphere Integratio...
File Edit Navigate Search Project Run Window Help
Debug
Business Inte...
Debug
Servers
1
Variables
Breakpoints
2
*StockQuote_Test
Mediation Flow Editor: StockQuote_Mediation...
Assembly Diagram: StockQuote
3
DelayedServicePortTy...
DelayedToStockQuot...
RealtimeServicePortT...
Response: getQuote
Request: getQuote
Console
Tasks
4

WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
WebSphere ESB Server v6.0 localhost:7779

StockQuote_MediationFlow.mfc(StockQuoteService:ge
StockQuote_MediationFlow.mfc(StockQuoteService:ge
StockQuote_MediationFlow.mfc(StockQuoteService:ge

```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server] WebSphere ESB Server v6.0 (WebSphere v6.0)
[2/23/06 14:26:59:303 EST] 0000000a WSChannelFram A   CHFW0019I: The Transpor
```

Figure 8-15 Debug perspective

At the top of the Debug view, there is a button bar that you use to direct the execution of the component instance. The buttons that you are most likely to use frequently are shown in Table 8-2.

Table 8-2 Most useful debug view buttons

Button Name / Function	Icon
Resume Continues component instance execution until the next breakpoint or until exit	▶
Stop Terminates the component instance execution	■
Step Over Continues component instance execution until next component is entered, at which time execution is suspended	↷

The Resume button will be enabled whenever the execution of the module has been suspended. Clicking this button will cause execution to continue until the next breakpoint is reached, or until the thread terminates.

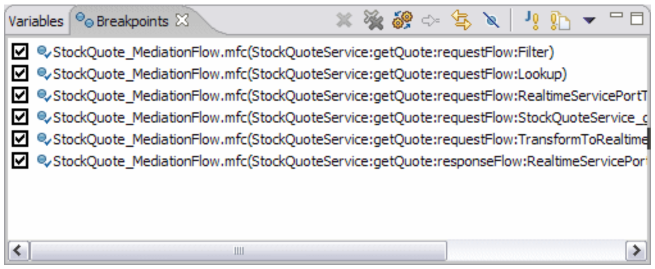
The Step Over button is also frequently used This will cause execution to continue to the next component in the mediation flow, where it will again be suspended. If you want to trace through every component in a mediation flow, this button is useful, yet if you are trying to get to a specific component quickly, setting an explicit breakpoint and using Resume can be a much quicker approach.

8.2.4 Using the Integrated Debugger

We examine the details of the various views in the Integrated debugger in the following sections We will use it to debug a successful invocation of the StockQuoteService enterprise application, which is a sample application shipped with WebSphere Enterprise Service Bus.

To follow these steps you will need to install and configure the StockQuoteService enterprise application. For information on how to do this, select **Help** -> **Samples Gallery**, and in the Samples Gallery expand **Application samples** -> **Business Integration** and select **Stock quote for mediation flows**.

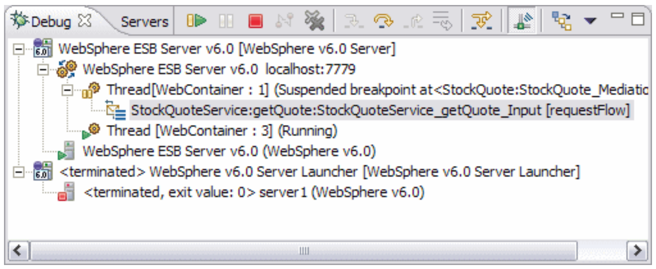
1. Open the mediation flow editor on the **StockQuote_MediationFlow**. Select the connection between the two operations, so that the request and response flows are displayed at the bottom of the pane.
2. Set breakpoints on the following components:
   a. StockQuoteService_getQuote_Input
   b. Lookup
   c. Filter
   d. TransformToRealtime
   e. RealtimeServicePortTypePartner_getQuote_Callout
   f. RealtimeServicePortTypePartner_getQuote_CalloutResponse
3. In the Test Client, select **StockQuoteService** as the component and enter the following values for the attributes of the request message:
   a. Symbol: `AAA`
   b. Customer: `CustomerB`
4. Click **Continue**, and allow the perspective to switch to the Debug perspective.
5. Look at the list of breakpoints, in the Breakpoints view. It should match those listed in Figure 8-16.



```
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:requestFlow:Filter)
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:requestFlow:Lookup)
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:requestFlow:RealtimeServicePortT
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:requestFlow:StockQuoteService_g
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:requestFlow:TransformToRealtime
☑ StockQuote_MediationFlow.mfc(StockQuoteService:getQuote:responseFlow:RealtimeServicePor
```

Figure 8-16 Breakpoint view

6. In the Debug view, notice that execution has stopped in the StockQuoteService_getQuote_Input (Figure 8-17 on page 238).



```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
  WebSphere ESB Server v6.0  localhost:7779
    Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
      StockQuoteService:getQuote:StockQuoteService_getQuote_Input [requestFlow]
    Thread [WebContainer : 3] (Running)
  WebSphere ESB Server v6.0 (WebSphere v6.0)
<terminated> WebSphere v6.0 Server Launcher [WebSphere v6.0 Server Launcher]
  <terminated, exit value: 0> server1 (WebSphere v6.0)
```

Figure 8-17 Break in StockQuoteService_getQuote_Input

7. In the Component view, notice the icon on the StockQuoteService_getQuote_Input. Use your mouse to hover over the orange debug icon. The text says that the breakpoint has popped. This is another good way to determine where the execution has been suspended (Figure 8-18).

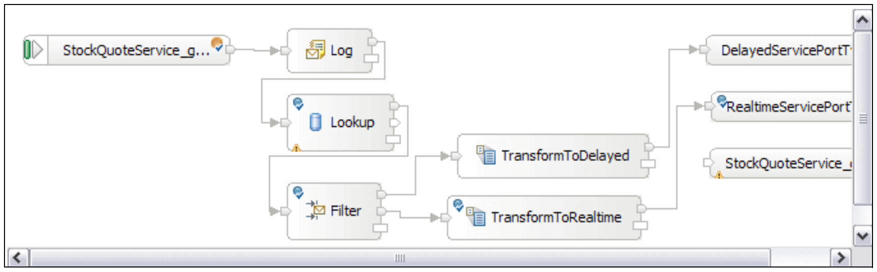
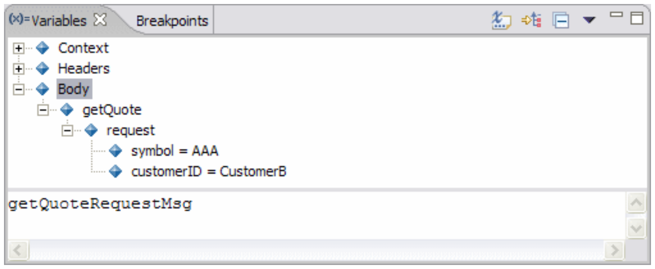


Figure 8-18 StockQuoteService_getQuote_Input icons updated

8. In the Variables view, inspect the Body of the message. You will see the contents of the request message (Figure 8-19).

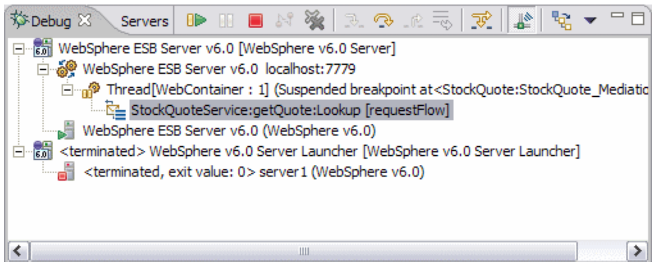


```
Context
Headers
Body
  getQuote
    request
      symbol = AAA
      customerID = CustomerB
getQuoteRequestMsg
```

Figure 8-19 Variables in StockQuoteService_getQuote_Input

9. Now click the **Resume** button in the Debug view.

10. Execution stops in the Lookup mediation primitive (Figure 8-20 on page 239).



```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
  WebSphere ESB Server v6.0  localhost:7779
    Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
      StockQuoteService:getQuote:Lookup [requestFlow]
  WebSphere ESB Server v6.0 (WebSphere v6.0)
<terminated> WebSphere v6.0 Server Launcher [WebSphere v6.0 Server Launcher]
  <terminated, exit value: 0> server1 (WebSphere v6.0)
```

Figure 8-20 Break in Lookup

11.The icons in the Component view have changed again. We see that the breakpoint in Lookup has been reached. We also see two new icons on the connections between the first three components. The purple circle with a check mark indicates the path that is being taken through the mediation flow (Figure 8-21).

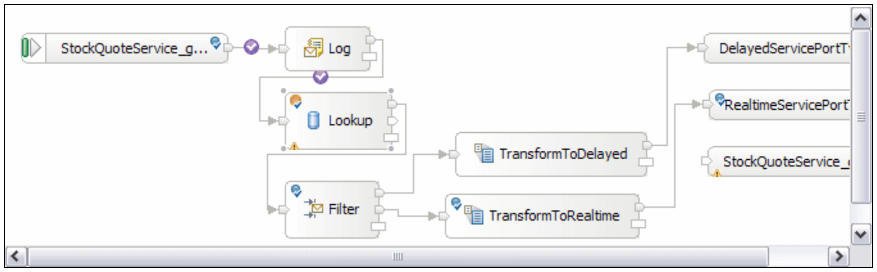
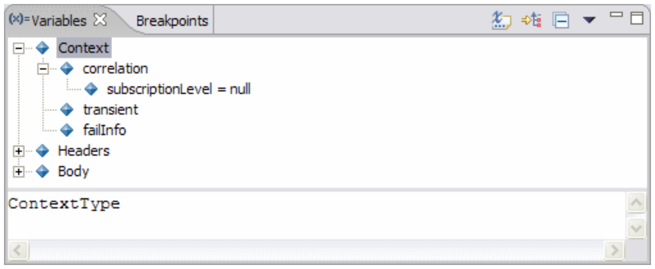


Figure 8-21 Lookup icons updated

12.In the Variables view, if we inspect the Context, we see the subscriptionLevel attribute of the Correlation is null(Figure 8-22).

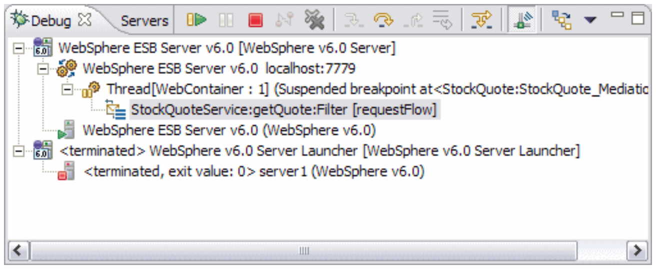


```
Context
  correlation
    subscriptionLevel = null
  transient
  failInfo
Headers
Body

ContextType
```

Figure 8-22 Variables in Lookup

13.Click **Resume** in the Debug view.

14.Execution is suspended in the Filter mediation primitive (Figure 8-23).



Debug Servers

```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
  WebSphere ESB Server v6.0  localhost:7779
    Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
      StockQuoteService:getQuote:Filter [requestFlow]
  WebSphere ESB Server v6.0 (WebSphere v6.0)
<terminated> WebSphere v6.0 Server Launcher [WebSphere v6.0 Server Launcher]
  <terminated, exit value: 0> server1 (WebSphere v6.0)
```

Figure 8-23 Break in Filter

15. Again, in the Component view, notice the breakpoint has popped in the Filter component and we have successfully traversed the connection from the Lookup to the Filter(Figure 8-24).

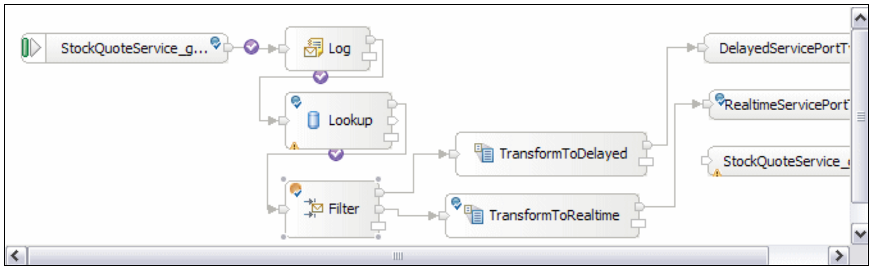
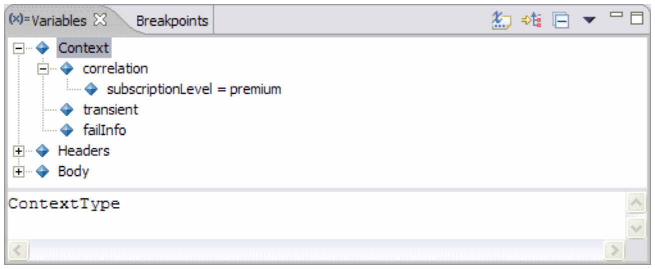


Figure 8-24 Filter icons updated

16. In the Variables view, inspect the Context again, and this time we see the value for the subscriptionLevel has changed to `premium`. This validates the processing done inside of Lookup (Figure 8-25).



Variables Breakpoints

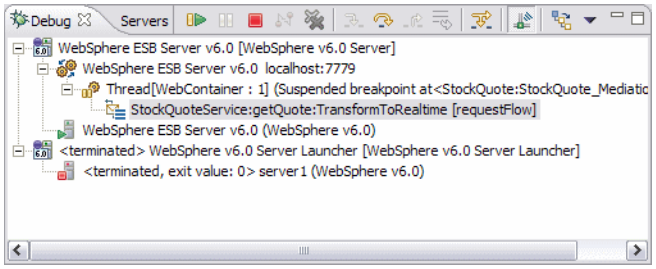
```
Context
  correlation
    subscriptionLevel = premium
  transient
  failInfo
Headers
Body
```

ContextType

Figure 8-25 Variables in Filter

17. Click **Resume** in the Debug view.

18. Execution suspends in the TransformToRealtime component (Figure 8-26).



```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
  WebSphere ESB Server v6.0  localhost:7779
    Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
      StockQuoteService:getQuote:TransformToRealtime [requestFlow]
  WebSphere ESB Server v6.0 (WebSphere v6.0)
<terminated> WebSphere v6.0 Server Launcher [WebSphere v6.0 Server Launcher]
  <terminated, exit value: 0> server1 (WebSphere v6.0)
```

Figure 8-26 Break in TransformToRealtime

19. Now in the Component view, we again see where the breakpoint has been reached, and we again see the connection that was traversed (Figure 8-27). This is particularly helpful, since there were two possible paths that might have been taken.

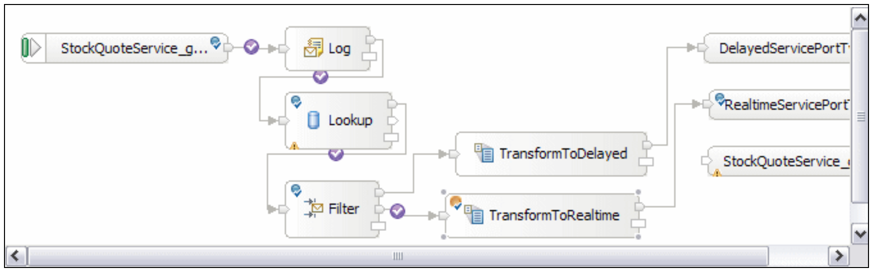
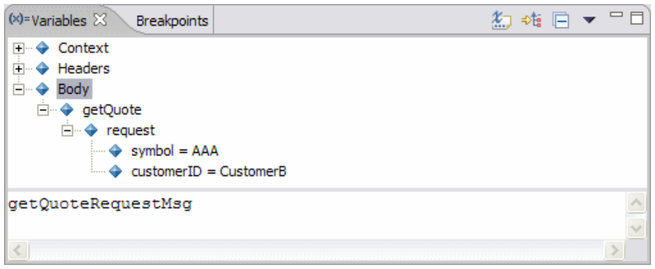


Figure 8-27 TransformToRealtime icons updated

20. In the Variables view, we see there has been no change yet to the message body (Figure 8-28).

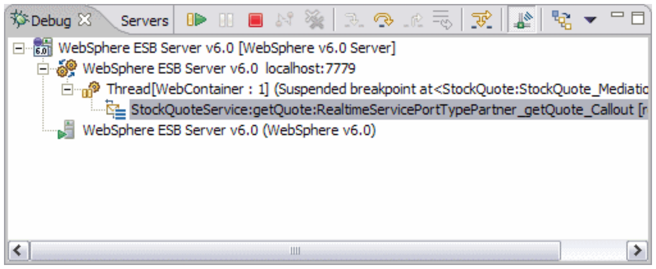


```
Context
Headers
Body
  getQuote
    request
      symbol = AAA
      customerID = CustomerB
getQuoteRequestMsg
```

Figure 8-28 Variables in TransformToRealtime

21. Click **Resume** in the Debug view.

22. Execution is suspended in the RealtimeServicePortTypePartner_getQuote_Callout (Figure 8-29).



WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
WebSphere ESB Server v6.0 localhost:7779
Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
StockQuoteService:getQuote:RealtimeServicePortTypePartner_getQuote_Callout [r
WebSphere ESB Server v6.0 (WebSphere v6.0)

Figure 8-29 Break in getQuote_Callout

23. Again, the icons show us where the breakpoint popped and the path we used to arrive there (Figure 8-30).

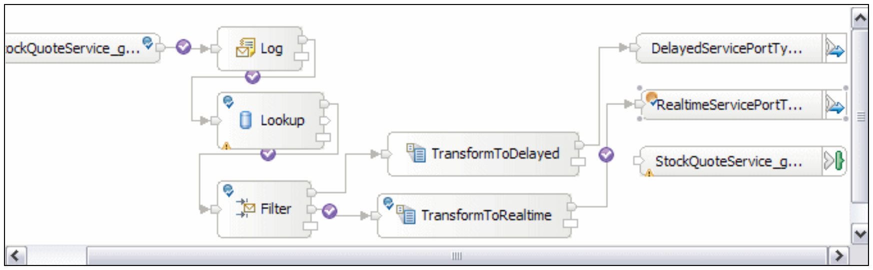
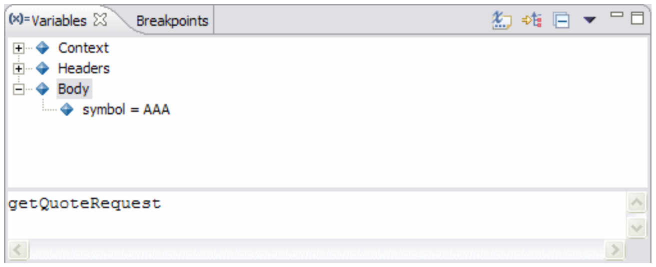


Figure 8-30 getQuote_Callout icons updated

24. In the Variables view, inspect the Body, and notice that the transform has modified the message. It simply carries a single attribute (symbol) now (Figure 8-31).



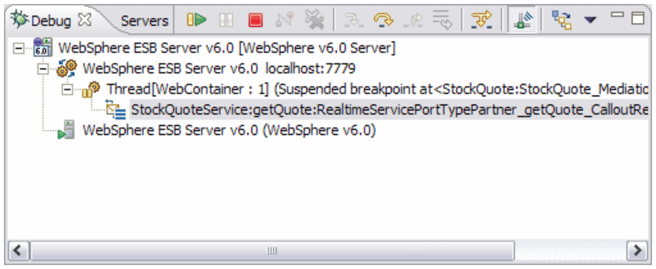
Context
Headers
Body
symbol = AAA

getQuoteRequest

Figure 8-31 Variables in getQuote_Callout

25. Click **Resume** in the Debug view.

26. Execution suspends in the getQuote_CalloutResponse (Figure 8-32 on page 243).



WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
WebSphere ESB Server v6.0 localhost:7779
Thread[WebContainer : 1] (Suspended breakpoint at<StockQuote:StockQuote_Mediatic
StockQuoteService:getQuote:RealtimeServicePortTypePartner_getQuote_CalloutRe
WebSphere ESB Server v6.0 (WebSphere v6.0)

Figure 8-32 Break in getQuote_CalloutResponse

27. Notice the Component view has been updated to display the response flow, and the icon indicates where the breakpoint popped (Figure 8-33).

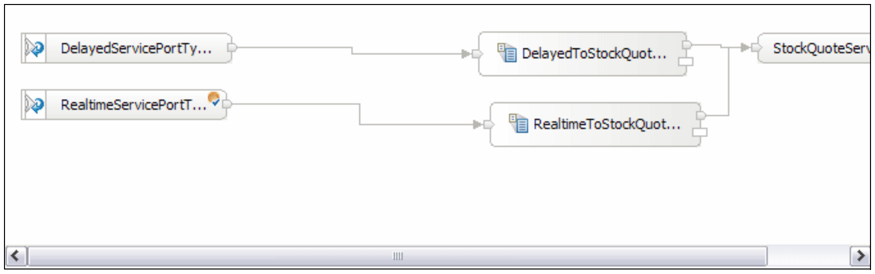
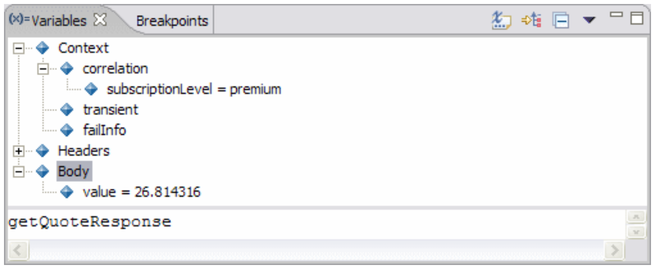


Figure 8-33 getQuote_CalloutResponse icons updated

28. In the Variables view, we can see the value being returned from the service (Figure 8-34).



```
Context
  correlation
    subscriptionLevel = premium
  transient
  failInfo
Headers
Body
  value = 26.814316
getQuoteResponse
```

Figure 8-34 Variables in getQuote_CalloutResponse

29. Click **Resume** in the Debug view. The test completes and the thread terminates.

Of course, there are many more capabilities of the Integration debugger that we have not discussed here, but those we have looked at should give you a very good start at debugging your mediation modules. You might want to experiment with some of the other functions, for example:

- Use the Variables view to actually change the value of a variable.
- Use the Step Over button in the Debug view, to avoid having to set breakpoints in every component in the flow.

8.3 Problem determination facilities

Once your mediation modules are deployed to a production environment it is not likely that you will want to use the Integration debugger to diagnose a problem. It is generally not reasonable to stop a thread and view the state of variables, or single step through the code. In many cases, due to the level of multiprocessing in a production environment, a debugger cannot isolate an application or runtime problem.

It is often necessary to test the problem determination functionality of your applications prior to deploying them to a production environment. Typically, in the later stages of testing, the techniques of debugging an application used in early testing are abandoned in favor of collecting problem determination data to isolate a problem.

8.3.1 Isolating problems with the WebSphere Integration Developer installation

If you have problems successfully installing WebSphere Integration Developer, there are a number of approaches to performing problem determination. See 4.6, “Troubleshooting installation issues” on page 83 for details.

8.3.2 Isolating problems with the WebSphere Enterprise Service Bus installation

If you have problems setting up your WebSphere Enterprise Service Bus runtime environment, see 5.7, “Problem determination for runtime installation and customization” on page 127 for problem determination actions you can take to isolate and resolve them.

8.3.3 Application logging and tracing

The standard logging API provided by Java is found in the java.util.logging package and provides a mechanism for a Java application to write messages and trace entries to a log. For mediation modules, this approach is not very useful, unless you want to use this API to instrument any Java code you write in a custom mediation.

The good news is that your mediation module is already instrumented for you, and you simply have to turn on the CEI events, as described in 8.3.6, “Using the CEI for problem determination” on page 247.

Enabling the CEI events in a production environment will have some impact on performance, so it is not recommended to have them always enabled purely for problem determination purposes. A better approach when using CEI for problem determination is to selectively enable events on specific modules during the execution of a problem scenario, and then to turn the events off, while doing analysis. This is also discussed in 8.3.6, “Using the CEI for problem determination” on page 247

8.3.4 Runtime logging and tracing

The WebSphere Enterprise Service Bus runtime environment makes use of Java logging to provide various levels of messages and traces for the server runtime. You can enable the trace through the administrative console or manually.

Steps to enable trace using the administrative console

Follow these steps to modify the trace settings for your server using the administrative console:

1. Open the administrative console and log in.
2. Expand **Troubleshooting** in the navigation frame.
3. Click **Logs and Trace**
4. Select the server you want to modify
5. Under General Properties, click **Change Log Detail Levels**
6. Paste the following string into the text box:

```
*=info:com.ibm.ws.sibx.*=fine:com.ibm.wsspi.sib.*=all:com.ibm.websphere.
sib.*=all
```

The trace will be sent to the same location as identified above, although you can modify the location by clicking **Diagnostic Trace Service** after you have selected the server you want to modify. Update the File Name field in the dialog to specify the directory and file name you want to be created.

Note: When using the administrative console to update trace settings, you will find both a Configuration tab and a Runtime tab. If you make updates in the Runtime tab, they will be made active immediately. Making updates in the Configuration tab requires you to stop and restart the server for the new trace settings to be in effect.

Steps to enable trace manually

Follow these steps to manually modify the trace settings for your server:

1. Stop your WebSphere Enterprise Service Bus server.
2. Open the file *<esb_install_dir>*/profiles/*<profile_name>*/config/cells/*<cell_name>*/nodes/*<node_name>*/servers/*<server_name>*/server.xml
3. Search for the string `startupTraceSpecification`.
4. Replace its current value with

```
*=info:com.ibm.ws.sibx.*=fine:com.ibm.wsspi.sib.*=all:com.ibm.websphere.
sib.*=all
```

5. Save the file and start the server

The trace will be output to *<esb_install_dir>*/profiles/*<profile_name>*/logs/*<server_name>*/trace.log

Note: If you are running a Network Deployment topology, the manual procedure is not recommended as the Deployment Manager controls the master copy of the server.xml file. It is best to use the administrative console in this case.

8.3.5 Analyzing messages on queue points

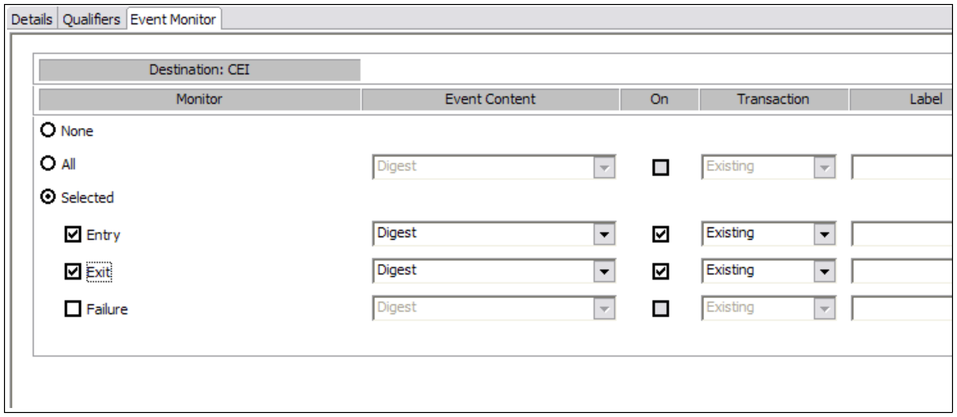
In addition to logs and traces, you may find you need to view the content of a message entering or leaving WebSphere Enterprise Service Bus.

In the administrative console, when you look at a destination on the service integration bus, you can switch to the Runtime tab, and the current message depth is displayed. The same technique can be used to look at any Queue point on the bus.

Both SCA buses (Application and System) also have a SYSTEM.Exception.Destination defined and so looking at the depth of those Queue Points may also be useful for problem determination. These destinations are used to handle messages that cannot be delivered to their intended destinations. There is also an Exception destination associated with the messaging engine, for cases where no explicit exception destination is associated with a bus.

8.3.6 Using the CEI for problem determination

You can enable the generation of CEI events in your mediation module. Use the Event Monitor tab in the Details of the Properties view for a given component. CEI events can be enabled on mediation flows, imports and exports. You need to select the interface and the operation, then the configuration pane for the CEI is shown, as in Figure 8-35 on page 247.


Details | Qualifiers | Event Monitor

Destination: CEI

Monitor	Event Content	On	Transaction	Label
○ None				
○ All	Digest	☐	Existing	
[✓ Selected]				
☑ Entry	Digest	☑	Existing	
☑ Exit	Digest	☑	Existing	
☐ Failure	Digest	☐	Existing	

Figure 8-35 Enabling CEI events

Once the events are enabled, when your module executes, they will be written to the database associated with the CEI datasource. In the development environment that is typically Cloudscape.

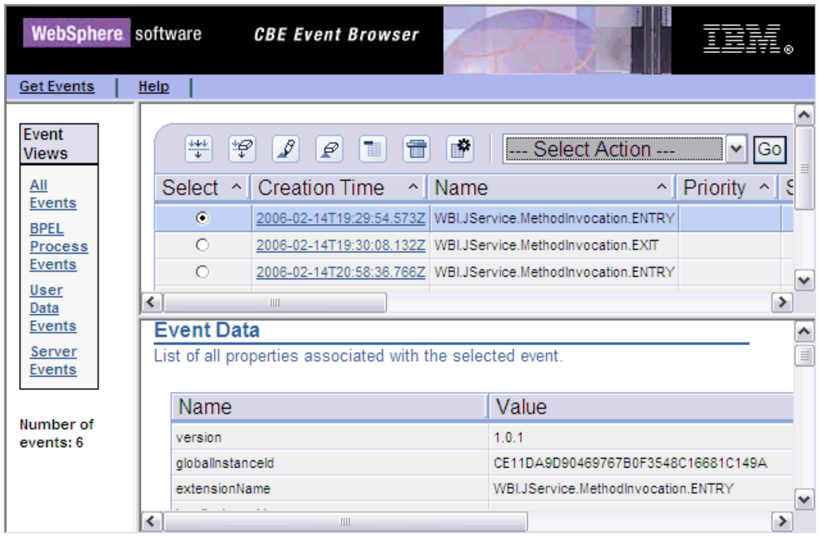
To view the data, you can launch the Cloudscape viewer, located in the *<install_root>*\runtimes\bi_v6\cloudscape\bin directory. The databases are located in the directory associated with the server profile. So, for a typical WebSphere Integration Developer installation that would be *<install_root>*\pf\esb\events. In a stand-alone server runtime environment, you will find an events subdirectory in the profile directory for the server. Viewing the raw event data in the Cloudscape viewer can be quite complicated, requiring you to understand the schema to find the data you are interested in.

An alternative to viewing the raw data is to launch the CBE Event Browser. Select the test server in WebSphere Integration Developer, right-click and select **Launch** -> **CBE Event Browser**. This is disabled for an WebSphere Enterprise

Service Bus server, but you can still invoke the application from a Web browser with the following URL:

```
http://localhost:9060/ibm/console/cbebrowser
```

The CBE Event Browser can be used to display the list of events and the event detail as shown in Figure 8-36 on page 248.



Select	Creation Time	Name	Priority
●	2006-02-14T19:29:54.573Z	WBI.JService.MethodInvocation.ENTRY	
○	2006-02-14T19:30:08.132Z	WBI.JService.MethodInvocation.EXIT	
○	2006-02-14T20:58:36.766Z	WBI.JService.MethodInvocation.ENTRY	

Event Data

List of all properties associated with the selected event.

Name	Value
version	1.0.1
globalInstanceId	CE11DA9D90469767B0F3548C16681C149A
extensionName	WBI.JService.MethodInvocation.ENTRY

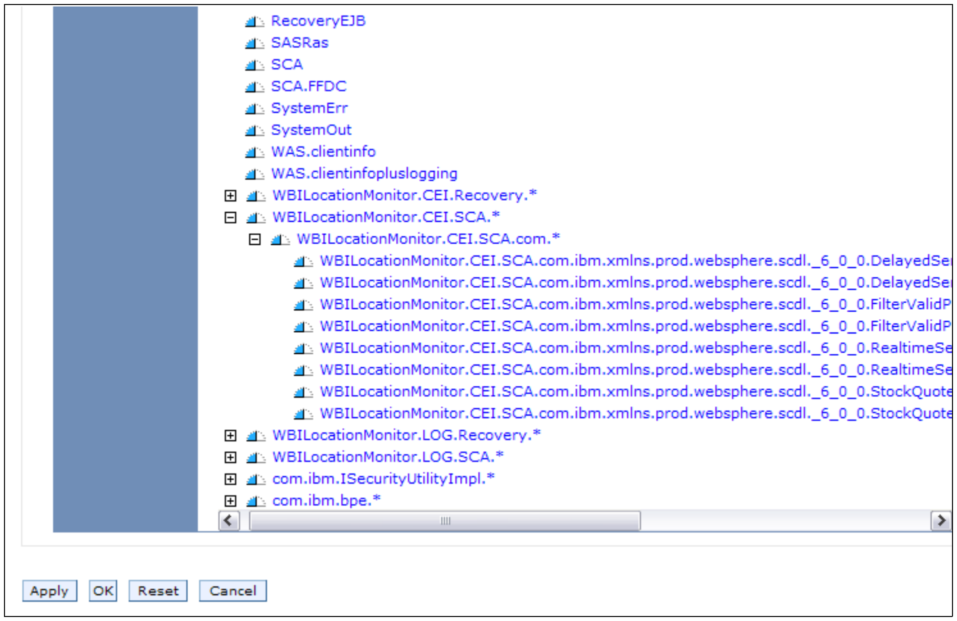
Figure 8-36 CBE Event Browser

Click the **Get Events** link in the upper left corner of the page, to retrieve all of the events that are recorded in the database. The selections in the navigation pane are then used to work with the list of events that have been retrieved.

The CEI information may be useful to trace through the flow of your mediation module. By using the entry and exit events you can determine the path taken during execution. In an integration test environment, setting failure events on can aid problem determination. You may also be able to gain some initial performance metrics for each of the components that are invoked in a mediation module, as the entry and exit events contain a timestamp.

In the runtime environment, an administrator can make changes to CEI event recording. This is accomplished in the administrative console by expanding **Troubleshooting** in the navigation pane. Select **logs and trace ->** **`your_server`-> Change Log Detail Levels**. You can make changes in the Configuration, which will be effective the next time the server is started, or you can make changes in the Runtime, which will become effective immediately.

When the details screen is displayed, scroll down and expand **WBILocationMonitor.CEI.SCA.*** and then expand **WBILocationMonitor.CEI.SCA.com.***. The list of deployed modules that can have their CEI event recording changed is displayed, as shown in Figure 8-37.



```
RecoveryEJB
SASRas
SCA
SCA.FFDC
SystemErr
SystemOut
WAS.clientinfo
WAS.clientinfopluslogging
⊞ WBILocationMonitor.CEI.Recovery.*
⊟ WBILocationMonitor.CEI.SCA.*
  ⊟ WBILocationMonitor.CEI.SCA.com.*
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.DelayedSe
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.DelayedSe
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.FilterValidP
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.FilterValidP
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.RealtimeSe
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.RealtimeSe
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.StockQuote
      WBILocationMonitor.CEI.SCA.com.ibm.xmlns.prod.websphere.scdl._6_0_0.StockQuote
⊞ WBILocationMonitor.LOG.Recovery.*
⊞ WBILocationMonitor.LOG.SCA.*
⊞ com.ibm.ISecurityUtilityImpl.*
⊞ com.ibm.bpe.*
```

Figure 8-37 Changing CEI event recording

When you click on any of the modules, the context menu displayed in Figure 8-38 is displayed. To turn events on click **all**, and to turn them off, click **off**.

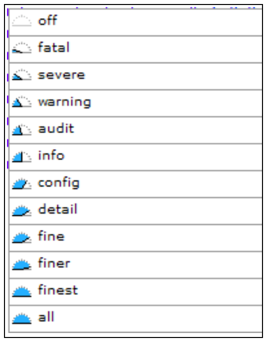
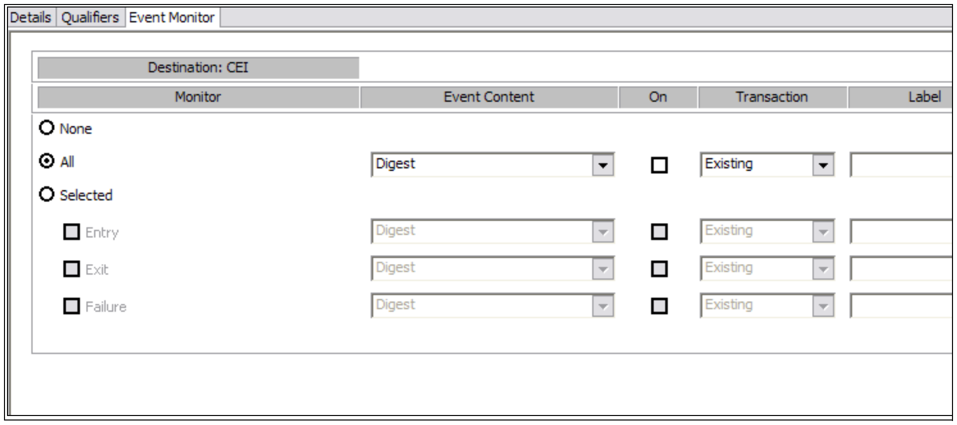


Figure 8-38 Trace detail levels

The behavior of CEI event recording is dependent on the combination of what is specified in WebSphere Integration Developer and the state of event recording in the runtime.

In Figure 8-35 on page 247, we saw how an integration developer can explicitly turn on specific CEI events for the components. Notice in that figure that the **On** checkbox is checked. If the module is deployed with those events checked, they will always be recorded, regardless of what the administrator attempts to do. In some cases, that is desirable. If the events are needed for monitoring or other processing, then it is a good idea to explicitly turn them on and make sure the checkbox is checked. However, the purpose of this discussion is problem determination, generally it would be best to be able to keep the event recording off until needed. In that case, deploy your modules with None or All (leaving the On checkbox unchecked), as shown in Figure 8-39 on page 250.



Destination: CEI

Monitor	Event Content	On	Transaction	Label
○ None				
⦿ All	Digest	☐	Existing	
○ Selected				
☐ Entry	Digest	☐	Existing	
☐ Exit	Digest	☐	Existing	
☐ Failure	Digest	☐	Existing	

Figure 8-39 Allowing CEI events recording to be changed in the runtime

Note: In our testing, when we modified the event recording for an Export component, it had no effect. Events were not recorded.

Tip: Deploy mediation modules to production runtime environments with monitor setting of None if you intend for the runtime administrator to have full control over enabling and disabling event recording.

9

Administering WebSphere Enterprise Service Bus

This chapter discusses the administration aspects of WebSphere Enterprise Service Bus V6.0.1.

WebSphere Enterprise Service Bus V6.0.1 is based on the WebSphere Application Server platform foundation and is built on top of WebSphere Application Server Network Deployment. WebSphere Enterprise Service Bus inherits all of the administration functionality from WebSphere Application Server Network Deployment including clustering, fail-over and security. Besides this, WebSphere Enterprise Service Bus introduces new administration tasks to deploy, administer and manage mediation modules, and service integration applications.

This chapter focuses only on the new administration tasks introduced in WebSphere Enterprise Service Bus. For information on administration tasks common to WebSphere Application Server Network Deployment and high availability, consult the following redbooks:

- *WebSphere Application Server V6: System Management and Configuration Handbook*, SG24-6451
- *WebSphere Application Server Network Deployment V6: High Availability Solutions,* SG24-6688

9.1 Administrative console

The WebSphere Enterprise Service Bus administrative console provides a way for system administrators to configure WebSphere Enterprise Service Bus.

For simplifying administration in WebSphere Enterprise Service Bus, the Welcome screen in administrative console presents a way to filter administrative tasks. You can choose to select **All, Application Integration** or **Server and Bus** (Figure 9-1). This helps reduce the complexity of administrative console by hiding functionality that is not applicable to a specific administrator's task. The filter can be modified at any time from the Welcome screen.

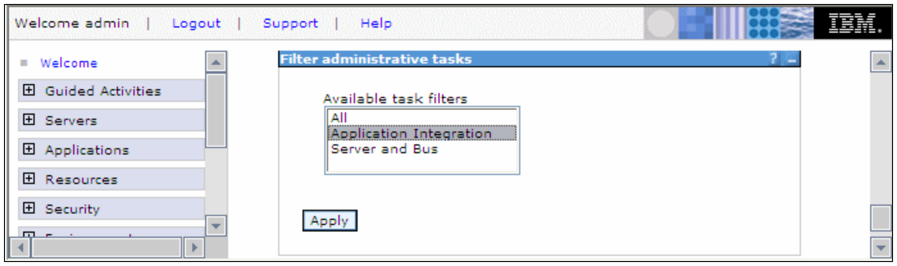


Figure 9-1 Task filter

The filters are:

- Application Integration filter

  Shows the options to deploy and manage mediation modules and service integration applications.
- Server and Bus filter

  Provides the ability to configure buses, servers, and resources, besides deploying and managing mediation modules and service integration applications.
- All

  Shows all the capabilities of the administrative console.

The main areas in the administrative console where WebSphere Enterprise Service Bus resources and services can be administered are as shown in Figure 9-2 on page 253.

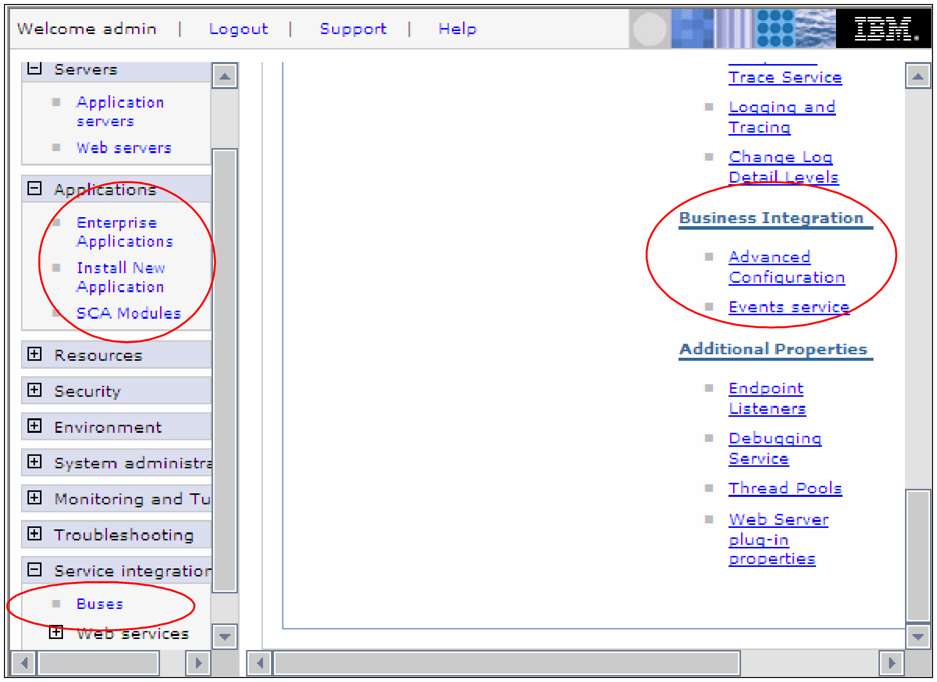


Figure 9-2 Areas of administration

Mediation modules use the resources provided by the service integration technology in WebSphere Application Server. Administration of these resources is the same as in WebSphere Application Server. WebSphere Enterprise Service Bus can also be administered from the command line using the command `wsadmin`.

Most of the functionality shown in Figure 9-2, is the same as for WebSphere Application Server. *SCA Modules* under *Applications* allows the administrator to manage mediation modules.

Clicking on SCA Modules provides an administrator with features to manage mediation modules and perform functions such as:

- Listing mediation modules that are deployed to WebSphere Enterprise Service Bus
- Starting or stopping mediation modules
- Inspecting module components like imports and exports
- Dynamically changing wiring between modules at runtime without a need to restart the server.

These features are further discussed in 9.3, “Mediation module administration” on page 261

Clicking on **Advanced Configuration** under Business Integration allows the administrator to configure the server to host SCA applications. This is discussed in detail in 5.4.3, “Final configuration steps” on page 98.

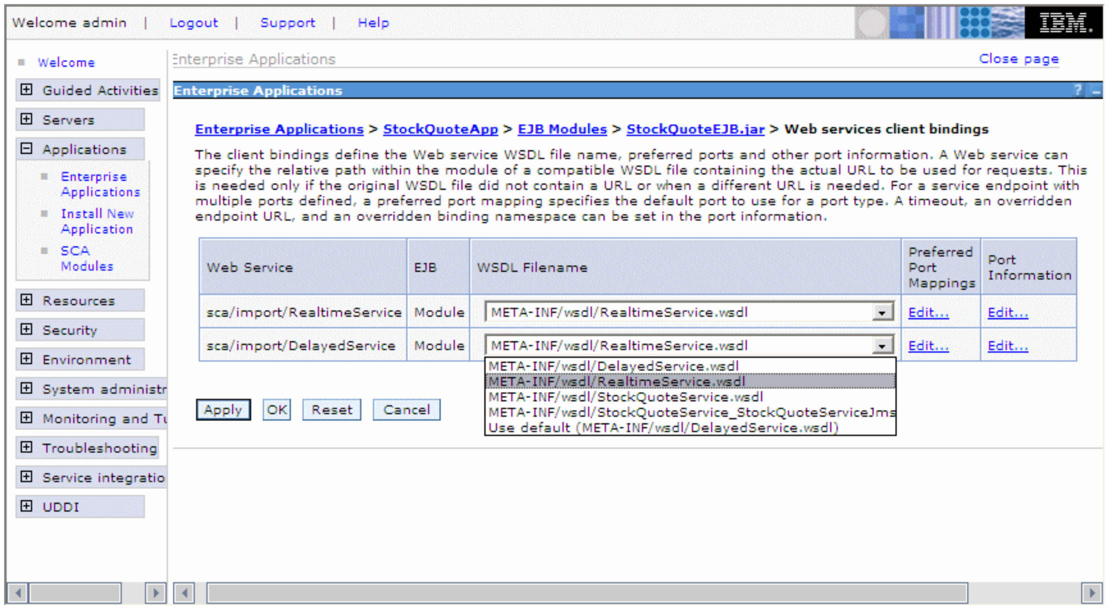
9.2 Deploying mediation modules

Depending on the mediation module, an administrator may have some tasks to complete before deploying the module. This will ensure a successful deployment and a working application. This section discusses some of the common tasks to consider before a deployment.

9.2.1 Configuring Web service bindings

If the mediation module being deployed is using a Web service binding on an Import, an administrator must be ensure the Web service binding is using the correct host name and port number on which the Web service provider has been bound.

While a mis-match in the port will not affect the deployment, it will lead to failure of the service integration solution. It is possible to change the Web service binding at runtime after the mediation module is deployed. This is done using administrative console by clicking on **Applications -> Enterprise Applications ->** <application_instance> **-> EJB modules ->** <module_instance> **-> Web services client bindings** as shown in Figure 9-3 on page 255.


Welcome admin | Logout | Support | Help

Enterprise Applications

Enterprise Applications > StockQuoteApp > EJB Modules > StockQuoteEJB.jar > Web services client bindings

The client bindings define the Web service WSDL file name, preferred ports and other port information. A Web service can specify the relative path within the module of a compatible WSDL file containing the actual URL to be used for requests. This is needed only if the original WSDL file did not contain a URL or when a different URL is needed. For a service endpoint with multiple ports defined, a preferred port mapping specifies the default port to use for a port type. A timeout, an overridden endpoint URL, and an overridden binding namespace can be set in the port information.

Web Service	EJB	WSDL Filename	Preferred Port Mappings	Port Information
sca/import/RealtimeService	Module	META-INF/wsdl/RealtimeService.wsdl	Edit...	Edit...
sca/import/DelayedService	Module	META-INF/wsdl/RealtimeService.wsdl	Edit...	Edit...

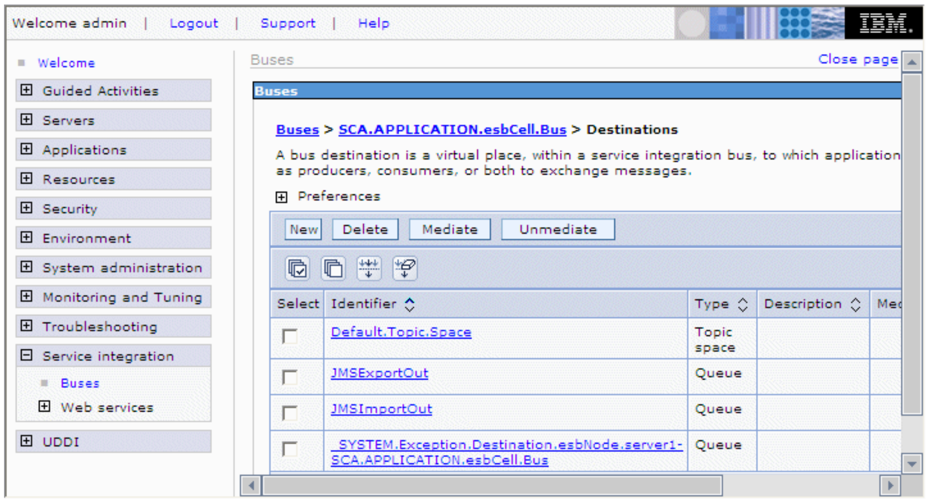
```
META-INF/wsdl/DelayedService.wsdl
META-INF/wsdl/RealtimeService.wsdl
META-INF/wsdl/StockQuoteService.wsdl
META-INF/wsdl/StockQuoteService_StockQuoteServiceJms
Use default (META-INF/wsdl/DelayedService.wsdl)
```

Figure 9-3 Change Web services client binding

The new Web service binding can be selected from the drop down box if its already existing on the server, or can be provided by clicking on **Edit** under Port Information.

9.2.2 Configuring JMS bindings

If the mediation module being deployed uses JMS bindings or SOAP over JMS bindings, the JMS resources required for the module should be defined on either your own bus, or on the default SCA.APPLICATION.esbCell.Bus bus (Figure 9-4 on page 256). These resources must also be defined as JMS resources for the respective JMS provider.



Buses > SCA.APPLICATION.esbCell.Bus > Destinations

A bus destination is a virtual place, within a service integration bus, to which application as producers, consumers, or both to exchange messages.

Select	Identifier	Type	Description	Me
☐	Default.Topic.Space	Topic space		
☐	JMSExportOut	Queue		
☐	JMSImportOut	Queue		
☐	_SYSTEM.Exception.Destination.esbNode.server1-SCA.APPLICATION.esbCell.Bus	Queue		

Figure 9-4 Defining destinations

The JMS resources required depend on where and how JMS binding is being used by a mediation module:

- JMS binding on an Import

  The JMS resources that need to be created for an Import are:

  – A queue destination on the bus.

  – A queue connection factory resource that is used to connect to the bus hosting the queue destination.

  – A queue resource that provides the JNDI name for the queue destination on the bus.

- JMS binding on an Export

  The JMS resources that need to be created for an Export depend on the type of operation being used on the interface.

  If the operation is a one way operation, the JMS resources that need to be created are similar to the resources needed for JMS binding on an Import (as discussed above). In addition an activation specification needs to be created that can be used by the mediation module to register the queue with a message listener in the mediation module.

  If the operation is a request response operation, the following resources are required:

  – A queue destination for the request

  – A queue destination for the response

  – JNDI resources for the queue connection factory resource that is used to connect to the bus hosting these queue destinations, and the queue resources that provides the JNDI name for the request and response queue destination on the bus.

  Besides these resources, you will also have to create an activation specification that is used by the mediation module to register the input queue with the message listener in the mediation module.

- SOAP over JMS binding

  The JMS resources needed for this type of binding are similar to the resources needed for a JMS binding on an Import as discussed above.

9.2.3 Methods to deploy service mediation modules

Depending on the environment, a mediation module can be deployed in one of the following ways:

- From WebSphere Integration Developer by right clicking on a server in the Servers view and selecting **Add and remove projects**.
- By exporting a deployable EAR file from WebSphere Integration Developer and installing it into WebSphere Enterprise Service Bus using the administrative console
- By exporting a deployable EAR file from WebSphere Integration Developer and installing it into WebSphere Enterprise Service Bus using the **`wsadmin`** command line utility.
- By exporting a deployable EAR file from WebSphere Integration Developer and installing it into WebSphere Enterprise Service Bus using the **`serviceDeploy`** command line utility.

Installing a mediation module using wsadmin

Installing the mediation module EAR file is similar to installing a WebSphere Application Server application. Use the **`wsadmin`** command line utility. Example 9-1 shows an example of usage.

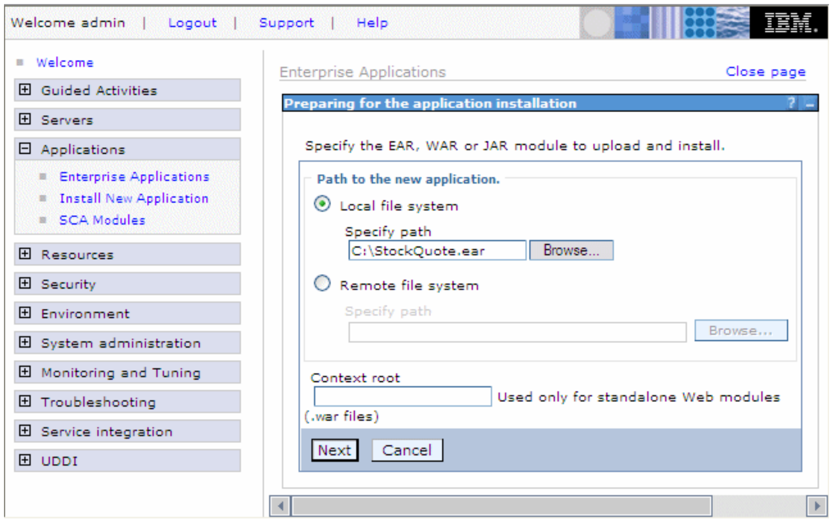
Example 9-1 Install using wsadmin

```
wsadmin -c “$AdminApp install StockQuote.ear”
```

Installing a mediation module using the administrative console

A mediation module can be installed using administrative console as follows:

1. Log into the administrative console, expand **Applications**, and click **Install New Application**.
2. Browse to the location of your EAR file (Figure 9-5) then click **Next**.



Specify the EAR, WAR or JAR module to upload and install.

Path to the new application.

[✓ Local file system]

Specify path: C:\StockQuote.ear

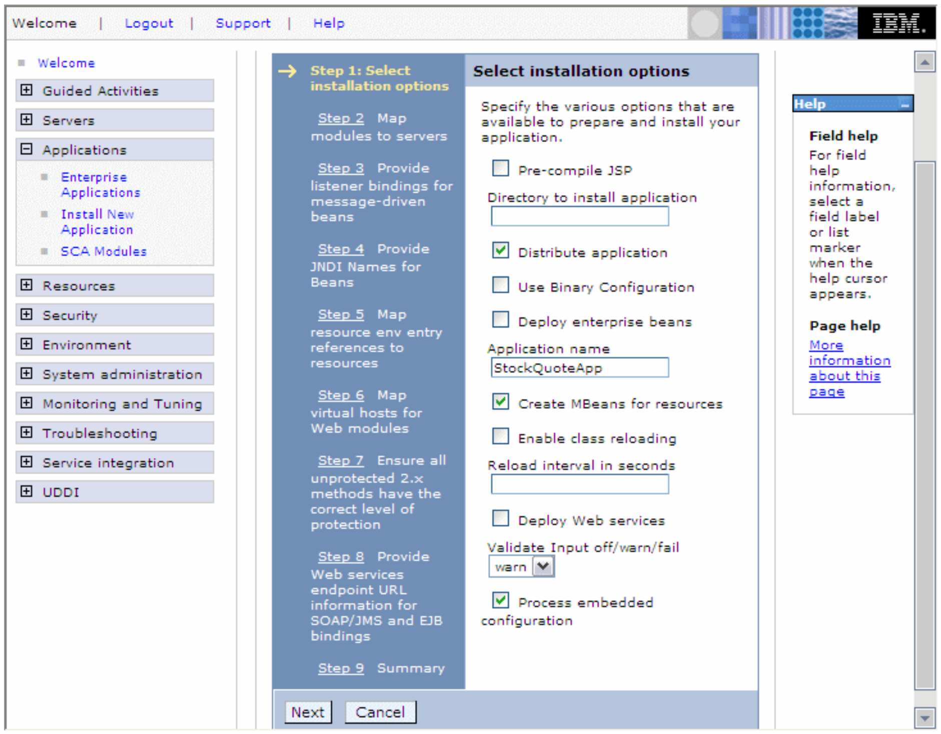
Remote file system

Specify path:

Context root: Used only for standalone Web modules (.war files)

Figure 9-5 Install application

3. Check the box **Generate Default Bindings** to use default bindings and click **Next**.
4. The following screens allow you to specify deployment options for the installation. In our example, this screens shows there are eight steps to install a mediation module (see Figure 9-6).


Welcome | Logout | Support | Help
IBM.
Welcome
Guided Activities
Servers
Applications
Enterprise Applications
Install New Application
SCA Modules
Resources
Security
Environment
System administration
Monitoring and Tuning
Troubleshooting
Service integration
UDDI
Step 1: Select installation options
Step 2 Map modules to servers
Step 3 Provide listener bindings for message-driven beans
Step 4 Provide JNDI Names for Beans
Step 5 Map resource env entry references to resources
Step 6 Map virtual hosts for Web modules
Step 7 Ensure all unprotected 2.x methods have the correct level of protection
Step 8 Provide Web services endpoint URL information for SOAP/JMS and EJB bindings
Step 9 Summary
Next
Cancel
Help
Field help
Page help
More information about this page

Select installation options

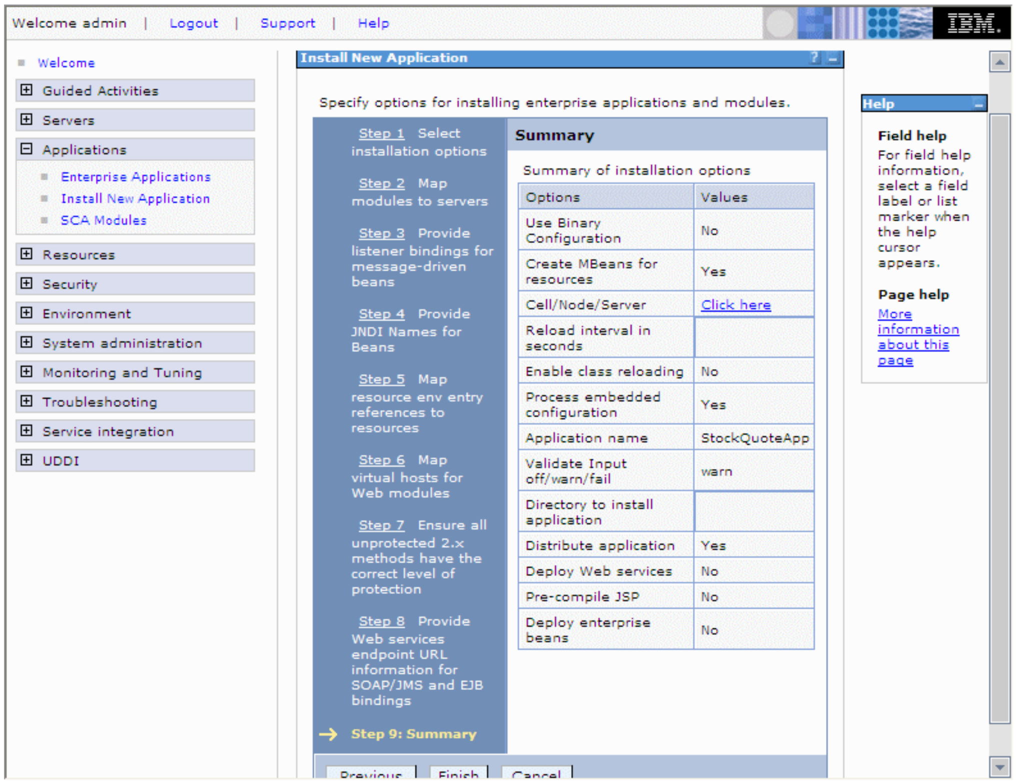
Specify the various options that are available to prepare and install your application.

- [ ] Pre-compile JSP
- Directory to install application:
- [✓] Distribute application
- [ ] Use Binary Configuration
- [ ] Deploy enterprise beans
- Application name: StockQuoteApp
- [✓] Create MBeans for resources
- [ ] Enable class reloading
- Reload interval in seconds:
- [ ] Deploy Web services
- Validate Input off/warn/fail: warn
- [✓] Process embedded configuration

For field help information, select a field label or list marker when the help cursor appears.

Figure 9-6 Install steps

5. Complete (or bypass) each step until your read the Summary page (Figure 9-7 on page 260).



Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

Step 2 Map modules to servers

Step 3 Provide listener bindings for message-driven beans

Step 4 Provide JNDI Names for Beans

Step 5 Map resource env entry references to resources

Step 6 Map virtual hosts for Web modules

Step 7 Ensure all unprotected 2.x methods have the correct level of protection

Step 8 Provide Web services endpoint URL information for SOAP/JMS and EJB bindings

→ **Step 9: Summary**

Summary

Summary of installation options

Options	Values
Use Binary Configuration	No
Create MBeans for resources	Yes
Cell/Node/Server	Click here
Reload interval in seconds	
Enable class reloading	No
Process embedded configuration	Yes
Application name	StockQuoteApp
Validate Input off/warn/fail	warn
Directory to install application	
Distribute application	Yes
Deploy Web services	No
Pre-compile JSP	No
Deploy enterprise beans	No

Help

Field help
For field help information, select a field label or list marker when the help cursor appears.

Page help
More information about this page

Figure 9-7 Install summary

6. Click **Finish** to start the install of the enterprise application containing the mediation module. This will start creating and configuring all the resources needed for the application. A report will show the status of the application install. After successful completion of install a message will be generated as shown in Example 9-2.

Example 9-2 Install message

```
ADMA5013I: Application StockQuoteApp installed successfully.

Application StockQuoteApp installed successfully.
```

7. You must save the changes to the server configuration before starting the application.

Installing a mediation module using service deploy

WebSphere Enterprise Service Bus provides a command line utility called **serviceDeploy** that can be used to build deployable mediation modules from zip or jar files containing service components. A mediation module can be exported from WebSphere Integration Developer to be used later by the serviceDeploy command for generation of deployable EAR file.

If a mediation module is exported from WebSphere Integration Developer as a zip file or a jar file, it will not contain any deployable code, only files that describe the module and it's components.

Running serviceDeploy will generate an installable EAR file, containing all the deployable code required for the module to run as a service application, on WebSphere Enterprise Service Bus. An example of using serviceDeploy is shown in Example 9-3.

Example 9-3 Using Service Deploy

```
serviceDeploy.bat c:\temp\MyModule.zip -outputApplication MyModule.ear
```

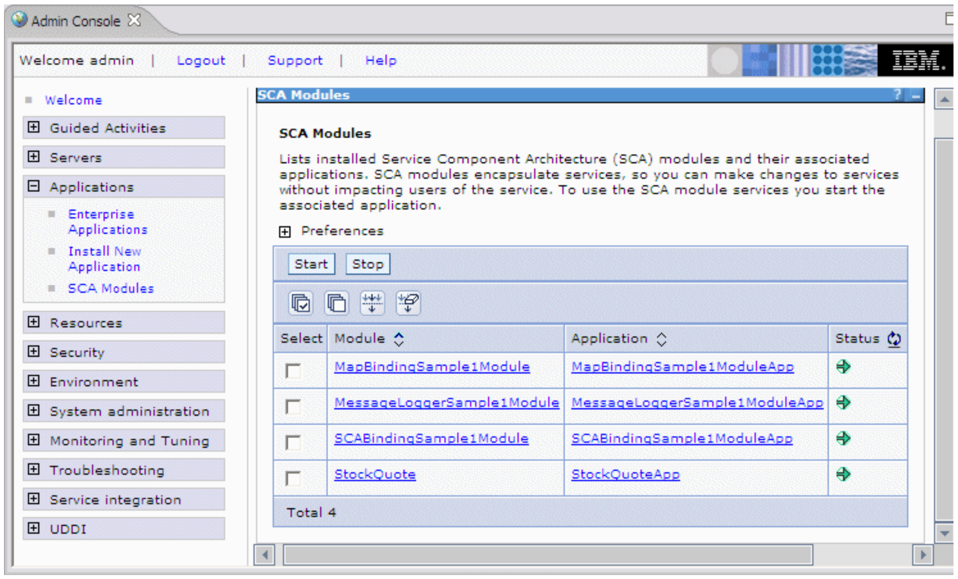
The utility serviceDeploy is commonly used by development teams using a version control system. After developers check in their mediation module projects into a source code repository, the modules can be extracted and built into installable EAR files using serviceDeploy. This can be useful for deploying the application in a system test environment or a production environment. Using this method ensures the runtime code is not checked in but generated when required.

9.3 Mediation module administration

WebSphere Enterprise Service Bus allows users to view deployed mediation modules in the administrative console or using the command line. Details of the components inside that module, such as imports and exports, can also be displayed. SCA bindings defined in a mediation module import can be modified at runtime to change the flow of messages through the bus.

9.3.1 Displaying SCA modules

A mediation module is a type of SCA module, therefore in the administrative console, mediation modules can be found under **Applications** -> **SCA Modules** (Figure 9-8).



SCA Modules

Lists installed Service Component Architecture (SCA) modules and their associated applications. SCA modules encapsulate services, so you can make changes to services without impacting users of the service. To use the SCA module services you start the associated application.

Select	Module	Application	Status
☐	MapBindingSample1Module	MapBindingSample1ModuleApp	➔
☐	MessageLoggerSample1Module	MessageLoggerSample1ModuleApp	➔
☐	SCABindingSample1Module	SCABindingSample1ModuleApp	➔
☐	StockQuote	StockQuoteApp	➔

Total 4

Figure 9-8 SCA modules in the administrative console

To list the mediation modules deployed to WebSphere Enterprise Service Bus using wsadmin run the command **`$AdminTask listSCAModules`**. An example is shown in Example 9-4 on page 262

Example 9-4 listSCAModules

```
wsadmin -c “$AdminTask listSCAModules”
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
SCABindingSample1Module:SCABindingSample1ModuleApp
StockQuote:StockQuoteApp
MessageLoggerSample1Module:MessageLoggerSample1ModuleApp
MapBindingSample1Module:MapBindingSample1ModuleApp
```

You can display more information about the mediation module by clicking on the module name in the administrative console, or by using the wsadmin and run the command **`$AdminTask showSCAModule`** specifying the moduleName as shown in Example 9-5 on page 262.

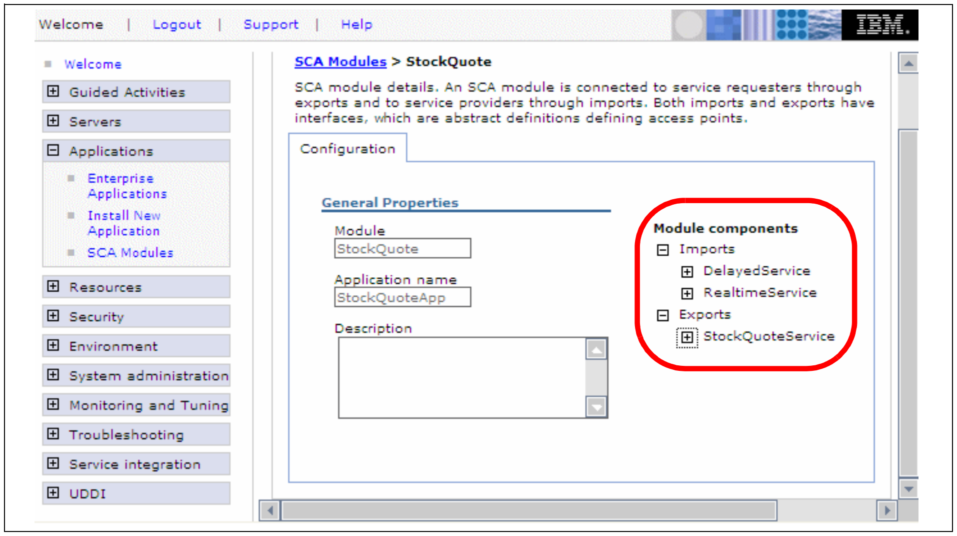
Example 9-5 showSCAModule

```
wsadmin -c “$AdminTask showSCAModule {-moduleName StockQuote}”
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
name:StockQuote
```

```
description:null
```

9.3.2 Displaying imports and exports

In the administrative console, clicking on a module lists its components. In the case of a mediation module this will include its imports and exports (Figure 9-9).



SCA Modules > StockQuote

SCA module details. An SCA module is connected to service requesters through exports and to service providers through imports. Both imports and exports have interfaces, which are abstract definitions defining access points.

Module: StockQuote

Application name: StockQuoteApp

Description

Module components
- Imports
  - DelayedService
  - RealtimeService
- Exports
  - StockQuoteService

Figure 9-9 Imports and exports in the administrative console

To display a mediation modules imports using wsadmin run the command **`$AdminTask listSCAImports`** specifying the moduleName. An example is shown in Example 9-6 on page 263.

Example 9-6 listSCAImports

```
wsadmin -c “$AdminTask listSCAImports {-moduleName StockQuote}”
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
DelayedService
RealtimeService
```

To display a mediation modules exports using wsadmin run the command **`$AdminTask listSCAExports`** specifying the moduleName. An example is shown in Example 9-7 on page 264

Example 9-7 listSCAExports

```
wsadmin -c "$AdminTask listSCAExports {-moduleName StockQuote}"
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
StockQuoteService
```

To display more information about an import run the command **`$AdminTask showSCAImport`** specifying the module name and the import. An example is shown in Example 9-8 on page 264.

Example 9-8 showImport

```
wsadmin -c "$AdminTask showSCAImport {-moduleName StockQuote -import
DelayedService}"
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
import:name=DelayedService,description=null
interface:type=WSDLPortType,portType=ns1:DelayedServicePortType
```

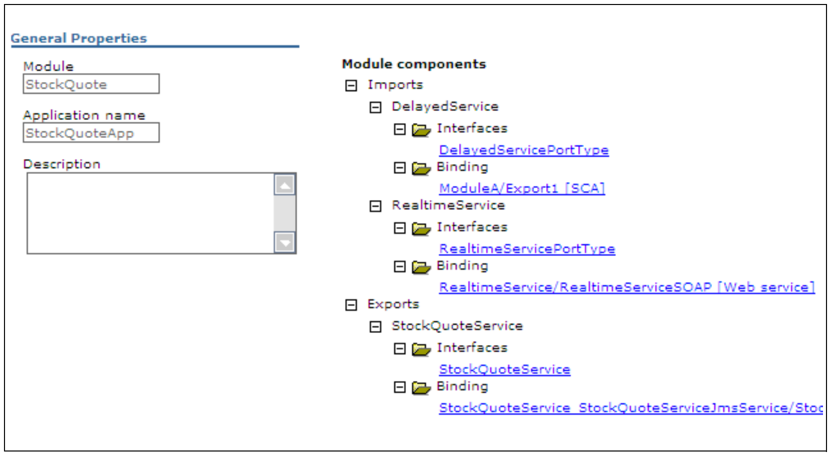
To display more information about an export run the command **`$AdminTask showSCAExport`** specifying the module name and the export. An example is shown in Example 9-9 on page 264.

Example 9-9 showExport

```
wsadmin -c "$AdminTask showSCAExport {-moduleName StockQuote -export
StockQuoteService}"
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
export:name=StockQuoteService,description=null
interface:type=WSDLPortType,portType=ns1:StockQuoteService
```

9.3.3 Displaying interfaces and bindings

In the administrative console, imports and exports can be expanded to view their interfaces and bindings (Figure 9-10 on page 265).



General Properties

Module
StockQuote

Application name
StockQuoteApp

Description

Module components
- Imports
  - DelayedService
    - Interfaces
      - DelayedServicePortType
    - Binding
      - ModuleA/Export1 [SCA]
  - RealtimeService
    - Interfaces
      - RealtimeServicePortType
    - Binding
      - RealtimeService/RealtimeServiceSOAP [Web service]
- Exports
  - StockQuoteService
    - Interfaces
      - StockQuoteService
    - Binding
      - StockQuoteService_StockQuoteServiceJmsService/Stoc

Figure 9-10 Interfaces and bindings in the administrative console

To display an import binding using wsadmin run the command **`$AdminTask showSCAImportBinding`** specifying the module name and import. An example is shown in Example 9-10 on page 265

Example 9-10 showSCAImportBinding

```
wsadmin -c “$AdminTask showSCAImportBinding {-moduleName StockQuote -import
DelayedService}”
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
importBinding:type=WebServiceImportBinding,endpoint=http://localhost:9080/Delay
edService/services/DelayedServiceSOAP,port=ns1:DelayedServiceSOAP,service=ns1:D
elayedService
```

To display an export binding using wsadmin run the command **`$AdminTask showSCAExportBinding`** specifying the module name and export. An example is shown in Example 9-11 on page 265

Example 9-11 showSCAExportBinding

```
wsadmin - c “$AdminTask showSCAExportBinding {-moduleName StockQuote -export
StockQuoteService}”
WASX7209I: Connected to process "server1" on node esbNode using SOAP connector;
 The type of process is: UnManagedProcess
exportBinding:type=WebServiceExportBinding,port=_:StockQuoteService_StockQuoteS
erviceJmsPort,service=_:StockQuoteService_StockQuoteServiceJmsService
```

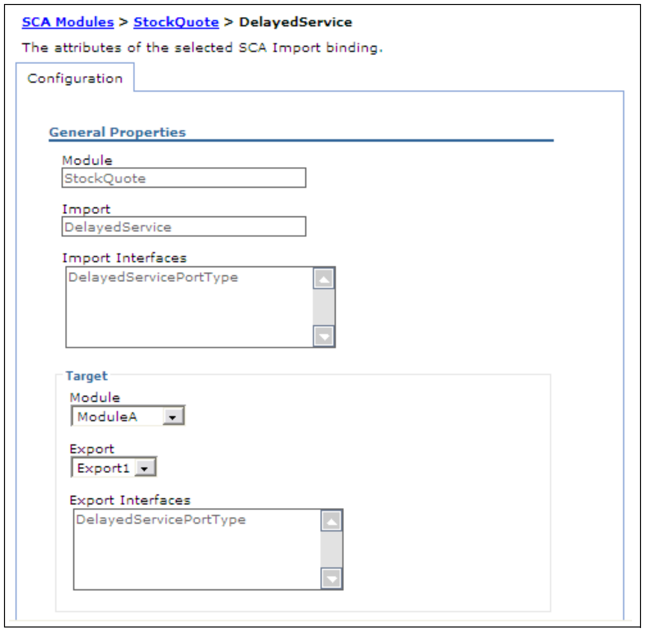
9.3.4 Changing bindings

Using the administrative console or wsadmin it is possible to modify a module's SCA import bindings, giving the administrator the ability to change a deployed flow. It effectively allows a flow of mediation modules to be rewired at runtime, eliminating the need for a developer to use WebSphere Integration Developer to modify, export and redeploy, and also does not require a server restart.

To modify SCA binding using the administrative console, expand **Applications**, click on **SCA Modules** -> <module_instance>, expand **Imports**, expand <import>, expand **Binding** and click on the SCA binding you wish to modify.

Tip: The binding type is shown in square brackets after the name of the binding.

This displays the properties of the SCA binding including the module and the export that this binding refers to (Figure 9-11 on page 266).


SCA Modules > StockQuote > DelayedService

The attributes of the selected SCA Import binding.

Configuration

General Properties

Module: StockQuote

Import: DelayedService

Import Interfaces: DelayedServicePortType

Target

Module: ModuleA

Export: Export1

Export Interfaces: DelayedServicePortType

Figure 9-11 An SCA binding in the administrative console

In the target section of the properties panel the drop down lists can be used to change the SCA import to use a different export from the same module, or change modules and select a new export (Figure 9-12 on page 267).

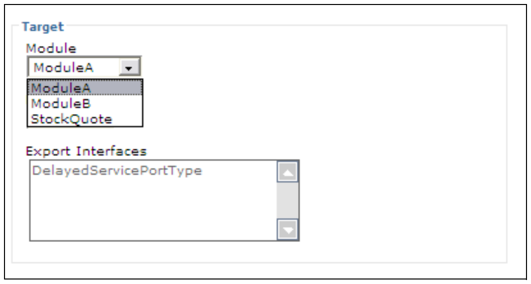


Figure 9-12 Changing an SCA binding using the administrative console

Once the required modifications have been made, save the changes and they will take effect immediately.

To change an SCA binding using wsadmin run the command **`$AdminTask modifySCAImportSCABinding`** specifying the moduleName, import, targetModule and targetExport. An example is shown in Example 9-12 on page 267

Example 9-12 modifySCAImportSCABinding

```
wsadmin -c “$AdminTask modifySCAImportSCABinding {-moduleName StockQuote
-import StockQuoteService -targetModule ModuleB -targetExport Export1}”
```

Tip: It is not currently possible to modify binding types other than SCA using this method.

Part 4

Development examples

10

Preparing for the development examples

This chapter describes the steps necessary to prepare your workspace for the development examples described in:

- Chapter 12, “Developing mediation logic using mediation primitives” on page 369
- Chapter 11, “Developing integration logic using mediation modules” on page 279
- Chapter 13, “Configuring modules to provide quality of service” on page 441

Complete this chapter before attempting any of the development samples. This chapter assumes WebSphere Integration Developer V6.0.1 has been installed. It contains the following:

- An overview of the development examples in this book
- Preparing your environment

10.1 An overview of the development examples in this book

Part 3 of this redbook describes a number of development examples, each of which provides step-by-step instructions that explain how to implement specific functions within WebSphere Enterprise Service Bus.

The development examples are grouped logically into three sections:

- Developing mediation logic using mediation primitives

  Describes how to use the mediation primitives features of WebSphere Enterprise Service Bus to build mediation flows. Each mediation primitive is described, and individual step-by-step instructions are provided.

  These development examples are located in Chapter 12, “Developing mediation logic using mediation primitives” on page 369.

- Developing integration logic using mediation modules

  Describes how to import services into an ESB, expose services on the ESB to clients, and how to map services on an ESB. Step-by-step instructions are provided.

  These development examples are located in Chapter 11, “Developing integration logic using mediation modules” on page 279.

- Configuring modules that provide quality of service

  Provides step-by-step instructions for configuring quality of service settings, including CEI events, security, and transactions.

  These development examples are located in Chapter 13, “Configuring modules to provide quality of service” on page 441.

For detailed information on how to perform basic tasks in WebSphere Integration Developer, refer to Chapter 7., “WebSphere Integration Developer key concepts and common tasks” on page 153.

10.2 Preparing your environment

This section describes the configuration you must complete before following any of the development examples. This section assumes you are running an empty workspace in WebSphere Integration Developer V6.0.1

It contains the following sections:

- Importing the book ordering resources
- Enabling the Web Services Developer capability

- Checking the default host port and changing Web service endpoints

Importing the book ordering system resources

Most of the development examples in this redbook are based on a book ordering system. The resources required to build them have been included as Project Interchange files and will need to be imported into your workspace. These include:

- A shared library called BookOrderResources

  This contains all the required business objects and interfaces and is required for the majority of the samples. The project interchange file is BookOrderResources.zip.

- A Web service called BookOrderService

  This is a simple Web service that implements the BookOrderService interface. The project interchange file is BookOrderService.zip.

- A Web service called ProfileService

  This is a simple Web service that implements the ProfileService interface. The project interchange file is ProfileService.zip

These project interchange files are shipped with the additional material accompanying this redbook. See Appendix A, “Additional material” on page 473 for instructions on how to obtain this code.

From an empty WebSphere Integration Developer workspace use **File** -> **Import** -> **Project Interchange** to import each of these projects. All three zip files are located in the `\BookOrder` directory of the additional material.

Note: The Web service project interchange files contain two projects, an EAR and a Web project. Be sure to select both when importing into your workspace.

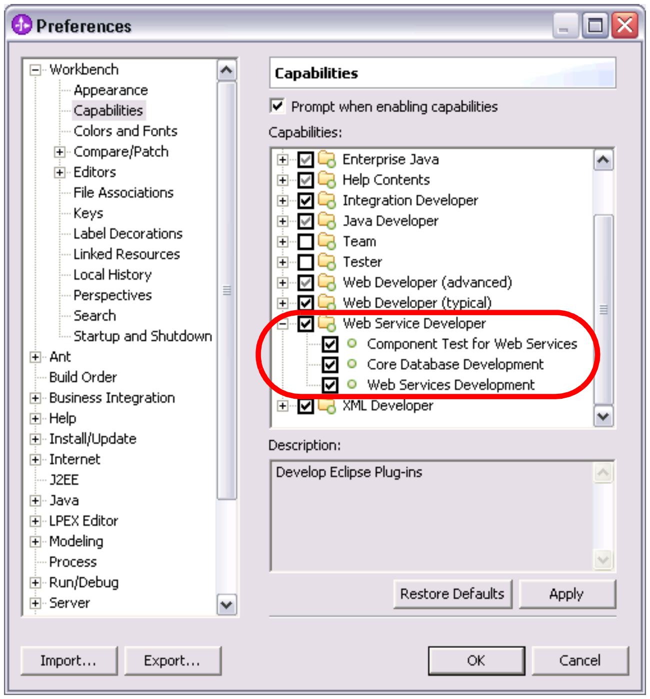
Also note once imported, the Web services projects will not be visible in the Business Integration view. To view them, right click in the Business Integration view and select **Show files**. The Web services projects will now be visible in the Physical Resources view.

Enabling the Web Service Developer capability

Some of the samples require you to use Web Service Developer functions of WebSphere Integration Developer. This is disabled by default so requires enabling. To enable it, perform the following

1. Click **Window -> Preferences**.

2. In the Preferences panel select **Workbench -> Capabilities**.
3. Expand the **Web Service Developer** role and check all of the check boxes (Figure 10-1 on page 274)



☑ Prompt when enabling capabilities

Capabilities:
- ☑ Enterprise Java
- ☑ Help Contents
- ☑ Integration Developer
- ☑ Java Developer
- ☐ Team
- ☐ Tester
- ☑ Web Developer (advanced)
- ☑ Web Developer (typical)
- ☑ Web Service Developer
  - ☑ Component Test for Web Services
  - ☑ Core Database Development
  - ☑ Web Services Development
- ☑ XML Developer

Description:

Develop Eclipse Plug-ins

Figure 10-1 Enabling Web Service Developer capability

4. Click **OK**. The Web Service Developer capabilities are now enabled.

Checking the default host port

The resources describing the Web service bindings in the BookOrderResources project use a default host port 9081. This is the default if you chose to install the WebSphere Enterprise Service Bus and WebSphere Process Server integrated test environments. If the default host port of your server is different then the SOAP address will need to be changed in the Web service definition.

To check which default host port you are using, perform the following:

1. Start the WebSphere Enterprise Service Bus integrated test environment server.
2. Right click the server and click **Run administrative console**.
3. Log in to the console.

4. Under Filter Administrative Tasks select **All** and click **Apply**. (Figure 10-2)

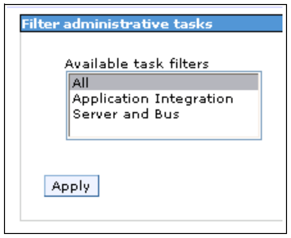
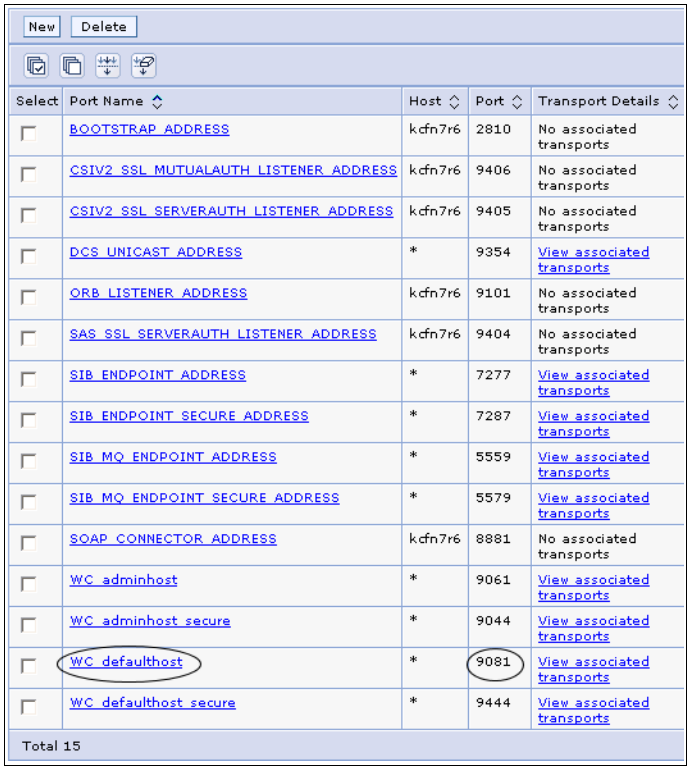


Figure 10-2 Filter administrative tasks

5. In the navigation expand **Servers** and click **Application servers**.
6. Click **server1**.
7. Under Communications click **Ports**.
8. A list of ports is displayed. Check the value of WC_defaulthost. (Figure 10-3 on page 275)



Select	Port Name	Host	Port	Transport Details
	BOOTSTRAP_ADDRESS	kcfn7r6	2810	No associated transports
	CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	kcfn7r6	9406	No associated transports
	CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	kcfn7r6	9405	No associated transports
	DCS_UNICAST_ADDRESS	*	9354	View associated transports
	ORB_LISTENER_ADDRESS	kcfn7r6	9101	No associated transports
	SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	kcfn7r6	9404	No associated transports
	SIB_ENDPOINT_ADDRESS	*	7277	View associated transports
	SIB_ENDPOINT_SECURE_ADDRESS	*	7287	View associated transports
	SIB_MQ_ENDPOINT_ADDRESS	*	5559	View associated transports
	SIB_MQ_ENDPOINT_SECURE_ADDRESS	*	5579	View associated transports
	SOAP_CONNECTOR_ADDRESS	kcfn7r6	8881	No associated transports
	WC_adminhost	*	9061	View associated transports
	WC_adminhost_secure	*	9044	View associated transports
	WC_defaulthost	*	9081	View associated transports
	WC_defaulthost_secure	*	9444	View associated transports

Total 15

Figure 10-3 Default host port number

If the port is not 9081, you will need to modify the port number used for the Web services in the BookOrderResources project. To do this, perform the following:

1. In the Business Integration view, expand the **BookOrderResources** project, expand **Web Service Ports** (Figure 10-4 on page 276).

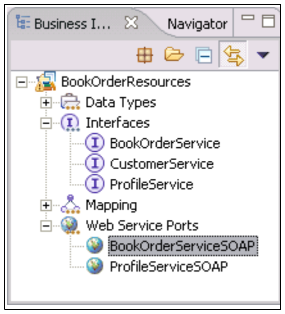


Figure 10-4 Web Service Ports

2. Right-click on **BookOrderServiceSOAP** and click **Open With -> WSDL Editor**.

> **Note:** The WSDL Editor will only be available if you turned on the Web Service Developer capability, as described in “Enabling the Web Service Developer capability” on page 273.

3. In the WSDL Editor under Services expand **BookOrderService** -> **BookOrderServiceSOAP** and select **soap:address** (Figure 10-5 on page 277).

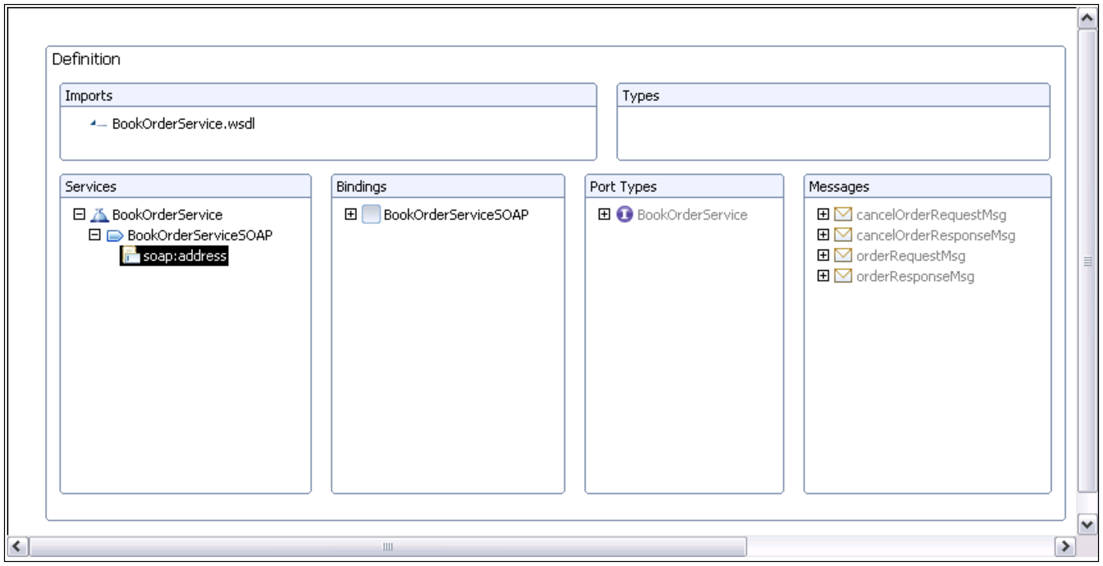
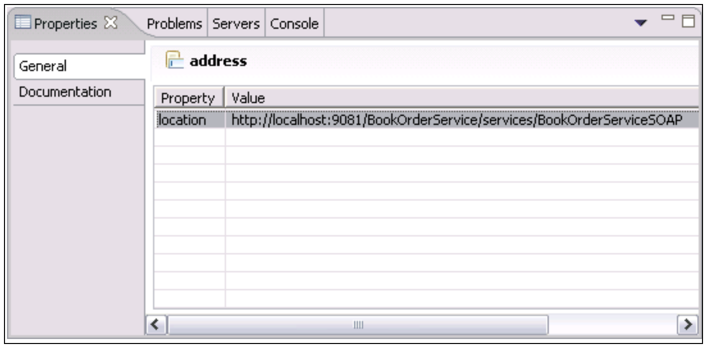


Figure 10-5 WSDL Editor

4. In the Properties view, notice the value the location property is set to (Figure 10-6):

   `http://localhost:9081/BookOrderService/services/BookOrderServiceSOAP`



Property	Value
location	http://localhost:9081/BookOrderService/services/BookOrderServiceSOAP

Figure 10-6 The location property

5. Modify the location property to use your server's default host port number. For example if the WC_defaulthost port is 9082, change the location property to:

   `http://localhost:`**`9082`**`/BookOrderService/services/BookOrderServiceSOAP.`

6. Save and close the WSDL file.
7. Repeat this process for the ProfileServiceSOAP Web Service Port.

11

Developing integration logic using mediation modules

This chapter provides step-by-step instructions on how to build basic integration logic in mediation modules for WebSphere Enterprise Service Bus. It contains the following three sections:

- 11.1, “Importing services” on page 281

  Describes how to import and use an existing Web service into a mediation module, link two mediation modules using an SCA binding, and how to bind to an Enterprise Information System such as CICS Transaction Server.
- 11.2, “Creating clients of mediation modules” on page 309

  Describes how to build clients that invoke mediation modules. Three clients are built: Web services, JMS, and SCA. Each client is built with a JSP front-end.
- 11.3, “Using services with mediation modules” on page 337

  Describes how to perform protocol transformation in a mediation module by mapping bindings, build a mediation flow to manipulate a request and response message, and how to handle faults.

These development examples assume you have configured your WebSphere Integration Developer workspace as described in Chapter 10, “Preparing for the development examples” on page 271.

You may find it useful to refer to Chapter 7, “WebSphere Integration Developer key concepts and common tasks” on page 153 for more detailed information on how to perform specific tasks in WebSphere Integration Developer.

11.1 Importing services

This section introduces how to import resources into mediation module, for use by an ESB. It discusses importing Web services, importing and exporting SCA modules, and importing an Enterprise Information System (EIS) such as CICS Transaction Service.

Each of the development examples in this section can be imported as Project Interchange files from the additional material supplied with this redbook in the `\ImportingServices\Solutions` directory.

11.1.1 Bindings

Both imports and exports require binding information, specifying the means of transport from and to the module. WebSphere Enterprise Service Bus supports the following bindings:

- SCA
- Web service
- EIS
- JMS
- EJB (import only)

We use many of this bindings in the development examples in this redbook, as described below.

Web service binding

The Web service binding is used in many of our samples, for example in 11.1.2, “Importing an existing Web service” on page 282 (import) and 11.3.1, “Mapping bindings” on page 338 (export).

JMS binding

The JMS binding is used in 11.3.1, “Mapping bindings” on page 338 (import) and in 11.2.2, “JMS client” on page 316 (export).

There is no example of a JMS custom binding in this book, but you will find information about it in the following article:

`http://www.ibm.com/developerworks/websphere/techjournal/0602_tost/0602_tost.html`

SCA binding

How to use SCA bindings is shown in 11.1.3, “Connect two modules using SCA binding” on page 286.

EJB binding

There is no sample of an import with an EJB binding in this book.

EIS binding

Look at 11.1.4, “EIS binding to CICS” on page 294 for an example on an import with an EIS binding.

11.1.2 Importing an existing Web service

This sample demonstrates how to invoke an existing Web service from a mediation module.

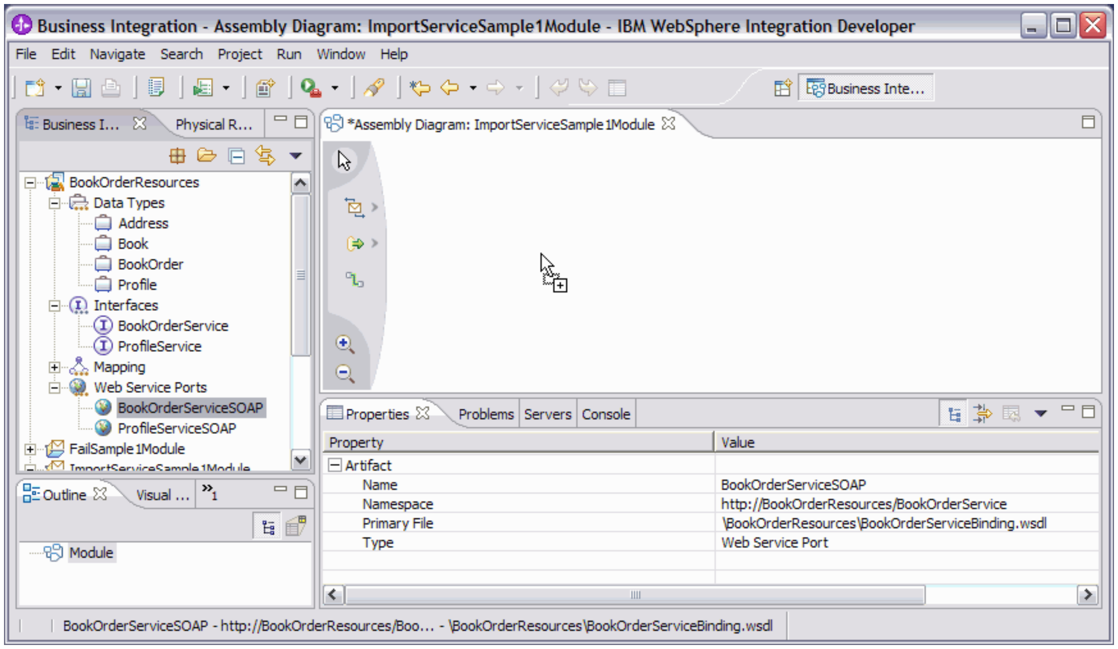
In many cases there will be existing Web services that you will want to make available to your ESB. Importing existing services makes them available to components that reside on the ESB.

This sample involves:

- Building a mediation module that imports an existing Web service.
- Using the Integration Test Client to send requests to the module.

The completed sample demonstrates that a book order request can be sent to a mediation module which forwards it to a Web service. The Web service returns a confirmation to the module that the book order was successful.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Uncheck the box to create a mediation flow component
   c. Set the Module Name to `ImportServiceSample1Module` and click **Next.**
   d. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the assembly diagram for the ImportServiceSample1Module module.
3. Expand the **BookOrderResources** library in the Business Integration view. Drag and drop the **BookOrderServiceSOAP** from the Business Integration view into the assembly diagram of the ImportServiceSample1Module module (Figure 11-1).

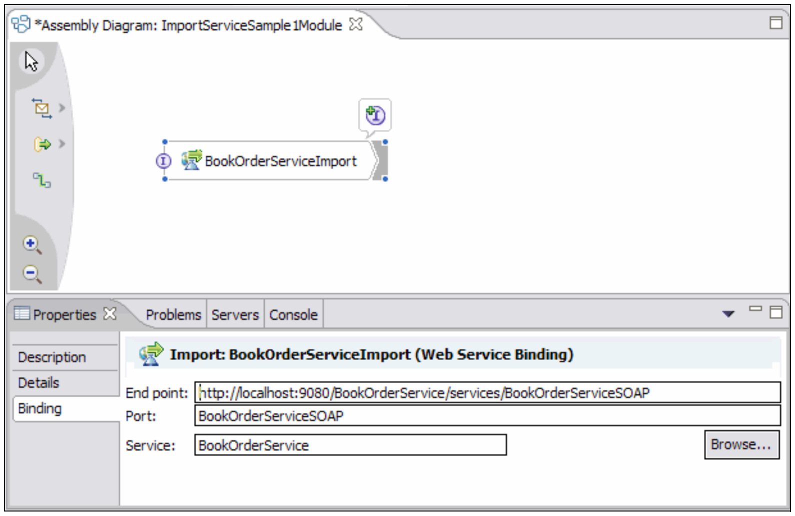


Property	Value
Artifact	
Name	BookOrderServiceSOAP
Namespace	http://BookOrderResources/BookOrderService
Primary File	\BookOrderResources\BookOrderServiceBinding.wsdl
Type	Web Service Port

BookOrderServiceSOAP - http://BookOrderResources/Boo... - \BookOrderResources\BookOrderServiceBinding.wsdl

Figure 11-1 Drag WSDL into the assembly diagram

4. This will open the Component Creation window. Select **Import with Web Service Binding** and click **OK**.
5. Rename the newly created import **Import1** in the assembly diagram to `BookOrderServiceImport`.
6. Select **BookOrderServiceImport**. In the Properties view and select the **Binding** tab. Review the settings, taken from the WSDL of the service (Figure 11-2).



Import: BookOrderServiceImport (Web Service Binding)

End point: http://localhost:9080/BookOrderService/services/BookOrderServiceSOAP

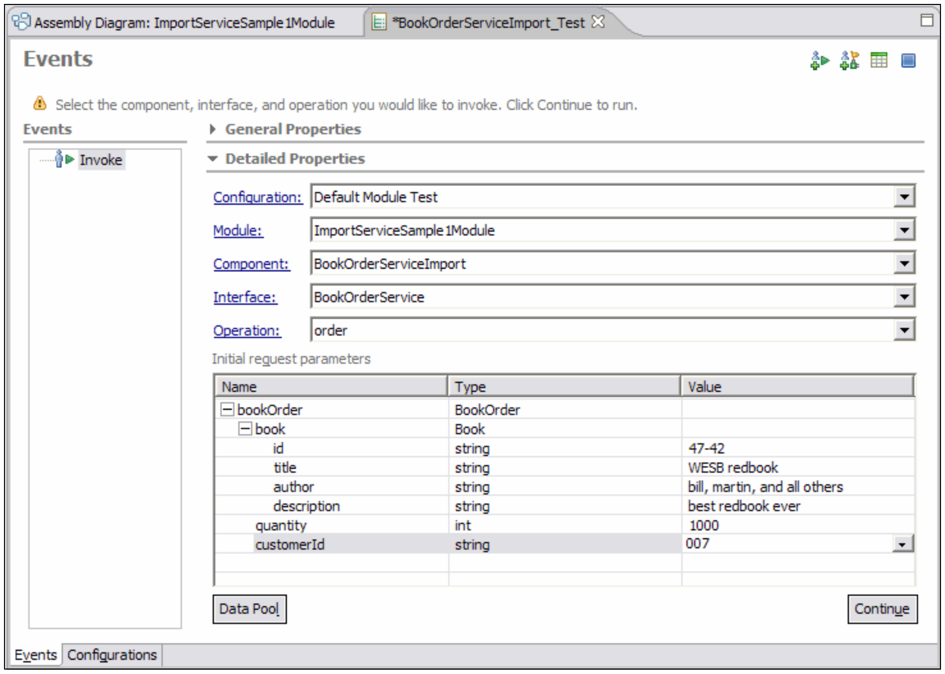
Port: BookOrderServiceSOAP

Service: BookOrderService

Figure 11-2 Import binding properties

Attention: You can overwrite the settings here, but it has no effect, as the content of the WSDL file, rather than the properties of the import is used at build time. So make sure the WSDL file of the service you want to use is configured correctly before you create the import.

7. Save the module.
8. Switch to the Servers view. Start your WebSphere Enterprise Service Bus server, if not already running.
9. Deploy the module and the Web service to the server.
   a. Switch to the Servers view.
   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
   c. Add ImportServiceSample1ModuleApp and BookOrderServiceEAR.
10. Click **Finish**.
11. Start the Integration Test Client by right-clicking the **BookOrderServiceImport** import in the assembly diagram and select **Test Component**. The BookOrderServiceImport_Test view opens.
12. Select the **order** operation and enter test data in the Initial request parameters section (Figure 11-3).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

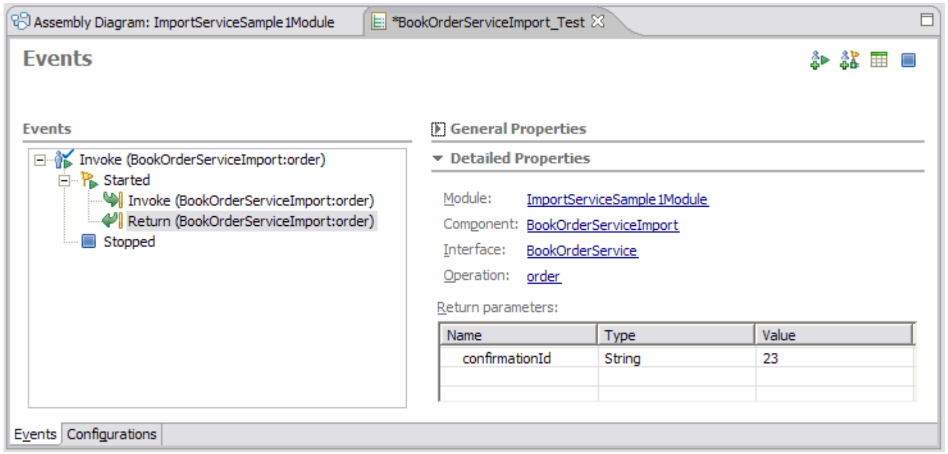
Configuration: Default Module Test
Module: ImportServiceSample1Module
Component: BookOrderServiceImport
Interface: BookOrderService
Operation: order

Initial request parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	47-42
title	string	WESB redbook
author	string	bill, martin, and all others
description	string	best redbook ever
quantity	int	1000
customerId	string	007

Figure 11-3 Test start settings

13. Click the **Continue** button. Select your WebSphere Enterprise Service Bus server as the deployment location. The test should finish returning a confirmationId (Figure 11-4).


Assembly Diagram: ImportServiceSample1Module
*BookOrderServiceImport_Test
Events
Invoke (BookOrderServiceImport:order)
Started
Invoke (BookOrderServiceImport:order)
Return (BookOrderServiceImport:order)
Stopped
General Properties
Detailed Properties
Events
Configurations

Module: ImportServiceSample1Module
Component: BookOrderServiceImport
Interface: BookOrderService
Operation: order

Return parameters:

Name	Type	Value
confirmationId	String	23

Figure 11-4 Test result

14. Congratulations! You successfully invoked an existing Web service from a mediation module.
15. Remove the projects from the test server.

Tip: Instead of dragging the WSDL file of the service into the assembly diagram, you can also create an import in the assembly diagram, add the interface, and then generate a Web service binding. After that you need to browse for the correct service binding WSDL file in the imports properties bindings section.

No matter which method is used to set up the Web service binding, the WSDL file must exist in the modules project or in a dependent library.

Typing in the bindings information manually is supported by the tooling, but it will show an error indicating the port attribute entered cannot be resolved. You can still deploy that module, but at runtime it will fail when trying to invoke the service.

11.1.3 Connect two modules using SCA binding

This sample demonstrates how to use SCA bindings to link SCA modules.

This sample involves:

- Creating a mediation module with an SCA Import.
- Creating a mediation module with an SCA Export and a Java component.
- Using the Integration Test Client to use one module to call the other.

The completed sample demonstrates how a mediation module can forward a book order request to another mediation module to handle.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Uncheck the box to create a mediation flow component.
   c. Set the Module Name to `SCABindingSample1Module` and click **Next.**
   d. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the assembly diagram editor for SCABindingSample1Module.
3. Add an import to the assembly diagram by using .
4. Rename Import1 to `SCABindingSample2ModuleImport`.

5. Right-click **SCABindingSample2ModuleImport** and select **Add Interface**.
6. Select the **BookOrderService** interface and click **OK**.
7. Right-click the import and click **Generate Binding** → **SCA Binding**.
8. Save the module.
9. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Uncheck the box to create a mediation flow component.
   c. Set the Module Name to `SCABindingSample2Module` and click **Next.**
   d. In the Select Required Libraries dialog tick the **BookOrderResources** library and click **Finish**.
10. Open the assembly diagram editor for SCABindingSample1Module.
11. Add an Export to the assembly diagram by using .
12. Rename Export1 to `SCABindingSample1ModuleExport`.
13. Right-click **SCABindingSample1ModuleExport** and select **Add Interface**.
14. Select the **BookOrderService** interface and click **OK**.
15. Right-click **SCABindingSample1ModuleExport** and select **Generate Binding -> SCA Binding**.
16. For testing purposes add a Java component to the assembly editor using .
17. Right-click on **Component1** and select **Add -> Interface**.
18. Select the **BookOrderService** interface and click **OK**.
19. Wire SCABindingSample1ModuleExport to Component1 (Figure 11-5 on page 287).

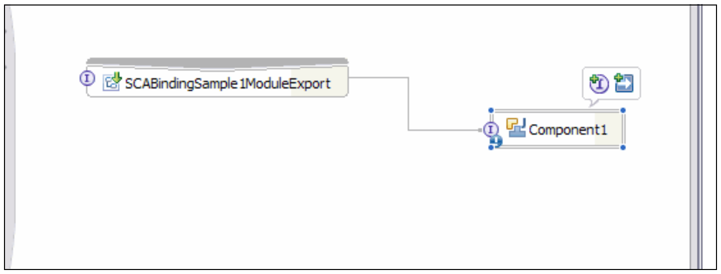
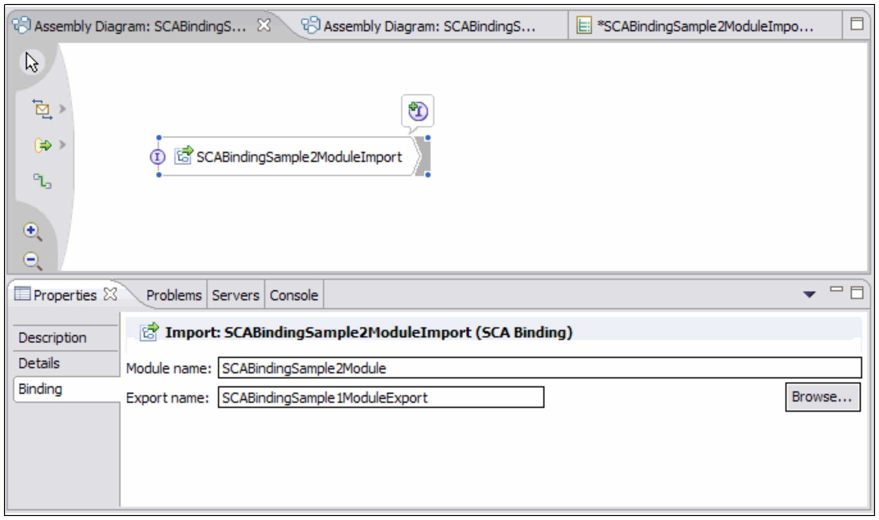


Figure 11-5 SCABindingSample2Module assembly diagram

20. Save the module.
21. Switch back to the SCABindingSample1Module assembly diagram.

22. Select the **SCABindingSample2ModuleImport** import and select the **Binding** tab in the Properties view.
23. Click **Browse** and select the **SCABindingSample1ModuleExport** in the Matches section of the SCA Export Selection window. Click **OK** (Figure 11-6 on page 288).


Assembly Diagram: SCABindingS...
Assembly Diagram: SCABindingS...
*SCABindingSample2ModuleImpo...
SCABindingSample2ModuleImport
Properties
Problems
Servers
Console
Description
Details
Binding
Browse...

Import: SCABindingSample2ModuleImport (SCA Binding)

Module name: SCABindingSample2Module

Export name: SCABindingSample1ModuleExport

Figure 11-6 SCA binding settings

24. Save the module.
25. Deploy both modules to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add SCABindingSample1ModuleApp and SCABindingSample2ModuleApp.
    d. Click **Finish**.
26. In the Business Integration view right-click the **SCABindingSample1Module** module and select **Test** → **Test Module**.
27. In the test view select the **Configurations** tab.
28. Click **Add**. Select **Module** and click **Next** (Figure 11-7).

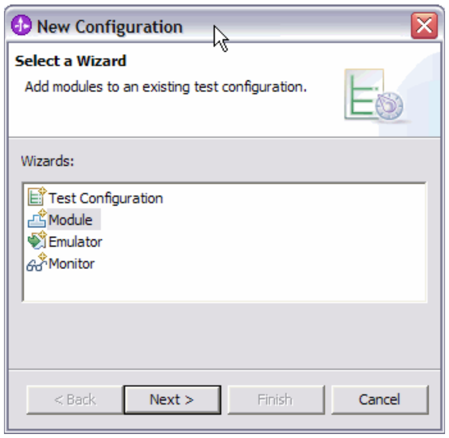


Figure 11-7 Add module to test

29. Select **Test Configuration Default Module Test** and click **Next** (Figure 11-8).

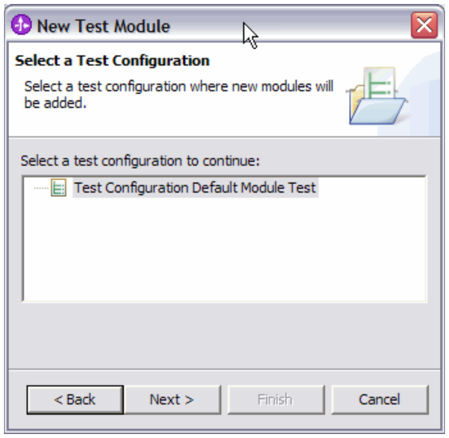

New Test Module
Select a Test Configuration
Select a test configuration where new modules will be added.
Select a test configuration to continue:
Test Configuration Default Module Test
< Back
Next >
Finish
Cancel

Figure 11-8 Select test configuration

30. Select the **SCABindingSample2Module** module and click **Finish** (Figure 11-9).

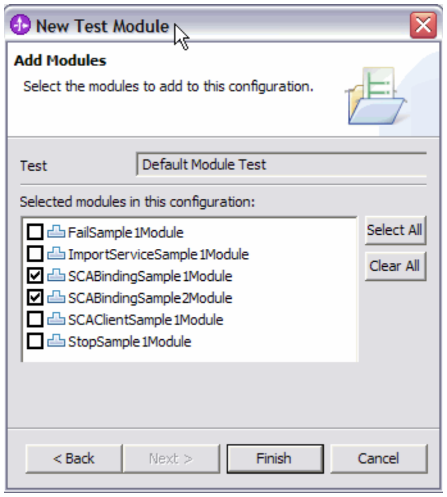
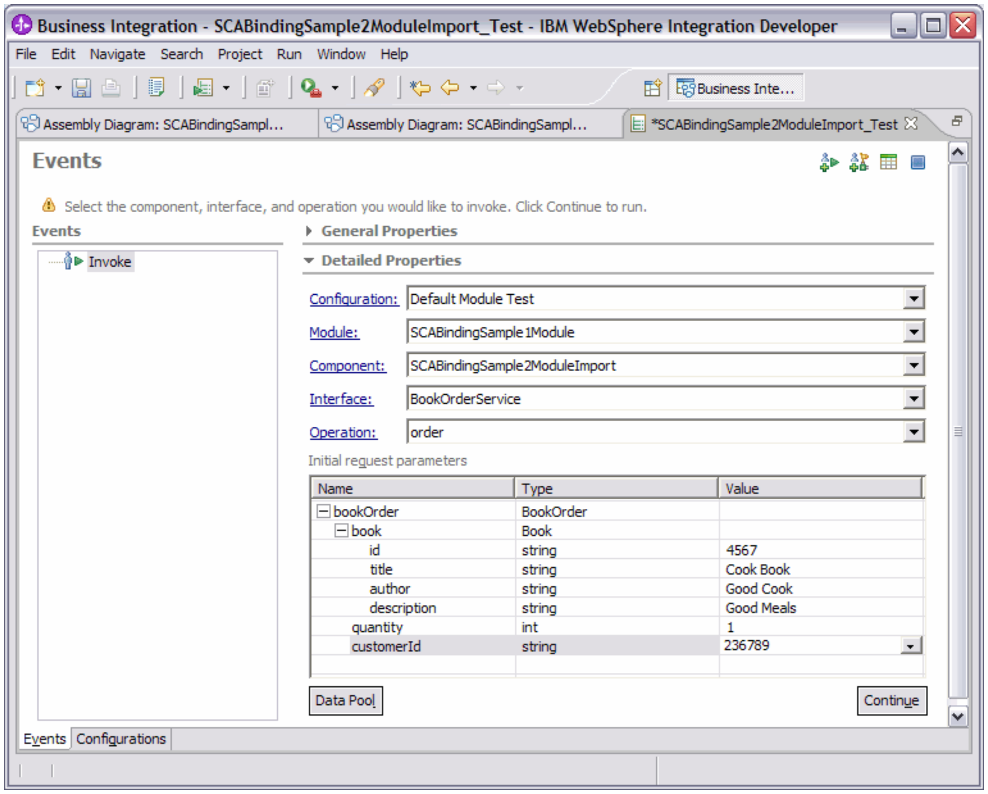

New Test Module
Add Modules
Select the modules to add to this configuration.
Test
Default Module Test
Selected modules in this configuration:
☐ FailSample1Module
☐ ImportServiceSample1Module
☑ SCABindingSample1Module
☑ SCABindingSample2Module
☐ SCAClientSample1Module
☐ StopSample1Module
Select All
Clear All
< Back
Next >
Finish
Cancel

Figure 11-9 Select module

31. Back in the test view, select the **Events** tab.
32. Select the **order** operation and enter test data (Figure 11-10 on page 291).


Business Integration - SCABindingSample2ModuleImport_Test - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Business Inte...
Assembly Diagram: SCABindingSampl...
Assembly Diagram: SCABindingSampl...
*SCABindingSample2ModuleImport_Test
Events
Select the component, interface, and operation you would like to invoke. Click Continue to run.
Events
Invoke
General Properties
Detailed Properties
Data Pool
Continue
Events Configurations

Configuration: Default Module Test
Module: SCABindingSample1Module
Component: SCABindingSample2ModuleImport
Interface: BookOrderService
Operation: order

Initial request parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	4567
title	string	Cook Book
author	string	Good Cook
description	string	Good Meals
quantity	int	1
customerId	string	236789

Figure 11-10 Test start settings

33. Click the **Continue** button.

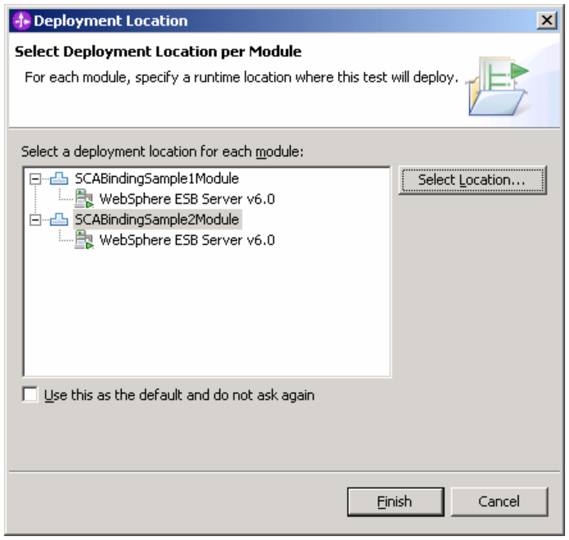
34. The Deployment Location dialog will open. We need to deploy each module to our WebSphere Enterprise Service Bus server (Figure 11-11 on page 292):

   a. Click on **SCABindingSample1Module** and click **Select Location**.

   b. In the dialog that opens, expand **WebSphere ESB Server v6.0**, click **WebSphere ESB Server v6.0** and click **Finish**.

   c. Repeat this process, this time selecting **SCABindingSample1Module** and selecting the location to again be WebSphere ESB Server v6.0.

   d. Click **Finish** to start the deployment.


Deployment Location

Select Deployment Location per Module

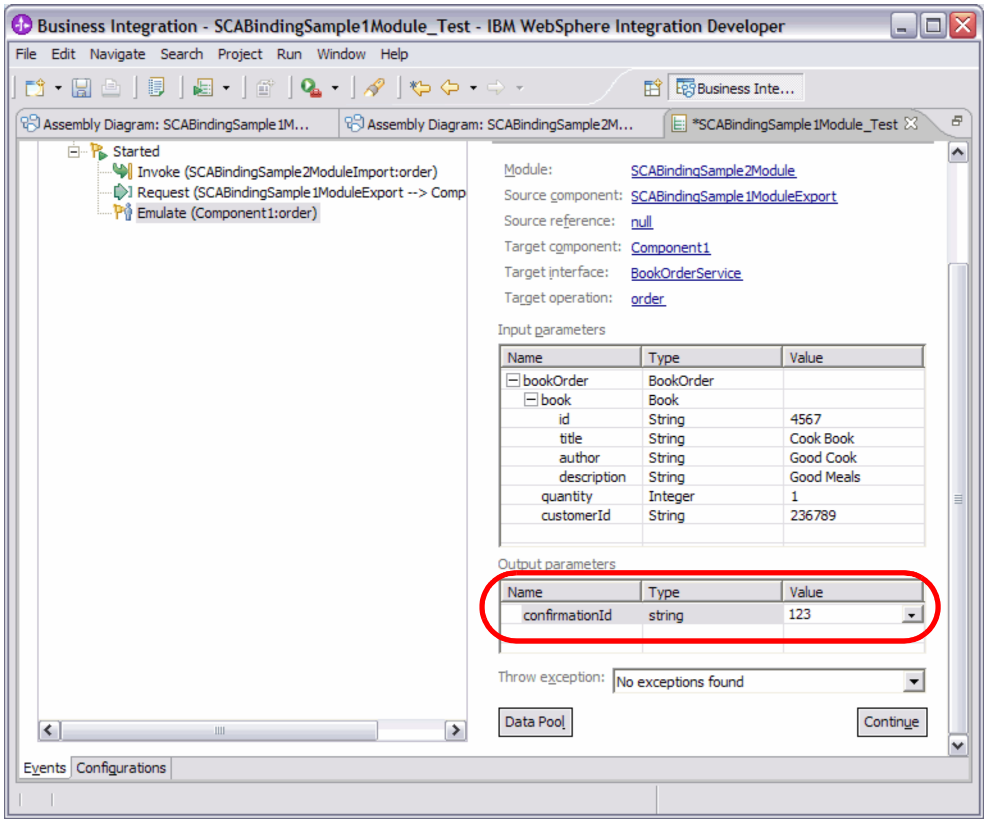
For each module, specify a runtime location where this test will deploy.

Select a deployment location for each module:

- SCABindingSample1Module
  - WebSphere ESB Server v6.0
- SCABindingSample2Module
  - WebSphere ESB Server v6.0

Figure 11-11 Selecting WebSphere Enterprise Service Bus as the deployment location

35. The test stops at the emulation of the Java component. Enter any string value in the confirmationId output parameter and click **Continue** (Figure 11-12 on page 293).


Business Integration - SCABindingSample1Module_Test - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Business Inte...
Assembly Diagram: SCABindingSample1M...
Assembly Diagram: SCABindingSample2M...
*SCABindingSample1Module_Test
Started
Invoke (SCABindingSample2ModuleImport:order)
Request (SCABindingSample1ModuleExport --> Comp
Emulate (Component1:order)
Data Pool
Continue
Events
Configurations

Module: SCABindingSample2Module
Source component: SCABindingSample1ModuleExport
Source reference: null
Target component: Component1
Target interface: BookOrderService
Target operation: order

Input parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	String	4567
title	String	Cook Book
author	String	Good Cook
description	String	Good Meals
quantity	Integer	1
customerId	String	236789

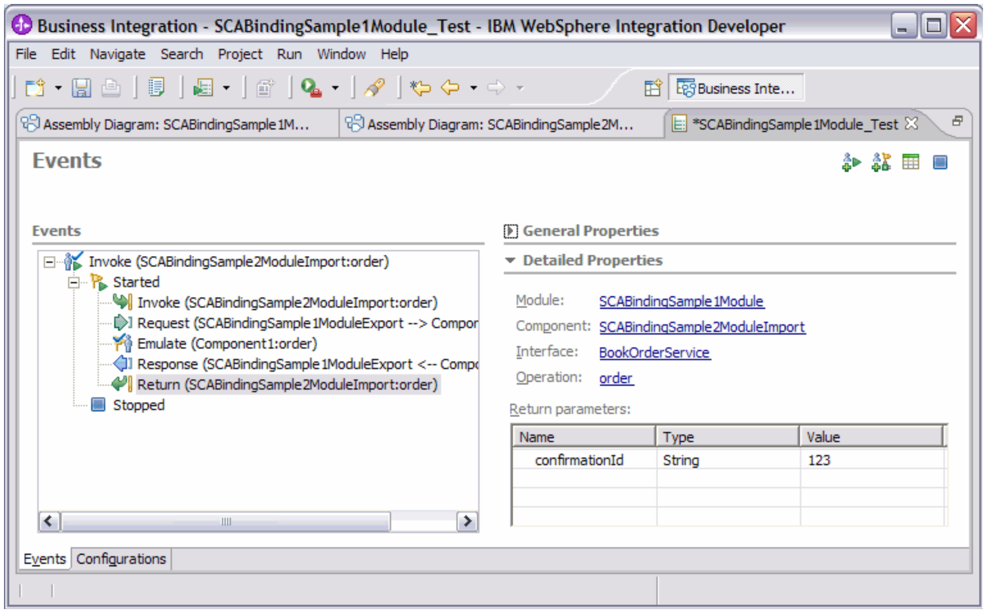
Output parameters

Name	Type	Value
confirmationId	string	123

Throw exception: No exceptions found

Figure 11-12 Component emulation

36. The test view should show the confirmationId value you entered as the test return parameter (Figure 11-13 on page 294).


Business Integration - SCABindingSample1Module_Test - IBM WebSphere Integration Developer
File Edit Navigate Search Project Run Window Help
Business Inte...
Assembly Diagram: SCABindingSample1M...
Assembly Diagram: SCABindingSample2M...
*SCABindingSample1Module_Test
Events
Events
General Properties
Detailed Properties
Events Configurations

Invoke (SCABindingSample2ModuleImport:order)
Started
Invoke (SCABindingSample2ModuleImport:order)
Request (SCABindingSample1ModuleExport --> Compor
Emulate (Component1:order)
Response (SCABindingSample1ModuleExport <-- Compo
Return (SCABindingSample2ModuleImport:order)
Stopped

Module: SCABindingSample1Module
Component: SCABindingSample2ModuleImport
Interface: BookOrderService
Operation: order

Return parameters:

Name	Type	Value
confirmationId	String	123

Figure 11-13 Test result

Congratulations you have successfully demonstrated how to use SCA bindings to allow mediation modules to communicate. This demonstrates that the request entered for the import in SCABindingSample1Module was passed to the export in SCABindingSample2Module and that the response was sent back.

37. Remove all projects from the test server.

11.1.4 EIS binding to CICS

This sample illustrates how to use WebSphere Integration Developer to develop a mediation module which contains a component that accesses a CICS commarea transaction.

This sample involves:

- Creating a resource adapter.
- Creating a mediation module with an EIS binding to a J2EE Connector Architecture (J2C) connector using the Enterprise Service Discovery wizard.
- Using the Web Services Explorer to test the module using a Web service request.

The completed sample demonstrates how a mediation module can use a Web service request to access a J2C interface to CICS.

For our sample, we will use the EC01 transaction which ships as a sample with the CICS Transaction Gateway. The transaction expects and returns a commarea that is defined by the following copybook, which we will use during the generation of the component.

```
01 DFHCOMMAREA.
            02 ECIDATE     PIC  X(8).
            02 FILLER      PIC  X(1).
            02 ECITIME     PIC  X(8).
            02 FILLER      PIC  X(3).
```

1. Start your WebSphere Enterprise Service Bus server.
2. Right-click on the server and select **Run administrative console** and log in.
3. Click **Resources** -> **Resource Adapters** and the **Install RAR** button.
4. Click **Browse**, locate your `<WID_INSTALL>/Resource Adapters/cics15` directory, select the **cicseci601.rar** file, click **Open**, click **Next** and click **OK.**
5. A new resource adapter named ECIResourceAdapter will now appear in the list. Click it.
6. Select **J2C Connection Factories** under Additional Properties.
7. Click the **New** button.
8. Create a new J2C Connection Factory.
   a. Set the Name to `CICS01`.
   b. Set JNDI Name to `eis/CICS`.
   c. Click **OK.**
9. Click the new connection factory **CICS01**.
10. Under Additional Properties select **Custom Properties.** You will have to set a minimum of 3 properties:
    a. Select **ConnectionURL** and set the value to the DNS name or IP address of the system running the CICS Transaction Gateway V6. Click **OK**.
    b. Select **PortNumber** and set the value to port number CICS Transaction Gateway is listening on (default is 2006). Click **OK**.
    c. Select **ServerName** and set the value to the name configured in the CTG for the particular CICS address space that is running the transaction you want to invoke. Click **OK**.

Note: If you do not have the CICS Transaction Gateway configured, you can still continue through this sample to gain an understanding of the function of the wizard and the generated components. You can enter any values you like above, or leave them blank all together. The resultant Import component will, off course, fail if you attempt to test it.

11. Click the **Save** link and confirm the changes by clicking the **Save** button.
12. Log out the Administrative console and restart the test server.

With the J2C Connection Factory configured in the server, we are ready to start building the mediation module.

13. Create a new mediation module.
    a. In the Business Integration view, right-click and select **New -> Mediation Module**.
    b. Set the Module Name to `CICSSample1Module` and click **Finish.**

Invoke the Enterprise Service Discovery wizard.

14. Right-click the new module **CICSSample1Module** and select **New** -> **Other.**
15. Select **Business Integration -> Enterprise Service Discovery** (Figure 11-14).
16. Click **Next**.

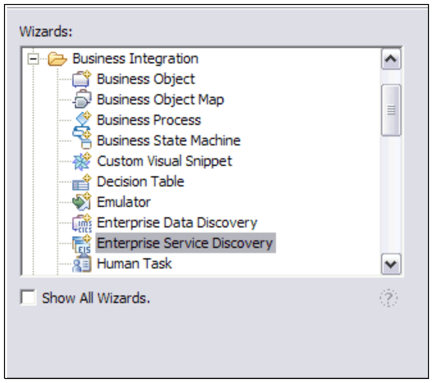


Figure 11-14 Selecting the Enterprise Service Discovery wizard

17. When the Enterprise Service Discovery wizard opens, click the **Import Resource Adapter button** (Figure 11-15). We have done this for the test server but now must do it for the development environment.

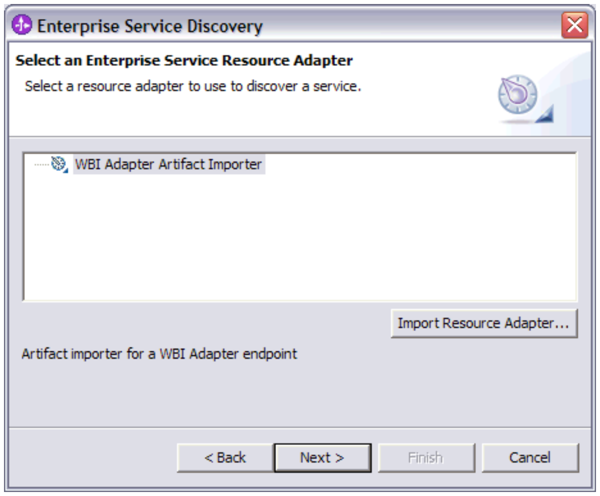
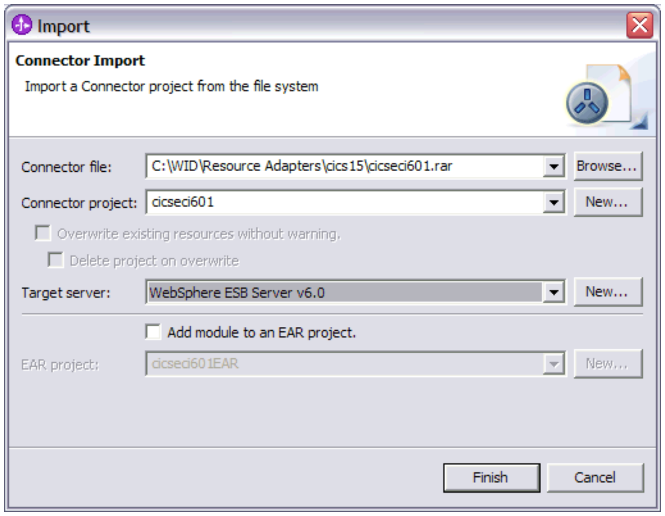

Enterprise Service Discovery
Select an Enterprise Service Resource Adapter
Select a resource adapter to use to discover a service.
WBI Adapter Artifact Importer
Import Resource Adapter...
Artifact importer for a WBI Adapter endpoint
< Back
Next >
Finish
Cancel

Figure 11-15 Importing the ECI Resource Adapter

18. Browse to your `<WID_INSTALL>/Resource Adapters/cics15` directory and select the **cicseci601.rar** file.
19. Ensure the Target server is set to your WebSphere Enterprise Service Bus server and click **Finish** (Figure 11-16).

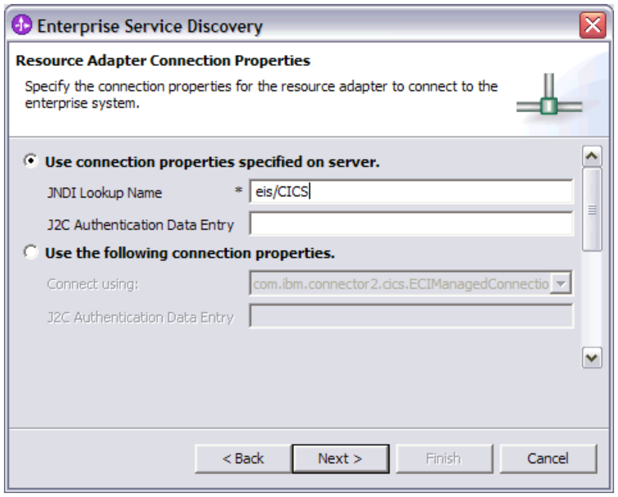


Connector file: C:\WID\Resource Adapters\cics15\cicseci601.rar
Connector project: cicseci601
Overwrite existing resources without warning.
Delete project on overwrite
Target server: WebSphere ESB Server v6.0
Add module to an EAR project.
EAR project: cicseci601EAR

Figure 11-16 Importing the connector project

20. When asked if you want to switch to the J2EE perspective, click **No**.
21. Now with the **ECIResourceAdapter** selected in the wizard, click **Next**.

22. Set the JNDI Lookup Name to `eis/CICS` and click **Next** (Figure 11-17).



Resource Adapter Connection Properties

Specify the connection properties for the resource adapter to connect to the enterprise system.

(●) Use connection properties specified on server.

JNDI Lookup Name * eis/CICS

J2C Authentication Data Entry

( ) Use the following connection properties.

Connect using: com.ibm.connector2.cics.ECIManagedConnectio

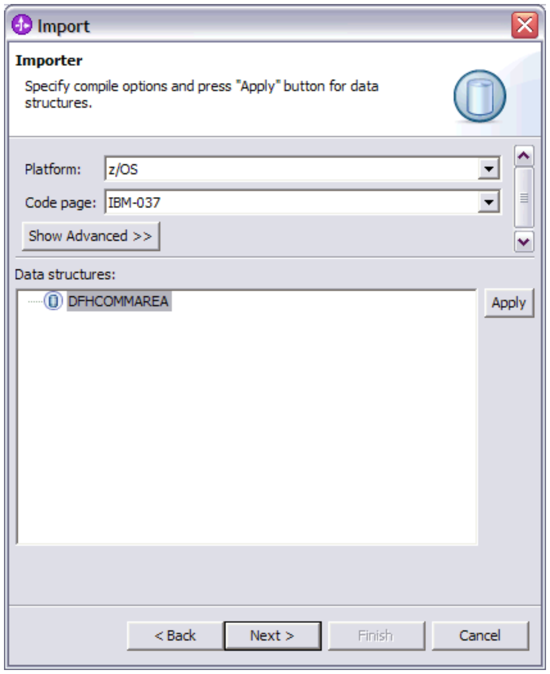
J2C Authentication Data Entry

Figure 11-17 Setting the JNDI Lookup Name of the connection factory

23. In the Add Operations page, click the **Add** button.
24. Set the Operation Name to `getTimeStamp`.
25. Set the Input type by pressing the **New** button and browsing to ecidate.cpy (the name of the copybook on your local system that defines the commarea layout for the transaction). Click **Next**.

> **Note:** The ecidate.cpy file is delivered with the additional material supplied with this redbook. You can find it in the directory:
>
> ```
> \ImportingServices\Resources\CICSImport
> ```

26. Change the Platform to **z/OS** and click **Apply**. This causes the wizard to read the copybook file and display the 01 level data structures it finds. In our case there is only one.
27. Select **DFHCOMMAREA** in the list and click **Next** (Figure 11-18 on page 299).


Import

Importer

Specify compile options and press "Apply" button for data structures.

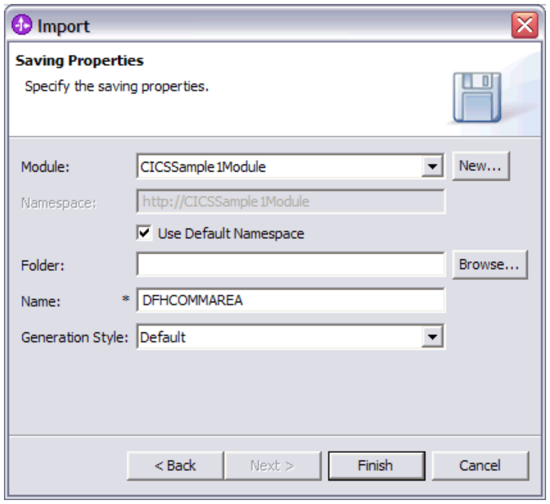
Platform: z/OS

Code page: IBM-037

Data structures: DFHCOMMAREA

Figure 11-18 Importer compile options

28. The Properties for the Import will look like Figure 11-19. Click **Finish**.


Import

Saving Properties

Specify the saving properties.

Module: CICSSample1Module

Namespace: http://CICSSample1Module

☑ Use Default Namespace

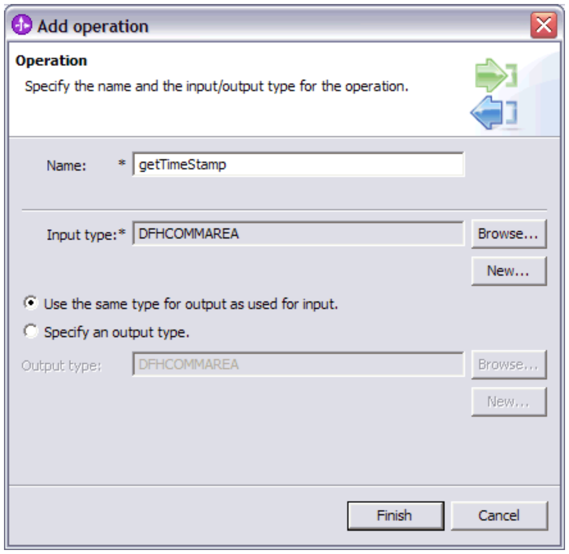
Folder:

Name: * DFHCOMMAREA

Generation Style: Default

Figure 11-19 Saving the generated data type

29. Back in the Add Operation page, select the radio button **Use the same type for output as input**.
30. Click **Finish** (Figure 11-20).


Add operation

Operation

Specify the name and the input/output type for the operation.

Name: * getTimeStamp

Input type:* DFHCOMMAREA

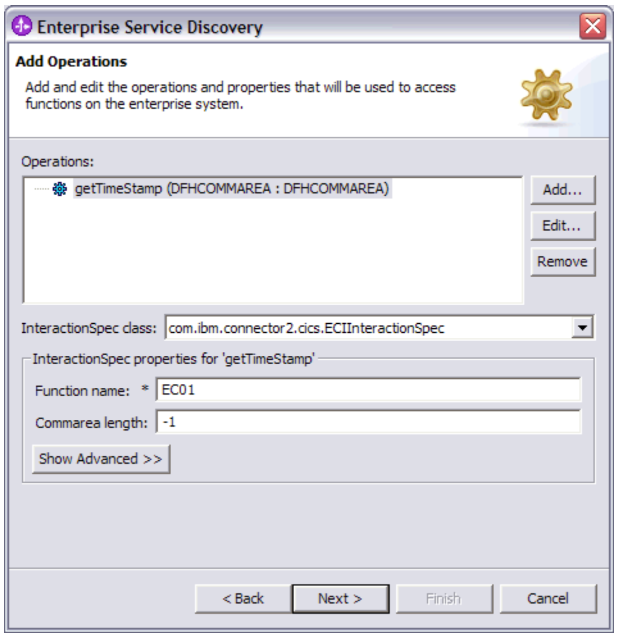
(●) Use the same type for output as used for input.

( ) Specify an output type.

Output type: DFHCOMMAREA

Figure 11-20 Create a new operation

31. In the next Add Operations page, we need to enter the name of the CICS program that we want to invoke. In our system, set Function name to `EC01` and click **Next** (Figure 11-21 on page 301).


Enterprise Service Discovery

Add Operations

Add and edit the operations and properties that will be used to access functions on the enterprise system.

Operations:

getTimeStamp (DFHCOMMAREA : DFHCOMMAREA)

InteractionSpec class: com.ibm.connector2.cics.ECIInteractionSpec

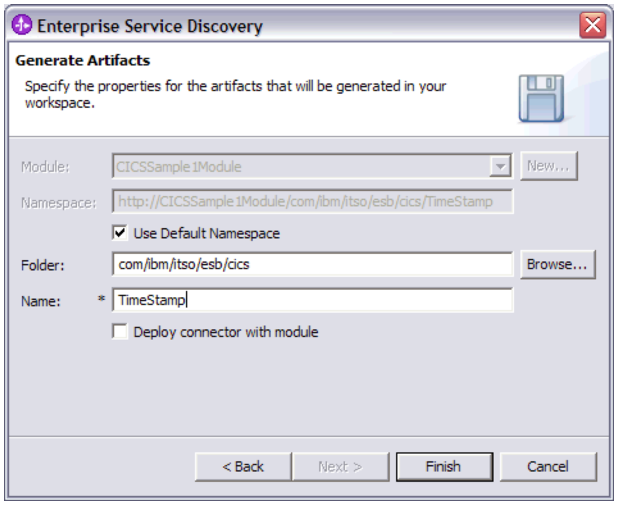
InteractionSpec properties for 'getTimeStamp'

Function name: * EC01

Commarea length: -1

Figure 11-21 Specify the CICS program name

32. On the Generate Artifacts dialog, create a new folder named `com/ibm/itso/esb/cics`
33. Set the name of the artifact to `TimeStamp`.
34. Click **Finish** (Figure 11-22).



Specify the properties for the artifacts that will be generated in your workspace.

Module: CICSSample1Module

Namespace: http://CICSSample1Module/com/ibm/itso/esb/cics/TimeStamp

☑ Use Default Namespace

Folder: com/ibm/itso/esb/cics

Name: * TimeStamp

☐ Deploy connector with module

Figure 11-22 Saving the generated artifacts

35. Examine the contents of the mediation module after the wizard finishes. Notice a new data type and interface were created (Figure 11-23).

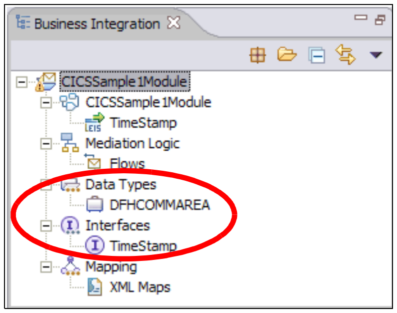


Figure 11-23 Looking at what was generated

36. Open the assembly diagram editor by double-clicking . for CICSSample1Module.

37. Click on the **TimeStamp** import and in the Properties view. Notice this component is an EIS Binding import. Click the **Binding** tab and notice the JNDI Name. This is how the component will find the connection factory to obtain its connection to the CICS Transaction Gateway and CICS.

38. To test the TimeStamp component, right-click on **Timestamp** and select **Test Component**. This opens the Integration Test Client.

39. Select **getTimeStamp** from the Operation drop down menu.

40. Click the **Continue** button.

41. If your CICS Transaction Gateway is listening, the response will be the current date and time.

WebSphere Integration Developer makes it very easy to build and test an SCA-enabled component that wraps the JCA interface to CICS. So, why did we build this component and what can WebSphere Enterprise Service Bus do with it? The rest of this sample illustrates one option, and hopefully will help you to understand the power of using the tooling to generate mediation components that access IMS™ and CICS back end systems.

Let's assume we wanted to make this component accessible as a Web service. We could do the following:

42. In the Business Integration view, expand CICSSample1Module, right-click on **Data Types** and select **New -> Business Object**.

43. Set the Name field to `TimeStampBO` and click **Finish**.

44. Add two attributes to the business object (Figure 11-24).

    a. Click the Add Attribute button and name it `date`.

    b. Click the Add Attribute button again and name it `time`.

    c. Save the business object, and close the business object editor.

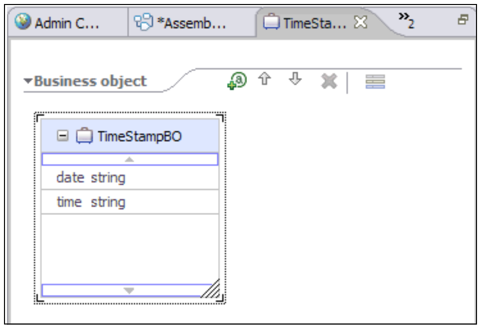


Figure 11-24 The business object

45. In the Business Integration view, expand CICSSample1Module, right click on **Interfaces** and select **New -> Interface**.

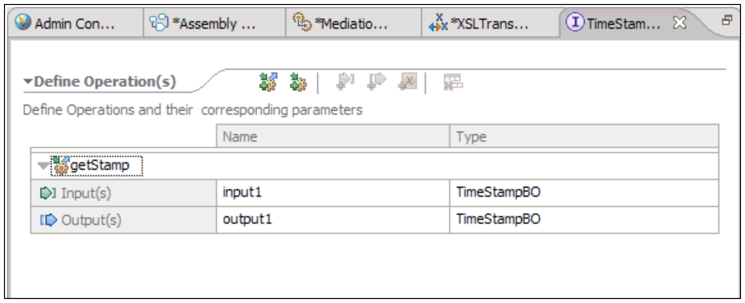
46. Set the name of the interface to `TimeStamp` and click **Finish**.

47. Define the interface (Figure 11-25 on page 304).

    a. Add a new two way operation by clicking the Add Request Response Operation button . Name it `getStamp`.

b. Add an input by clicking the Add Input button . and set the Type to TimeStampBO.

c. Add an output by click the Add Output button .and set the Type to TimeStampBO.



	Name	Type
getStamp		
Input(s)	input1	TimeStampBO
Output(s)	output1	TimeStampBO

Figure 11-25 The TimeStamp interface

48. Save and close the interface

49. Add an export to the CICSSample1Module assembly diagram by using .

50. Rename Export1 to `TimeStampServiceExport`.

51. Rename Mediation1 to `CICSMediation`.

52. Right-click **TimeStampServiceExport** and select **Add Interface**.

53. Select the **TimeStamp** interface and click **OK**.

54. Wire the components together: TimeStampServiceExport to CICSMediation to TimeStamp (Figure 11-26).

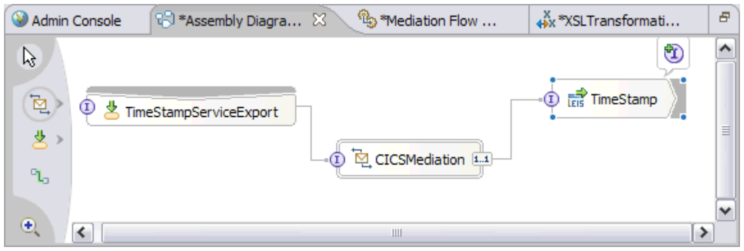


Figure 11-26 Using the import in the assembly diagram

55. Right click on **TimeStampServiceExport** and select **Generate Binding -> Web Service Binding**.

56. In the Binding File Generation dialog select Yes, and in the Transport Selection dialog select **soap/http** and click **OK**.

Note: The soap address generated in the binding WSDL uses a default port of 9080. If you WebSphere Enterprise Service Bus server uses a default host port of 9081 (or any other value) you will either need to modify the WSDL file to reflect this, or use the TCP/IP Monitor.

57. Double-click on **CICSMediation** to generate an implementation for the mediation flow component. In the Open dialog click **Yes**, then in the Generate Implementation dialog click **OK**.
58. In the Operation connections panel, wire the **getStamp** operation on the TimeStampService interface to the **getTimeStamp** operation on the TimeStampPartner reference.
59. Add an XSL Transform mediation primitive to the request flow.
60. Wire the request flow (Figure 11-27).
    a. Wire TimeStampService_getStamp_Input to the in terminal of XSLTransformation1.
    b. Wire the out terminal of XSLTransformation1 to TimeStampPartner_getTimeStamp_Callout.

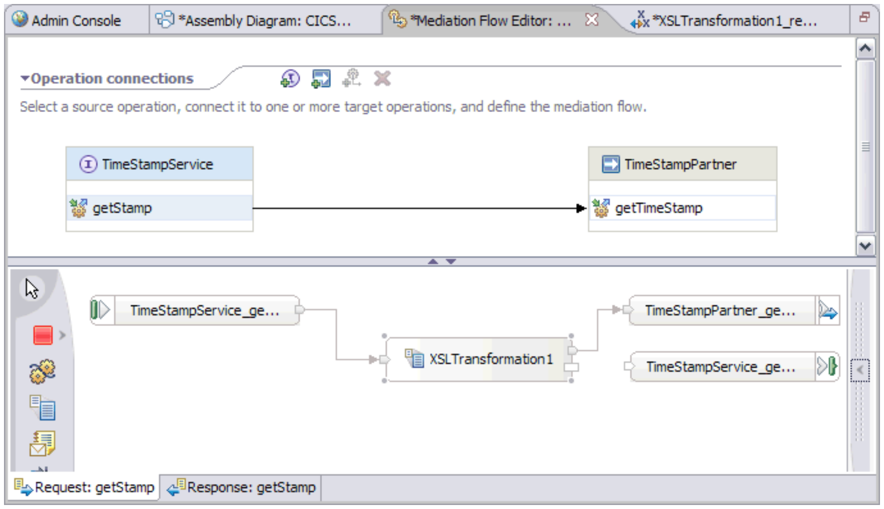


Figure 11-27 A sample mediation request flow

61. Click **XSLTransform1** and in the Properties view select the **Details** tab.
62. Click the **New** button to create a new mapping and in the New XSLT Mapping dialog click **Finish** to accept the default message types.
63. Create the mapping (Figure 11-28 on page 306).

a. In the source panel select **body -> getStamp -> input1 -> date**.

b. In the target panel select **body -> getTimeStamp -> tns_1:getTimeStampInput -> xsd1:ecidate**.

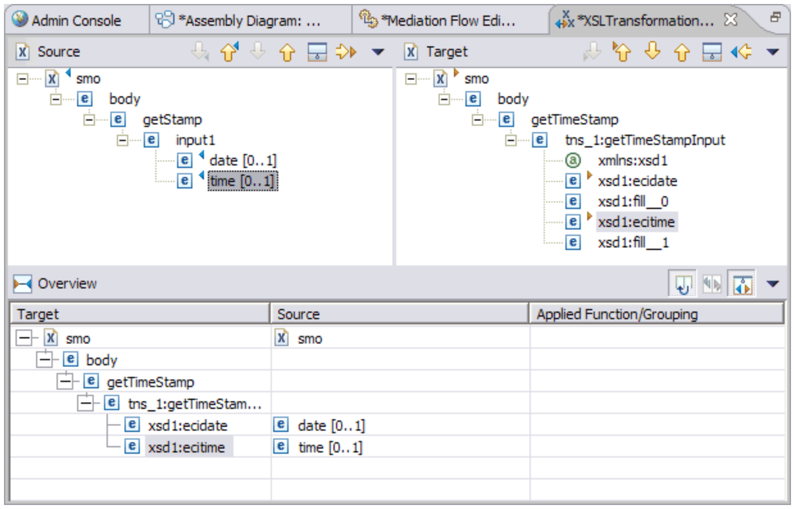
c. In the source panel right-click on **date** and select **Create Mapping**.

d. In the source panel select **body -> getStamp -> input1 -> time**.

e. In the target panel select **body -> getTimeStamp -> tns_1:getTimeStampInput -> xsd1:ecitime**.

f. In the source panel right-click on **time** and select **Create Mapping**.

g. Save the mapping file and close it.



Target	Source	Applied Function/Grouping
smo	smo	
body		
getTimeStamp		
tns_1:getTimeStam...		
xsd1:ecidate	date [0..1]	
xsd1:ecitime	time [0..1]	

Figure 11-28 A simple XSLT transformation

64. Click the **Regenerate XSL** button.

65. In the mediation flow editor click on the **Response** tab.

66. Add an XSL Transform mediation primitive to the response flow.

67. Wire the response flow (Figure 11-29 on page 307).

a. Wire TimeStampServicePartner_getTimeStamp_CalloutResponse to the in terminal of XSLTransformation1.

b. Wire the out terminal of XSLTransformation1 to TimeStamp_getStamp_InputReponse.

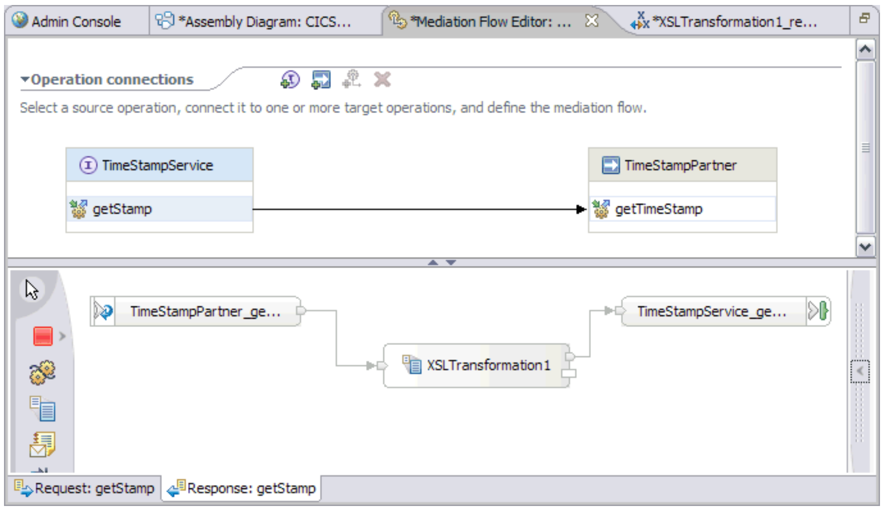


Figure 11-29 A sample mediation response flow

68.Click **XSLTransform1** and in the Properties view select the **Details** tab.

69.Click the **New** button to create a new mapping and in the New XSLT Mapping dialog click **Finish** to accept the default message types.

70.Create the mapping.

a. In the source panel select **body -> getTimeStampResponse -> tns_1:getTimeStampOutput -> xsd1:ecidate**.

b. In the target panel select **body -> getStampResponse -> output1 -> date**.

c. In the source panel right-click on **xsd1:ecidate** and select **Create Mapping**.

a. In the source panel select **body -> getTimeStampResponse -> tns_1:getTimeStampOutput -> xsd1:ecitime**.

b. In the target panel select **body -> getStampResponse -> output1 -> time**.

c. In the source panel right-click on **xsd1:ecitime** and select **Create Mapping**.

d. Save the mapping file and close it.

71.Click the **Regenerate XSL** button.

72.Save the mediation flow and the mediation module.

73.Deploy the module to the server.

a. Switch to the Servers view.

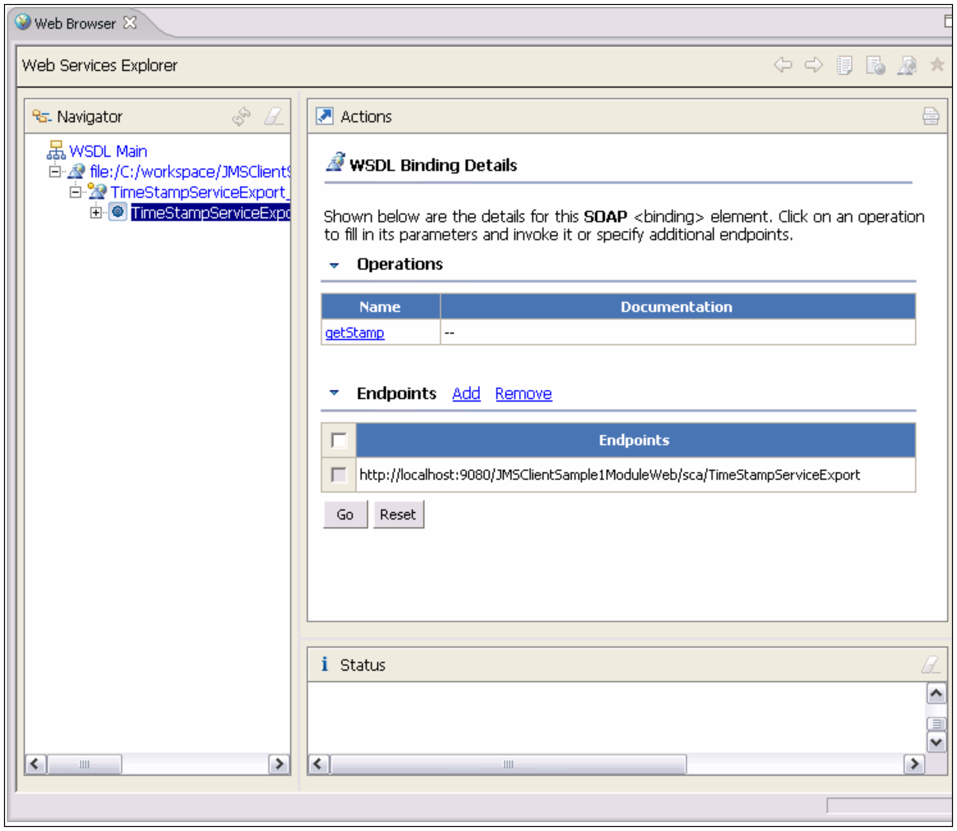
b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.

c. Add CICSSample1Module if it is not already added.

d. Click **Finish**.

74.Open the Physical Resources view by right-clicking on the **CICSSample1Module** module and selecting **Show Files**.

75.In the CICSSample1Module project right-click on **TimeStampServiceExport_TimeStampServiceHttp_Service.wsdl** and select **Web Services -> Test with Web Services Explorer** (Figure 77).



Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Name	Documentation
getStamp	--

	Endpoints
☐	http://localhost:9080/JMSClientSample1ModuleWeb/sca/TimeStampServiceExport

Figure 11-30 Web Services Explorer

76.In the Web Services Explorer, click on the **getStamp** operation.

77.Next to Date, click **Add** and add the date `01-01-2006`.

78.Next to Time, click Add and add the time `00:01`.

79. Click the **Go** button and check the returned value which should come from CICS.

80. Remove the project from the server.

Congratulations, you have successfully built and tested a module that takes in a Web service request and calls a J2C connector.

11.2 Creating clients of mediation modules

This section describes how to build clients that invoke mediation modules. Three clients are built:

- Web services client
- JMS client
- SCA client

Each client is invoked from a custom JSP page.

Each of the development examples in this section can be imported as Project Interchange files from the additional material supplied with this redbook in the `\Clients\Solutions` directory.

11.2.1 Web services client

This example demonstrates how to create a Web services client with a JSP front end, capable of communicating with a mediation module.

A Web services client uses Java to locate a service and invoke it.

This sample involves:

- Modifying a mediation module to accept Web service requests.
- Generating a Java Web service client.
- Creating a JSP page to invoke the Web service client.
- Using the generated client to invoke a Web service.

The completed sample demonstrates a Web services client making a request to create a new customer profile. The mediation module converts this request and invokes a Web service that creates a profile. The Web services client receives confirmation that a customer profile has been created.

1. Import the XSL Transformation mediation primitive development example into your workspace.

a. Click **File** -> **Import** -> **Project Interchange** and click **Next**.

b. Click **Browse** next to From zip file, and select XSLTransformation.zip which is located in the `\MediationPrimitives\Solutions` directory in the additional material supplied with this redbook.

c. Select **XSLTSample1Module** and click **Finish**.

2. Open XSLTSample1Module in the Assembly Editor using (Figure 11-31).

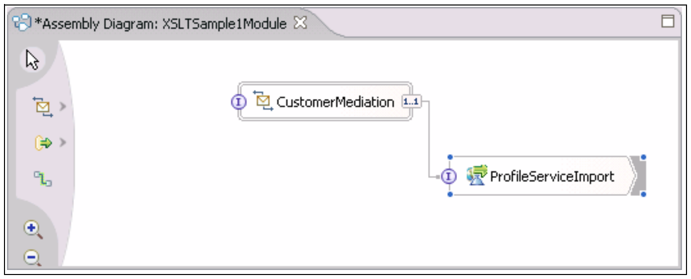


Figure 11-31 Mediation module from XSLTransform sample

3. Add an Export to the assembly diagram by using and rename it `CSExport`.

Note: We cannot name the Export CustomerServiceExport because the file name of the generated WSDL becomes too large and causes a URI is greater than the Windows limit exception during deploy.

4. Wire CSExport to CustomerMediation (Figure 11-32 on page 310).

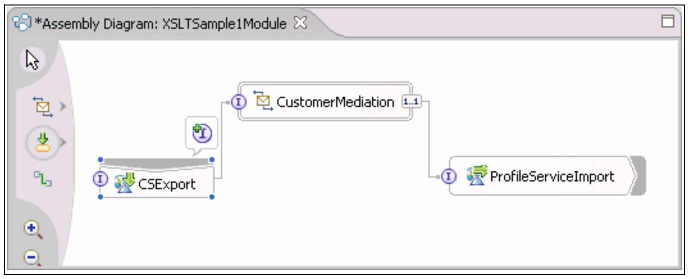


Figure 11-32 Wired module

5. Right-click on **CSExport** and select **Generate Binding -> Web Service Binding**.

6. When asked whether to automatically generate WSDL click **Yes**.

7. In the Select Transport dialog select soap/http and click **OK.**

8. Save the module.

9. Open the Physical Resources view by right-clicking on the **XSLTSample1Module** module and selecting **Show Files**.
10. Right click **CSExport_CustomerServiceHttp_Service.wsdl** from the WSClientSample1Module project and select **Open With -> WSDL Editor**.
11. Navigate to the soap address of the service (Figure 11-33).

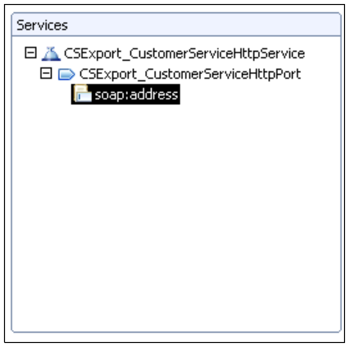
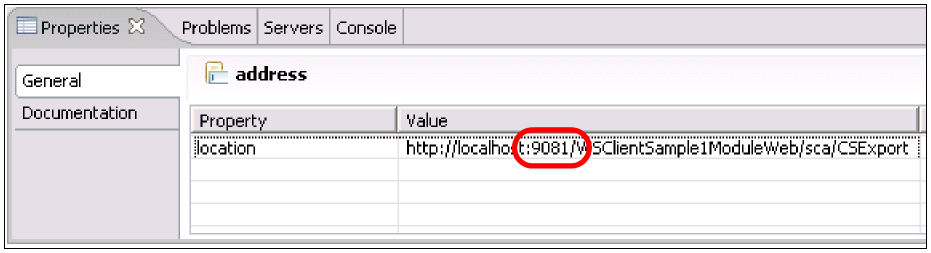

Services
CSExport_CustomerServiceHttpService
CSExport_CustomerServiceHttpPort
soap:address

Figure 11-33 Service soap address

12. In the Properties view, if required, change the port number in the location property to the default host port used by your server (Figure 11-34). In our environment, this was port 9081.



Property	Value
location	http://localhost:9081/WSClientSample1ModuleWeb/sca/CSExport

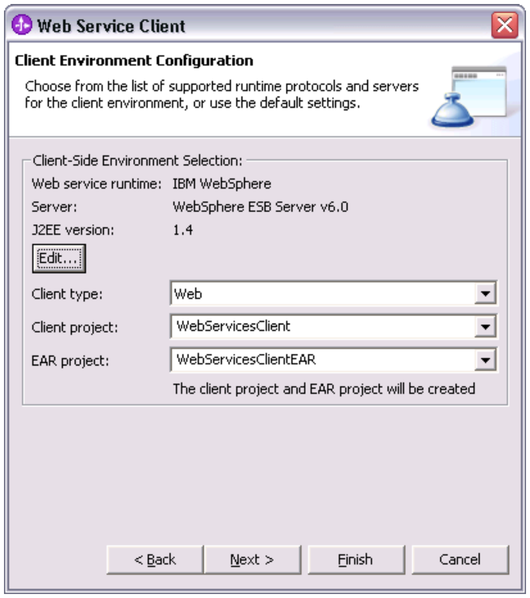
Figure 11-34 Service soap address location URL

13. Save the WSDL file and close the WSDL Editor.

Note: When generating a Web services client, the WSDL describing the interface and the WSDL describing the binding must be in the same project.

14. Copy **CSExport_CustomerServiceHttp_Service.wsdl** from the WSClientSample1Module project to the BookOrderResources project.

15. In the BookOrderResources project, right-click on **CSExport_CustomerServiceHttp_Service.wsdl** and select **Web services -> Generate client**.
16. On the Web service client dialog click **Next**.
17. On the Web service selection page click **Next**.
18. Complete the client environment configuration panel (Figure 11-35 on page 312).
    a. Ensure the server is set to your WebSphere Enterprise Service Bus server.
    a. Select **Web** as the client type.
    a. Set the Client Project to `WebServicesClient`.
    b. Set the EAR project to `WebServicesClientEAR`.


Web Service Client

Client Environment Configuration

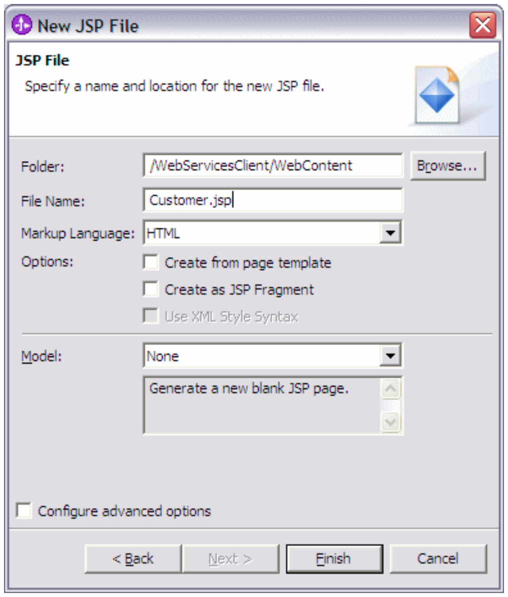
Choose from the list of supported runtime protocols and servers for the client environment, or use the default settings.

Client-Side Environment Selection:
Web service runtime: IBM WebSphere
Server: WebSphere ESB Server v6.0
J2EE version: 1.4

Client type: Web
Client project: WebServicesClient
EAR project: WebServicesClientEAR
The client project and EAR project will be created

Figure 11-35 Client environment configuration

19. Click **Finish**.
20. In the Physical Resources view expand the new project **WebServicesClient**, right click on the **WebContent** directory and select **New -> Other**.
21. In the New panel select **Web -> JSP File** and click **Next**.
22. Set the File Name to `Customer.jsp` and click **Finish** (Figure 11-36 on page 313).



Folder: /WebServicesClient/WebContent
File Name: Customer.jsp
Markup Language: HTML
Options:
- [ ] Create from page template
- [ ] Create as JSP Fragment
- [ ] Use XML Style Syntax

Model: None
Generate a new blank JSP page.

- [ ] Configure advanced options

Figure 11-36 New JSP dialog

23. The JSP will be opened in the Page Designer editor. Replace the contents with Example 11-1, then save and close the Page Designer editor.

> **Note:** The code for Customer.jsp is also available in the additional material supplied with this redbook in the directory:
>
> `\Clients\Resources\WebServices`

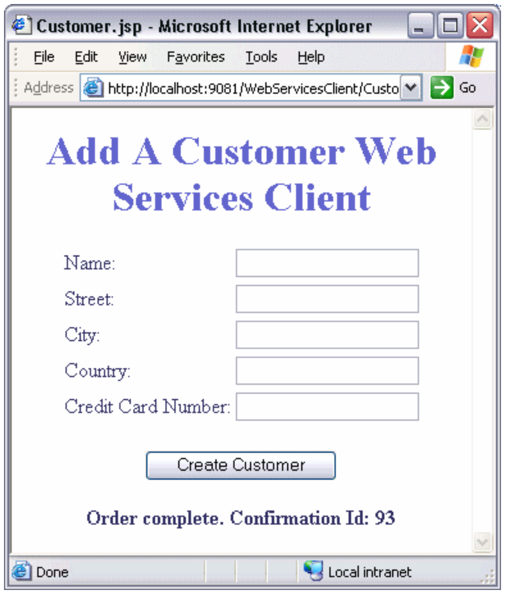
Example 11-1 Customer.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">


<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta name="GENERATOR" content="IBM Software Development Platform" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<link href="theme/Master.css" rel="stylesheet"
   type="text/css" />
<title>Customer.jsp</title>
```

```
</head>
<body>
<div style="text-align: center">
<h1>Add a Customer Web services Client</h1>
   <form method="get" action="Customer.jsp">
      <table>
         <tr>
            <td>Name:</td>
            <td><input type="text" name="name"/></td>
         </tr>
         <tr>
            <td>Street:</td>
            <td><input type="text" name="street"/></td>
         </tr>
         <tr>
            <td>City:</td>
            <td><input type="text" name="city"/></td>
         </tr>
         <tr>
            <td>Country:</td>
            <td><input type="text" name="country"/></td>
         </tr>
         <tr>
            <td>Credit Card Number:</td>
            <td><input type="text" name="creditCardNumber"/></td>
         </tr>
      </table>
      <br/>
      <input name="create" type="submit" value="Create Customer"/>
   </form>
</div>
<div style="text-align: center; font-weight: bolder;">

<!-- Create an instance of the proxy for our Customer Service -->
<jsp:useBean id="customerServiceProxy" scope="session"
class="BookOrderResources.CustomerServiceProxy"/>
<!-- Create an instance of a Customer requied for the CustomerService -->
<jsp:useBean id="customer" scope="session"
class="BookOrderResources.Customer"/>
<%
if (request.getParameter("name") != null) {

   try {
      //Fill in the fields of the customer
      customer.setName(request.getParameter("name"));
      customer.setStreet(request.getParameter("street"));
      customer.setCity(request.getParameter("city"));
      customer.setCountry(request.getParameter("country"));
      customer.setCreditCardNum(request.getParameter("creditCardNumber"));
```

```
            customer.setLastUpdate(new java.util.Date());

            //Call the addCustomer operation on the CustomerService Web service
            String confirmation = customerServiceProxy.addCustomer(customer);
            //Display the confirmation ID
            out.println("<p>Order complete. Confirmation Id: "+confirmation+"</p>");
        }
        catch (Exception e) {
            out.println(e);
        }
    }
%>
</div>
</body>
</html>
```

24. Deploy the module the Web service and the client to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add ProfileServiceEAR, XSLTSample1ModuleApp and WebServiceClientEAR.
    d. Click **Finish**.
25. Open a Web browser and enter the URL `http://localhost:9081/WebServicesClient/Customer.jsp` (the port number may vary depending on your install).
26. Fill in all the fields and click **Create Customer**.
27. A message is displayed containing the confirmation ID returned by the module (Figure 11-37).


Customer.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Go
Done
Local intranet

http://localhost:9081/WebServicesClient/Custo

Add A Customer Web Services Client

Name:

Street:

City:

Country:

Credit Card Number:

Create Customer

Order complete. Confirmation Id: 93

Figure 11-37 Customer.jsp confirmation

Congratulations you have successfully demonstrated how to create a Web services client that communicates with a mediation module.

28. Remove the projects from the test server.

11.2.2 JMS client

This sample demonstrates how to create a JMS client, capable of communicating with a mediation module, using an export with a JMS binding.

A JMS client uses the Java Messaging Service to send a message to JMS queue. The message contains all the information required to invoke a messaging service.

This sample involves:

- Defining server JMS resources required by the client and module.
- Creating a mediation module to accept JMS request messages and call a Web service.
- Writing a JMS client to send request messages and wait for a response message.
- Using the JMS client to invoke a Web service.

The completed sample demonstrates a JMS client making a request to create a new profile. The mediation module converts the JMS request message to a Web services request message and invokes a Web service that creates a profile. The module then converts the Web service response message to a JMS response message. The JMS client receives the response message containing confirmation that a customer profile has been created.

Attention: This sample requires WebSphere Enterprise Service Bus Fixpack1

There are four JMS resources that are required for a mediation module to exchange messages with a JMS client.

- Input queue - This is used by the mediation module to receive request messages.
- Output queue - This is used by the mediation module to send response messages.
- QueueConnectionFactory - This is used by the mediation module to connect to the bus hosting the input and output queues.
- ActivationSpec - This is used by the mediation module to register the input queue with the mediation module's message listener.

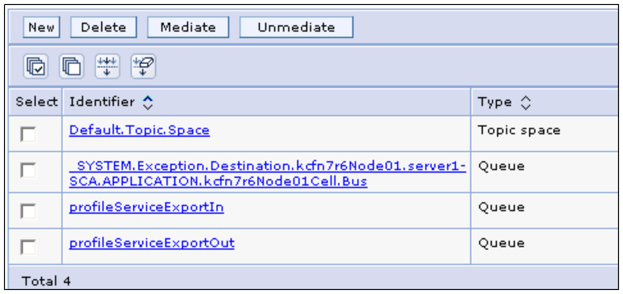
Create these resources using the administrative console.

1. In the Servers view, right-click on the WebSphere Enterprise Service Bus server and select **Run administrative console**.
2. Log in to the console.
3. First we need to create the physical queues on the bus. Click **Service Integration -> Buses**.
4. Select **SCA.APPLICATION.esbCell.Bus**.

   Note: If your WebSphere Enterprise Service Bus cell is not named esbCell then the bus name will reflect your own cell name.

5. In the Destination resources section, click **Destinations**.
6. Click **New**, and click **Next** to create a new Queue.
7. Set the Identifier field to `profileServiceExportIn`.
8. Click **Next** to see the Assign Queue to Bus member panel.
9. Click **Next** to see the Confirm queue creation panel.
10. Click **Finish**.

11. Create another new queue using the same method, called `profileServiceExportOut`.
12. You should see both queues in the list of destinations (Figure 11-38).



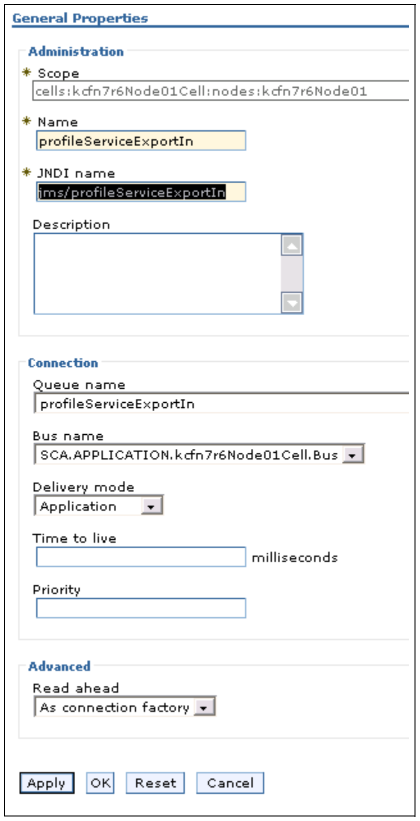
Select	Identifier	Type
☐	Default.Topic.Space	Topic space
☐	_SYSTEM.Exception.Destination.kcfn7r6Node01.server1-SCA.APPLICATION.kcfn7r6Node01Cell.Bus	Queue
☐	profileServiceExportIn	Queue
☐	profileServiceExportOut	Queue

Total 4

Figure 11-38 List of destinations

We need to add references to these queues in JNDI so the client and module can access them.

13. Click **Resources -> JMS Providers -> Default messaging**.
14. Under Destinations, click **JMS queue**.
15. Click **New** to create a new queue (Figure 11-39).
    a. Set the name field to `profileServiceExportIn`.
    b. Set the JNDI name field to `jms/profileServiceExportIn`.
    c. Select the **SCA.APPLICATION.esbCell.Bus** bus from the Bus name drop down menu.
    d. Select **profileServiceExportIn** from Queue name drop down menu.


General Properties

Administration

* Scope: cells:kcfn7r6Node01Cell:nodes:kcfn7r6Node01
* Name: profileServiceExportIn
* JNDI name: jms/profileServiceExportIn

Description:

Connection

Queue name: profileServiceExportIn

Bus name: SCA.APPLICATION.kcfn7r6Node01Cell.Bus

Delivery mode: Application

Time to live: milliseconds

Priority:

Advanced

Read ahead: As connection factory

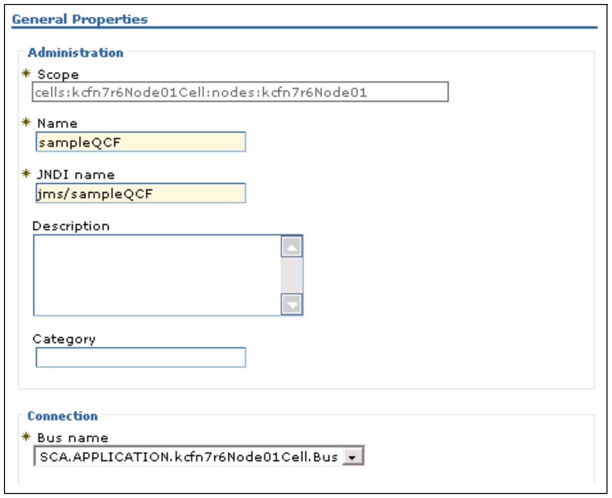
Figure 11-39 Administrative Console defining a queue.

16. Click **OK**.
17. Click **New** to create another new queue.
    e. Set the name field to `profileServiceExportOut`.
    f. Set the **JNDI name** field to `jms/profileServiceExportOut`.
    g. Select the **SCA.APPLICATION.esbCell.Bus** bus from the Bus name drop down menu.
    h. Select queue **profileServiceExportOut** from the Queue name drop down menu.
18. Click **OK**.

Now we need to create the QueueConnectionFactory so the client and the module can access the bus hosting the input and output queues.

19. Click **Resources -> JMS Providers -> Default messaging**.

20. Under Connection Factories, click **JMS queue connection factory**.
21. Click **New** to create a queue connection factory (Figure 11-40).
    a. Set the name field to `sampleQCF`.
    b. Set the JNDI name field to `jms/sampleQCF`.
    c. Select the **SCA.APPLICATION.esbCell.Bus** bus from the Bus name drop down menu.


General Properties

Administration

* Scope: cells:kcfn7r6Node01Cell:nodes:kcfn7r6Node01

* Name: sampleQCF

* JNDI name: jms/sampleQCF

Description:

Category:

Connection

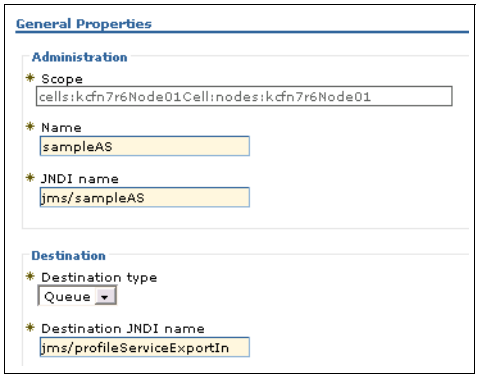
* Bus name: SCA.APPLICATION.kcfn7r6Node01Cell.Bus

Figure 11-40 Creating a new queue connection factory

22. Click **OK**.

Now we need to create the activation specification so the module can register the input queue with its message listener.

23. Click **Resources -> JMS Providers -> Default messaging**.
24. Under Activation Specifications, click **JMS activation specification**.
25. Click **New** to create an activation specification (Figure 11-41).
    a. Set the name field to `sampleAS`.
    b. Set the JNDI name field to `jms/sampleAS`.
    c. Select the destination type as **Queue.**
    d. Set the Destination JNDI name to `jms/profileServiceExportIn`.
    e. Select the **SCA.APPLICATION.esbCell.Bus** bus from the Bus name drop down menu.


General Properties

Administration

* Scope
cells:kcfn7r6Node01Cell:nodes:kcfn7r6Node01

* Name
sampleAS

* JNDI name
jms/sampleAS

Destination

* Destination type
Queue

* Destination JNDI name
jms/profileServiceExportIn

Figure 11-41 Creating a new activation specification

26. Click **OK**.
27. Click the **Save** link at the top of the console.
28. Press the **Save** button to confirm the save.
29. Log out and close the administrative console.
30. At this point you will need to restart your server for the new resources to take effect.

Next we must create a mediation module that will listen for JMS messages on the input queue profileServiceExportIn, call a Web service and return a JMS response message to the queue profileServiceExportOut.

31. Create a new mediation module.
    a. In the Business Integration view, right-click and select **New -> Mediation Module**.
    b. Set the Module Name to `JMSClientSample1Module` and click **Next.**
    c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
32. Open the module in the Assembly Editor.
33. Add an export to the assembly diagram and name it `ProfileServiceExport.`
34. Right-click on **ProfileServiceExport** and select **Add Interface**.
35. Select the **ProfileService** interface and click **OK**.
36. Add an import to the assembly diagram and name it `ProfileServiceImport.`
37. Right-click on **ProfileServiceImport** and select **Add Interface**.
38. Select the **ProfileService** interface and click **OK**.

39. Rename the mediation flow component Mediation1 to `MyMediationFlowComponent`.
40. Wire the components together (Figure 11-42).
    a. Wire ProfileServiceExport to MyMediationFlowComponent.
    b. Wire MyMediationFlowComponent to ProfileServiceImport.

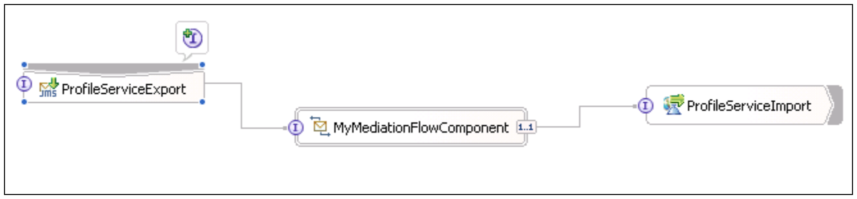
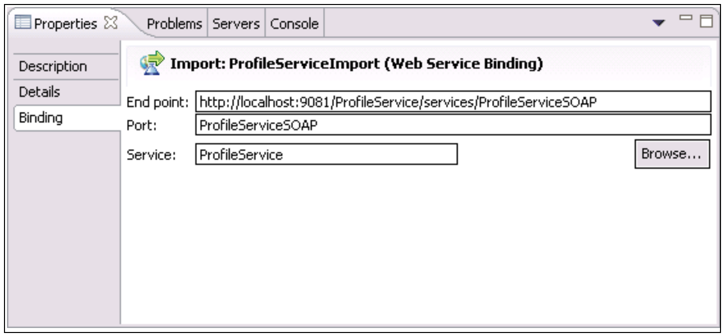


Figure 11-42 JMSClientSample1Module

41. Right-click on **ProfileServiceImport** and select **Generate Binding -> Web Service Binding**.
42. Click on **ProfileServiceImport** and in the Properties view, select the **Binding** tab and click **Browse**.
43. From the project BookOrderResources select **ProfileServiceBinding.wsdl** and click **OK**. The binding should now look like Figure 11-43.

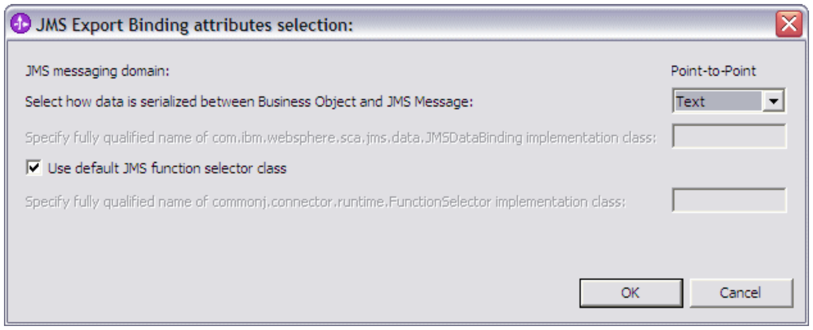


Import: ProfileServiceImport (Web Service Binding)

End point: http://localhost:9081/ProfileService/services/ProfileServiceSOAP
Port: ProfileServiceSOAP
Service: ProfileService

Figure 11-43 SOAP/HTTP import binding

44. Right-click **ProfileServiceExport** and select **Generate Binding -> JMS Binding**.
45. In the JMS Export Binding attributes selection dialog, select **Text** from the Select how data is serialized between Business Object and JMS Message drop down menu and click **OK** (Figure 11-44 on page 323).


JMS Export Binding attributes selection:

JMS messaging domain: Point-to-Point

Select how data is serialized between Business Object and JMS Message: Text

Specify fully qualified name of com.ibm.websphere.sca.jms.data.JMSDataBinding implementation class:

☑ Use default JMS function selector class

Specify fully qualified name of commonj.connector.runtime.FunctionSelector implementation class:

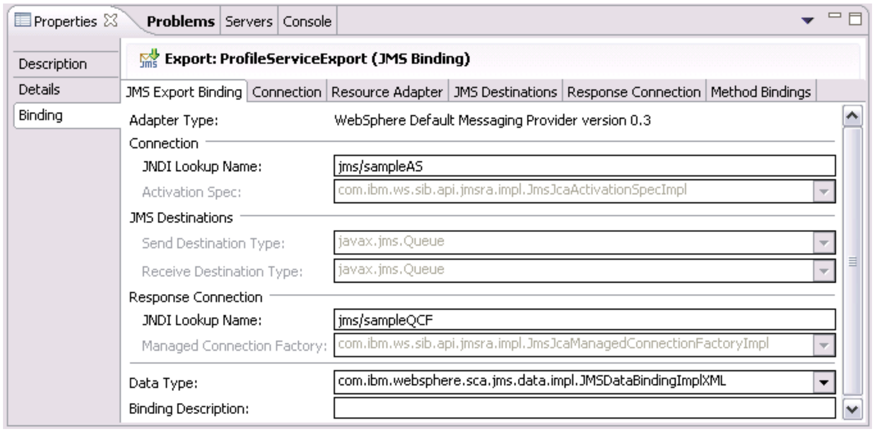
Figure 11-44 JMS Export Binding attributes selection

46. Select the ProfileServiceExport and in the Properties view, select the **Binding** tab.

47. In the displayed panel select the **JMS Export Binding** tab. These properties allow us to define how the module will connect to the input and output queues (Figure 11-45).

    a. Set the Connection JNDI Lookup Name for the input queue to `jms/sampleAS`.

    b. Set the Response Connection JNDI Lookup Name to for the output queue to `jms/sampleQCF`.



Export: ProfileServiceExport (JMS Binding)

Adapter Type: WebSphere Default Messaging Provider version 0.3

Connection

JNDI Lookup Name: jms/sampleAS

Activation Spec: com.ibm.ws.sib.api.jmsra.impl.JmsJcaActivationSpecImpl

JMS Destinations

Send Destination Type: javax.jms.Queue

Receive Destination Type: javax.jms.Queue

Response Connection

JNDI Lookup Name: jms/sampleQCF

Managed Connection Factory: com.ibm.ws.sib.api.jmsra.impl.JmsJcaManagedConnectionFactoryImpl

Data Type: com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML

Binding Description:

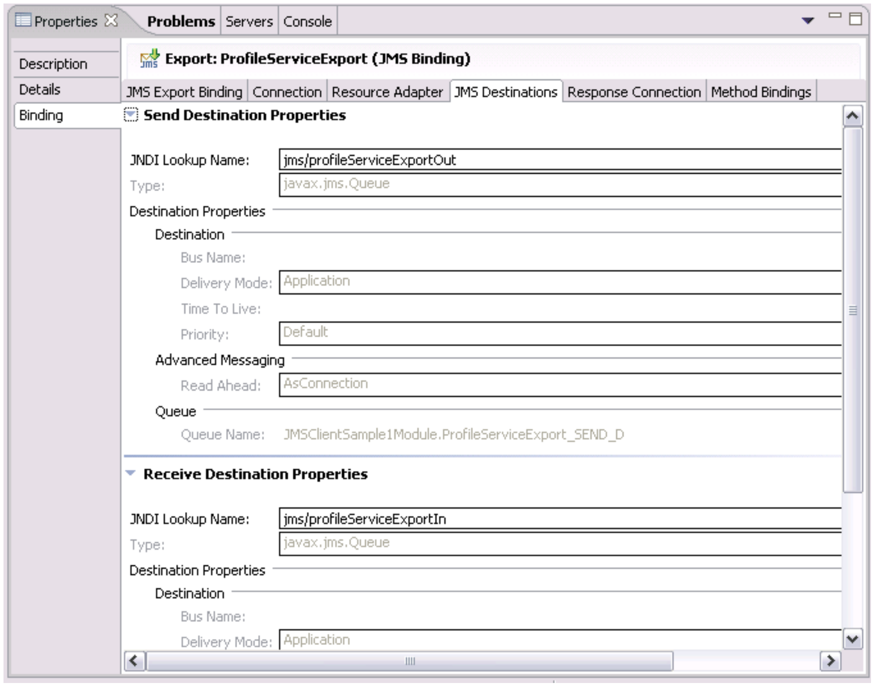
Figure 11-45 JMS Export binding

48. Select the **JMS Destinations** tab and expand the **Send Destination Properties**. This will be the queue used by the export to send response messages.

49. Set the JNDI Lookup Name to `jms/profileServiceExportOut`.

50. Expand the **Receive Destination Properties**. This will be the queue used by the mediation module to receive request messages.

51. Set the JNDI Lookup Name to `jms/profileServiceExportIn` (Figure 11-46).


Properties | Problems | Servers | Console

Description | Details | Binding

Export: ProfileServiceExport (JMS Binding)

JMS Export Binding | Connection | Resource Adapter | JMS Destinations | Response Connection | Method Bindings

Send Destination Properties

JNDI Lookup Name: jms/profileServiceExportOut
Type: javax.jms.Queue
Destination Properties
Destination
Bus Name:
Delivery Mode: Application
Time To Live:
Priority: Default
Advanced Messaging
Read Ahead: AsConnection
Queue
Queue Name: JMSClientSample1Module.ProfileServiceExport_SEND_D

Receive Destination Properties

JNDI Lookup Name: jms/profileServiceExportIn
Type: javax.jms.Queue
Destination Properties
Destination
Bus Name:
Delivery Mode: Application

Figure 11-46 JMS Export binding destination properties

52. Select the **Method Bindings** tab.

53. Under Bound Methods click **add**.

54. Select **com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML** from the In Data Type drop down menu.

55. Select **com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML** from the Out Data Type drop down menu.

56. Back in the assembly diagram, right-click **MyMediationModule** and select **Generate Implementation** then click OK to accept the default destination. This opens the Mediation Flow Editor.

57. In this sample we are only concerned with the add operation so wire that **add** operation on the ProfileService interface to the **add** operation on the ProfileServicePartner reference (Figure 11-47).

58. Add a Message Logger mediation primitive to the request flow.

59. Wire the request flow (Figure 11-47 on page 325).

   a. Wire **ProfileService_add_Input** to the **in** terminal of MessageLogger1.

   b. Wire the **out** terminal of MessageLogger1 to **ProfileServicePartner_add_Callout**.

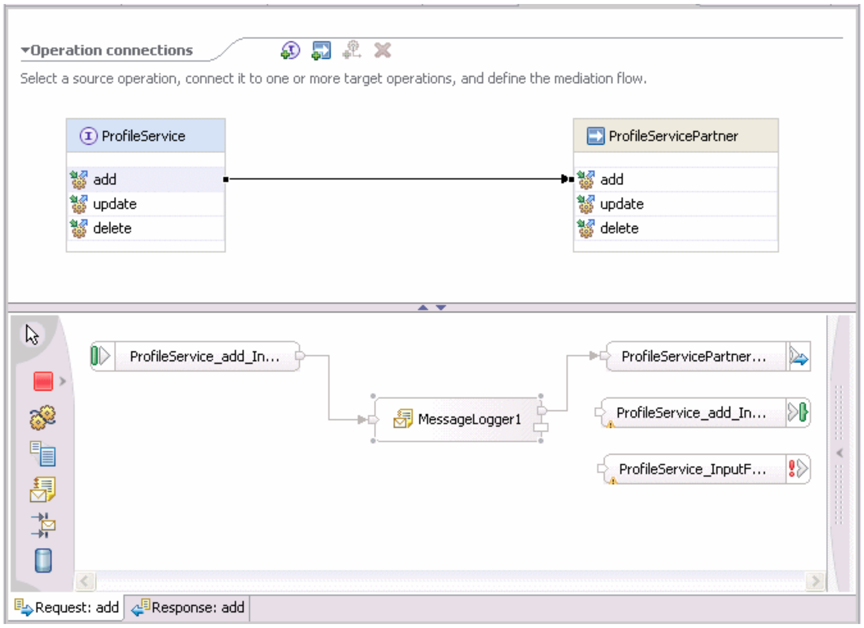


Figure 11-47 Mediation flow for MyMediationFlowComponent

60. Click on the **Response** tab to display the response flow.

61. Wire the response flow (Figure 11-48).

   a. Wire **ProfileServicePartner_add_CalloutResponse** to **ProfileService_add_InputResponse**.

   b. Wire **ProfileServicePartner_CalloutFault** to **ProfileService_InputFault**.

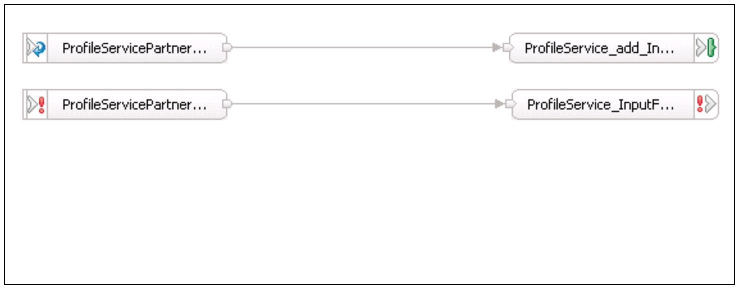


Figure 11-48 Response Flow

62. Save the mediation flow and the module.

Now we have a module that listens for incoming request messages and returns response messages. Now lets create a JMS client capable of sending request messages to the module and receiving responses. We will use a JSP so we can easily enter our profile information and print the response to a Web browser.

63. In the Business Integration view right click **JMSClientSample1Module** and click **Show Files** to open the Physical Resources view. In the Physical Resources view, right-click **JMSClientSample1Module** and select **New -> Other**.
64. Select **Web -> Dynamic Web Project** and click **Next**.
65. Enter the name as `JMSClientSample1`.
66. Click the **Show Advanced** button and ensure the Target server is set to your WebSphere Enterprise Service Bus server.
67. Click **Finish**. If asked to switch to the Web perspective click **No**.
68. In the new JMSClientSample1 project right-click on the **WebContent** directory and select **New -> Other**.
69. Select **Web -> JSP File** and click **Next**.
70. Name the JSP `Profile.jsp` and click **Finish**.
71. The contents of Profile.jsp are displayed in the Page Designer. Replace the contents with code shown in Example 11-2 on page 327, then save and close the Page Designer.

Note: The code for Profile.jsp is also available in the additional material supplied with this redbook in the following directory:

```
\Clients\Resources\JSP
```

Example 11-2 Profile.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="GENERATOR" content="IBM Software Development Platform">
<title>Profile.jsp</title>
</head>
<body>

<div style="text-align: center">
   <h1>Add a Profile JMS Client</h1>
   <form method="get" action="Profile.jsp">
      <table>
         <tr>
            <td>Name:</td>
            <td><input type="text" name="name"/></td>
         </tr>
         <tr>
            <td>Street:</td>
            <td><input type="text" name="street"/></td>
         </tr>
         <tr>
            <td>City:</td>
            <td><input type="text" name="city"/></td>
         </tr>
         <tr>
            <td>Country:</td>
            <td><input type="text" name="country"/></td>
         </tr>
         <tr>
            <td>Credit Card Number:</td>
            <td><input type="text" name="creditCardNumber"/></td>
         </tr>
      </table>
      <br/>
      <input name="create" type="submit" value="Create Profile"/>
   </form>

<%
if (request.getParameter("name") != null) {

   //The Initial Context Factory
   String icf = "com.ibm.websphere.naming.WsnInitialContextFactory";
   //the Provider URL
   String url = "iiop://localhost:2810/";
   //The Queue Connection Factory used to connect to the bus
   String sampleQCF = "jms/sampleQCF";
   //The Queue used to send requests to the mediation module
   String sampleSendQueue = "jms/profileServiceExportIn";
   //The Queue used to receive responses from the mediation module
   String sampleReceiveQueue = "jms/profileServiceExportOut";
   //The XML representation of a Profile which is the Business Object required
by the add operation
   //on the ProfileService Interface
   String message = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";
   message += "<xs1:Profile xmlns:xs1=\"http://BookOrderResources\">";
   message +=   "<name>"+request.getParameter("name")+"</name>";
   message +=   "<xs2:Address xmlns:xs2=\"http://BookOrderResources\">";
   message +=      "<street>"+request.getParameter("street")+"</street>";
   message +=      "<city>"+request.getParameter("city")+"</city>";
   message +=      "<country>"+request.getParameter("country")+"</country>";
   message +=   "</xs2:Address>";
   message +=
"<creditCardNum>"+request.getParameter("creditCardNum")+"</creditCardNum>";
   message +=   "<lastUpdate>2006-02-16</lastUpdate>";
   message +=  "</xs1:Profile>";

   try {
      //Create the Initial Context
       java.util.Hashtable env = new java.util.Hashtable();
        env.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, icf);
        env.put(javax.naming.Context.PROVIDER_URL, url);
        javax.naming.Context ctx = new
javax.naming.directory.InitialDirContext(env);

        //Lookup the ConnectionFactory
        javax.jms.ConnectionFactory factory =
(javax.jms.ConnectionFactory)ctx.lookup(sampleQCF);
      //Create a Connection
        javax.jms.Connection connection = factory.createConnection();
        //Start the Connection
        connection.start();
        //Create a Session
        javax.jms.Session jmsSession = connection.createSession(false,
javax.jms.Session.AUTO_ACKNOWLEDGE);
        //Lookup the send Destination
        javax.jms.Destination sendQueue = (javax.jms.Destination)
ctx.lookup(sampleSendQueue);
        //Create a MessageProducer
        javax.jms.MessageProducer producer =
jmsSession.createProducer(sendQueue);
        //Create the TextMessage that will hold out profile as text
        javax.jms.TextMessage sendMessage = jmsSession.createTextMessage();
        //Set the content of the message to be the XML defined Profile
        sendMessage.setText(message);
        //Set the operation to call on the ProfileService interface to be add
        sendMessage.setStringProperty("TargetFunctionName", "add");
        //Send the message
        producer.send(sendMessage);

        //Lookup the receive Destination
        javax.jms.Destination receiveQueue = (javax.jms.Destination)
ctx.lookup(sampleReceiveQueue);
        //Create a MessageConsumer
        javax.jms.MessageConsumer consumer =
jmsSession.createConsumer(receiveQueue);
        //Wait 15 seconds to receieve the response
        javax.jms.TextMessage receiveMessage = (javax.jms.TextMessage)
consumer.receive(15000);
        //If we receive a response print the contents of the message to the
screen
        String confirmation = "Profile creation failed.";
        if (receiveMessage != null) {
        //The contents of the message will be a Confirmation object that
contains a String.
        confirmation = "Profile created.<br/>Confirmation Id:
"+receiveMessage.getText();
        }
      out.println("<p>"+confirmation+"</p>");

      //Close the Connection
        connection.close();
   }
   catch (Exception e) {
      out.println(e);
   }
}

%>
</div>
</body>
</html>
```

To test the client, mediation module and Web service you need to deploy all three enterprise applications to the WebSphere Enterprise Service Bus server.

72. Deploy the mediation module, Web service and JMS client to the server.

   a. Switch to the Servers view.

   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.

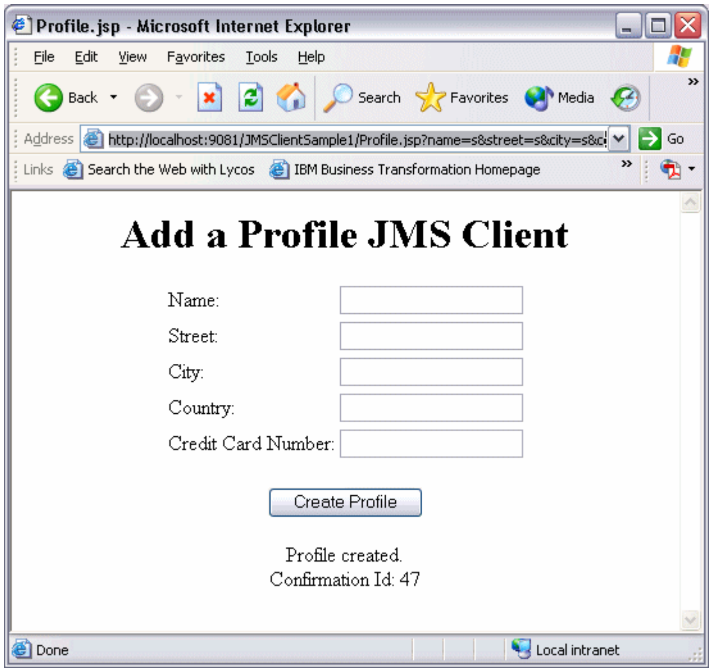
   c. Add JMSClientSample1ModuleApp, ProfileServiceEAR and JMSClientSample1EAR.

   d. Click **Finish**.

73. Open a Web browser and enter the URL `http://localhost:9081/JMSClientSample1/Profile.jsp` (the port number may vary depending on your install).

74. Fill in all the fields and click **Create Profile**.

75. You should see a message containing the confirmation Id returned by the module (Figure 11-49).


Profile.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9081/JMSClientSample1/Profile.jsp?name=s&street=s&city=s&c Go
Links Search the Web with Lycos IBM Business Transformation Homepage
Create Profile
Done Local intranet

Add a Profile JMS Client

Name:	
Street:	
City:	
Country:	
Credit Card Number:	

Profile created.
Confirmation Id: 47

Figure 11-49 Profile.jsp confirmation

Congratulations you have successfully demonstrated how to create a JMS client and mediation module allowing this client to communicate with a Web service.

76. Remove the projects from the test server.

11.2.3 SCA client

This sample demonstrates how to use the SCA programming model to create a client capable of communicating with a mediation module.

An SCA client uses the Java SCA programming model to communicate with an SCA component.

This sample involves:

- Creating a mediation module to accept SCA calls and return a response.
- Writing an SCA client to send requests and receive responses.
- Using the SCA client to invoke an operation on the SCA module.

The completed sample demonstrates an SCA client making a request to order a book. The mediation module uses a Java component to return a confirmation of the order. The client receives a confirmation that the order was successful.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `SCAClientSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Right-click on Mediation1 and select **Delete**. The mediation flow component is not required to test the SCA client.

Next we need to create a stand-alone reference. This allows an SCA client to communicate with the module.

4. In the Assembly Editor add a stand-alone reference to the assembly diagram.
5. Right-click on the stand-alone reference and select **Add Reference**.
6. Select the **BookOrderService** and click **OK**.
7. If asked to convert the WSDL interface to a Java interface click **Yes**.

Note: We will use a Java component to implement the BookOrderService.

8. Add a Java Component to the assembly diagram.
9. Right-click the Java component and select **Add -> Interface**.
10. Click the **Show WSDL** radio button, select the **BookOrderService** interface and click **OK**.
11. Wire the stand-alone reference to the Java component (Figure 11-50).

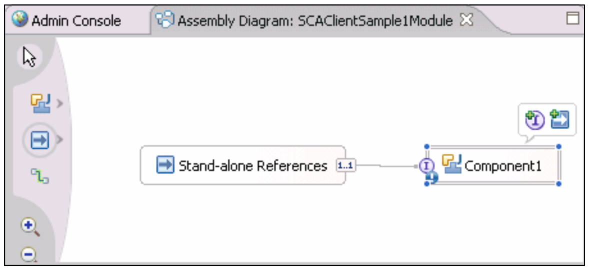


Figure 11-50 Wiring stand-alone reference to java component

12. Right-click on the Java component **Component1** and select **Generate Implementation**.
13. Click **OK** to create in the default package.
14. The default implementation is fine, so save the file and close it.

> **Note:** For visual purposes, this sample uses a JSP to provide a Web-based front end to the mediation module. However the SCA programming model can be used to access mediation modules from any Java component such as an EJB or Web project.

15. Create a new dynamic Web project.
    a. Open the Physical Resources view by right-clicking on the **SCAClientSample1Module** module and selecting **Show Files**.
    a. Right-click **SCAClientSample1Module** and select **New -> Other**.
    b. In the New Project wizard select **Web -> Dynamic Web Project** and click **Next**.
    c. Enter the name of the project as `SCAClientSample1`.
    d. Click the **Show Advanced** button.
    e. Set the Target server to your WebSphere Enterprise Service Bus server.
    f. Uncheck the **Add module to EAR project** check box.
    g. Click **Finish**.
    h. If asked to switch to the Web perspective, click **No**.
16. In the SCAClientSample1 project, right-click the **WebContent** directory and select **New -> Other**.
17. Select **Web -> JSP File** and click **Next**.
18. Set the File Name to `BookOrder.jsp` and click **Finish**.

19. Replace the contents of the JSP with Example 11-3, then save and close the Page Designer.

Note: The code for BookOrder.jsp is also available in the additional material supplied with this redbook in the following directory:

```
\Clients\Resources\SCA
```

Example 11-3 BookOrder.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<meta name="GENERATOR" content="IBM Software Development Platform" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<link href="theme/Master.css" rel="stylesheet"
   type="text/css" />
<title>BookOrder.jsp</title>
</head>
<body>
<%@ page import="com.ibm.websphere.sca.ServiceManager" %>
<%@ page import="com.ibm.websphere.sca.Service" %>
<%@ page import="commonj.sdo.DataObject" %>
<%@ page import="com.ibm.websphere.bo.BOFactory" %>

<div style="text-align: center">
   <h1>Book Order Client</h1>
   <form method="get" action="BookOrder.jsp">
      <table>
         <tr>
            <td>CustomerId:</td>
            <td><input type="text" name="customerId"/></td>
         </tr>
         <tr>
            <td>Title:</td>
            <td><input type="text" name="title"/></td>
         </tr>
         <tr>
            <td>Author:</td>
            <td><input type="text" name="author"/></td>
         </tr>
         <tr>
```

```
                <td>Description:</td>
                <td><input type="text" name="description"/></td>
            </tr>
            <tr>
                <td>Quantity:</td>
                <td>
                    <select name="quantity">
                        <option value="1">1</option>
                        <option value="2">2</option>
                        <option value="3">3</option>
                        <option value="4">4</option>
                        <option value="5">5</option>
                    </select>
                </td>
            </tr>
        </table>
        <br/>
        <input name="order" type="submit" value="Order"/>
    </form>
</div>
<div style="text-align: center; font-weight: bolder;">
<%
if (request.getParameter("customerId") != null &&
    request.getParameter("title") != null &&
    request.getParameter("author") != null &&
    request.getParameter("description") != null) {

    try {
        // First lets get the Service we will be using
        ServiceManager serviceManager = new ServiceManager();
        Service service = (Service)
serviceManager.locateService("BookOrderServicePartner");

        // we can get the BOFactory via its 'well known' location name
        // com/ibm/websphere/bo/BOFactor
        BOFactory bofactory = (BOFactory)
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");

        // Create an input message by specifying it's element type
        DataObject order =
bofactory.createByElement("http://BookOrderResources/BookOrderService",
"order");

        // Get the part of message, in this case a BookOrder
        DataObject bookOrder = order.createDataObject("bookOrder");

        // Set it's fields from the values entered in the web form
        bookOrder.setString("customerId", request.getParameter("customerId"));
        bookOrder.setString("quantity", request.getParameter("quantity"));
        // Create a data object of type Book (as defined in the BookOrder)
        DataObject book = bookOrder.createDataObject("book");

        // Set it's fields
        book.setString("title", request.getParameter("title"));
        book.setString("author", request.getParameter("author"));
        book.setString("description", request.getParameter("title"));

        // Add the Book to the BookOrder
        bookOrder.setDataObject("book", book);

        // Now we can invoke an operation order whih returns a string
        String confirmationId = (String)service.invoke("order", bookOrder);
        // Finally put it on the browser page
        out.println("<p> Order complete. Confirmation Id: " + confirmationId +
"</p>");

    } catch (Exception e) {
        out.println(e);
    }
}
%>
</div>
</body>
</html>
```

Example 11-3 on page 333 shows how Java DataObjects are used to build a BookOrder business object. This is used as the input to the invoke of the order operation. The order operation returns a String which is displayed on the Web page.

In order for the SCAClientSample1 Web project to access the BookOrderServicePartner we need to add it as a Module dependency.

20. In the Business Integration view, double click on the **SCAClientSample1Module** project to open the dependency editor.
21. Expand **J2EE** and click the **Add** button.
22. Select the **SCAClientSample1** Web project and click **OK** (Figure 11-51 on page 336).

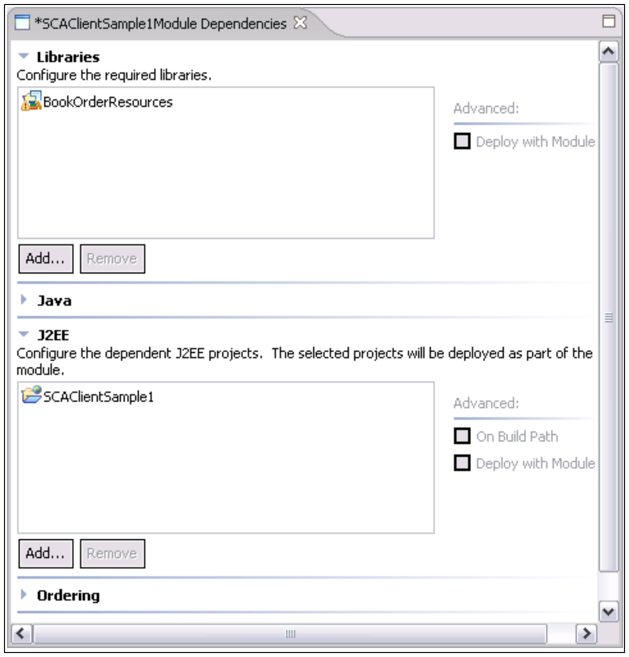

*SCAClientSample1Module Dependencies
Libraries
Configure the required libraries.
BookOrderResources
Advanced:
Deploy with Module
Add...
Remove
Java
J2EE
Configure the dependent J2EE projects. The selected projects will be deployed as part of the module.
SCAClientSample1
Advanced:
On Build Path
Deploy with Module
Add...
Remove
Ordering

Figure 11-51 Adding module dependencies

23. Save and close the dependencies editor.

Now we will test the JSP.

24. Deploy the module to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add SCAClientSample1ModuleApp.
    d. Click **Finish**.
25. Enter the URL `http://localhost:9081/SCAClientSample1/BookOrder.jsp` into a Web browser (the port number may vary depending on your install). This will display the JSP which should look like Figure 11-52 on page 337.

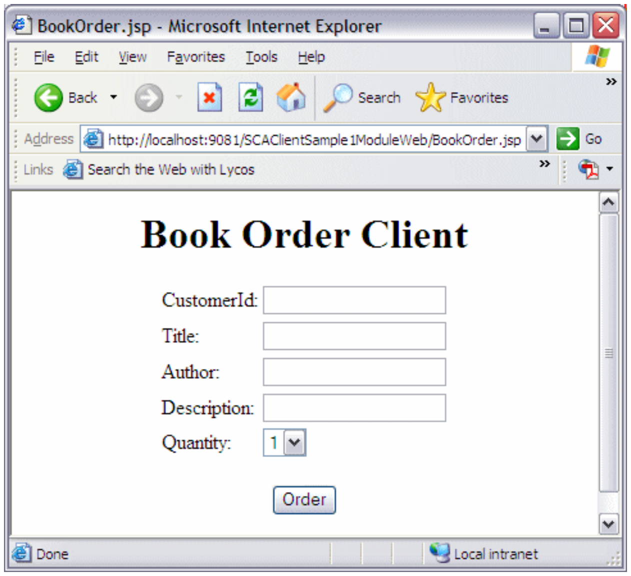

BookOrder.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost:9081/SCAClientSample1ModuleWeb/BookOrder.jsp
Go
Links Search the Web with Lycos
Book Order Client
CustomerId:
Title:
Author:
Description:
Quantity: 1
Order
Done
Local intranet

Figure 11-52 Book Order JSP

26. Enter values into each of the fields.
27. Click the **Order** button. A message is displayed confirming the order and containing the string from the Java component:

```
Order complete. Confirmation Id: Result from calling the order(DataObject
bookOrder) method.
```

Congratulations you have successfully demonstrated how to create and test a SCA client.

28. Remove the project from the test server.

11.3 Using services with mediation modules

This section describes some of the ways you can use services with mediation modules. Three development examples are provided, to demonstrate:

- How to perform mapping of bindings allowing transport protocol mapping between a client to a mediation module, and a reference from a mediation module.
- Mediation on the request flow of a service and the response flow of a service by using two mediation primitives.
- Fault handling in a mediation module.

Each of the development examples in this section can be imported as Project Interchange files from the additional material supplied with this redbook in the `\UsingServices\Solutions` directory.

11.3.1 Mapping bindings

This sample shows that different bindings can easily be mapped by wiring an export using one binding with an import using another binding. In this sample we will map a SOAP/HTTP request to a JMS request. This provides protocol transformation, which is a common functionality of ESBs.

Bindings can be defined on an import or export. On an export they describe how a client or SCA component communicates with the mediation module. On an import they describe how the mediation module communicates with the defined service.

This sample involves:

- Creating a business object.
- Creating an interface
- Creating a mediation module with an export using SOAP/HTTP connected directly to the import using JMS.
- Using the Integration Test Client to test the protocol mapping.

The completed sample demonstrates how a mediation module can be used to transcode a request that uses a different transport protocol to that used by the service being invoked.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Uncheck the box to create a mediation flow component.
   c. Set the Module Name to `MapBindingSample1Module` and click **Finish.**
2. In the Business Integration view, expand **MapBindingSample1Module**, right-click on **Data Types** and select **New -> Business Object**.
3. Set the Name to `SimpleBook` and click **Finish**.
4. In the business object editor, click the **Add Attribute** button.
5. Name the new attribute `title` and leave its type as string.
6. Save the business object (Figure 11-53 on page 339) and close the editor.

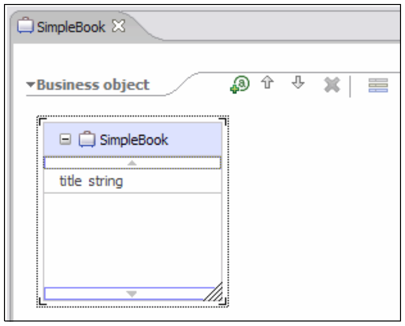
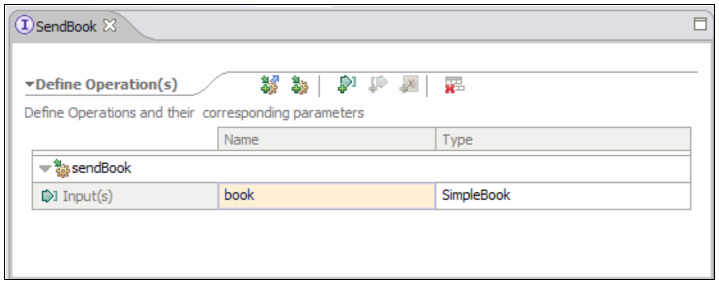


Figure 11-53 SimpleBook business object

7. In the Business Integration view, expand **MapBindingSample1Module**, right-click on **Interfaces** and select **New -> Interface**.
8. In the New Interface wizard set Name to `SendBook` and click **Finish**.
9. In the Interface editor click the **Add One Way Operation** button.
10. Name the operation `sendBook`.
11. Click the Add Input button.
12. Name the input `book` and give it a type of **SimpleBook**.
13. Save the interface (Figure 11-54) and close the interface editor.



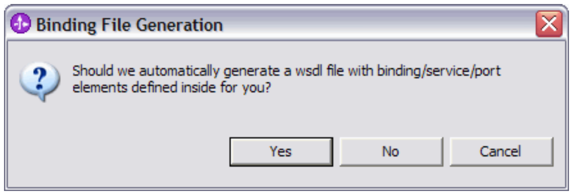
	Name	Type
sendBook		
Input(s)	book	SimpleBook

Figure 11-54 SendBook interface

14. Open the assembly diagram of the MapBindingSample1Module by double clicking .
15. Add an export to the assembly diagram by using .
16. Rename Export1 to `SOAPHTTPExport`.
17. Right-click **SOAPHTTPExport** and select **Add Interface**.
18. Select the **SendBook** interface and click **OK**.

19. Right-click **SOAPHTTPExport** and select **Generate Binding -> Web service binding**.

20. When prompted, if a WSDL file should be generated automatically, click **Yes** (Figure 11-55 on page 340).



Should we automatically generate a wsdl file with binding/service/port elements defined inside for you?

Figure 11-55 WSDL auto generation

21. Select **soap/http** as the transport in the Select Transport window, then click **OK**.

22. In the Business Integration view, you should now see the automatically created Web service port for the export (Figure 11-56).

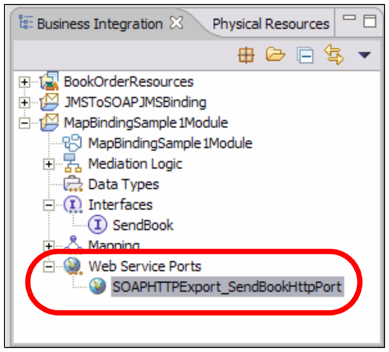


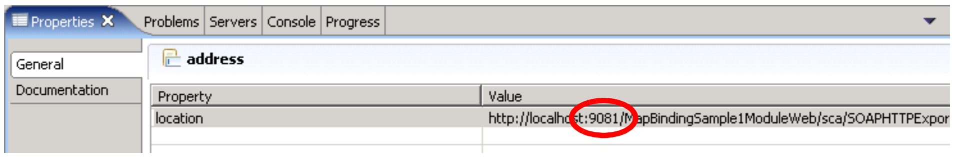
Figure 11-56 Web service port

23. We need to change the HTTP port specified in this WSDL file to represent the default host port used by your WebSphere Enterprise Service Bus server.

   a. Right click **SOAPHTTPExport_SendBookHttpPort** and select **Open With -> WSDL Editor**.

   b. In the Services area, expand **SOAPHTTPExport_SendBookHttpService -> SOAPHTTPExport_SendBookHttpPort** and click **soap:address**.

   c. In the Properties view, change the port of the location property to the default host port used by your server (Figure 11-57). In our environment this was port 9081.



Property	Value
location	http://localhost:9081/MapBindingSample1ModuleWeb/sca/SOAPHTTPExpor

Figure 11-57 SOAP address port change

d. Save the WSDL file and close the editor.

24. Back in the assembly diagram add an import to the module by using .

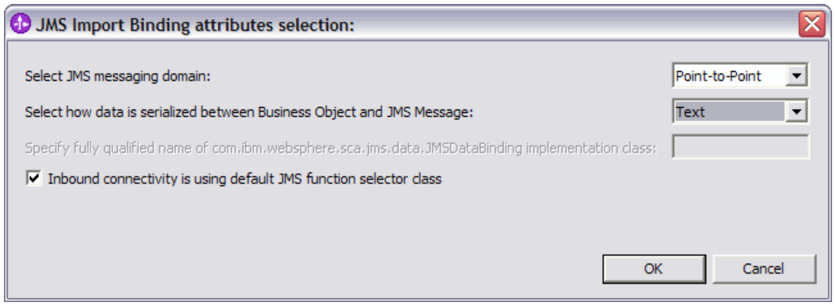
25. Rename Import1 to `JMSImport`.

26. Right-click **JMSImport** and select **Add Interface**.

27. Select the **SendBook** interface and click **OK**.

28. Right-click **JMSImport** and select **Generate Binding -> JMS binding**.

29. In the JMS Import Binding attributes selection window, select **Text** as the serialization method. Click **OK** (Figure 11-58).



JMS Import Binding attributes selection:

Select JMS messaging domain: Point-to-Point

Select how data is serialized between Business Object and JMS Message: Text

Specify fully qualified name of com.ibm.websphere.sca.jms.data.JMSDataBinding implementation class:

☑ Inbound connectivity is using default JMS function selector class

Figure 11-58 JMS import binding attributes selection

30. Wire the SOAPHTTPExport to the JMSImport (Figure 11-59).

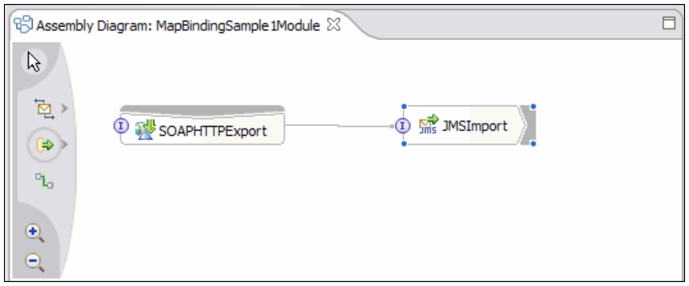
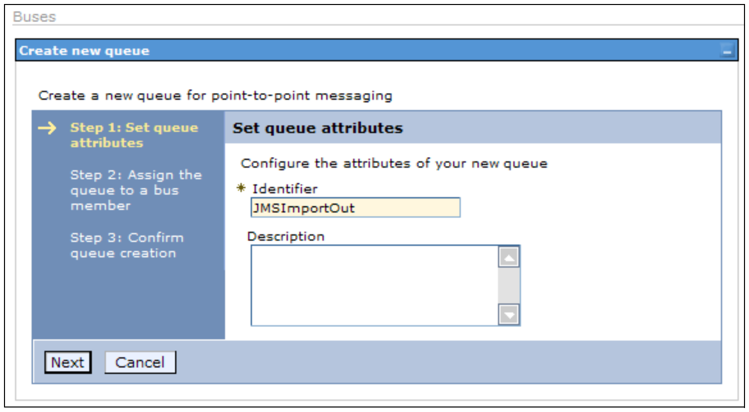


Figure 11-59 Wired module

31. Switch to the Servers view and start your WebSphere Enterprise Service Bus server, if not already running.
32. Right-click on the server and select **Run the administrative console** and log in.
33. Select **Service integration** → **Buses**.
34. Click on the **SCA.APPLICATION.esbCell.Bus** link. The Configuration window for that bus is displayed.
35. Under Destination resources on the right, click on **Destinations**.
36. You will see two pre-defined destinations, a topic and a queue.
37. Click **New** to create a new destination on the bus.
38. The next screen shows four types of destination that can be specified. Select **Queue** and click **Next**.
39. The queue attributes screen is displayed. Enter `JMSImportOut` as the identifier (Figure 11-60) then click **Next**.

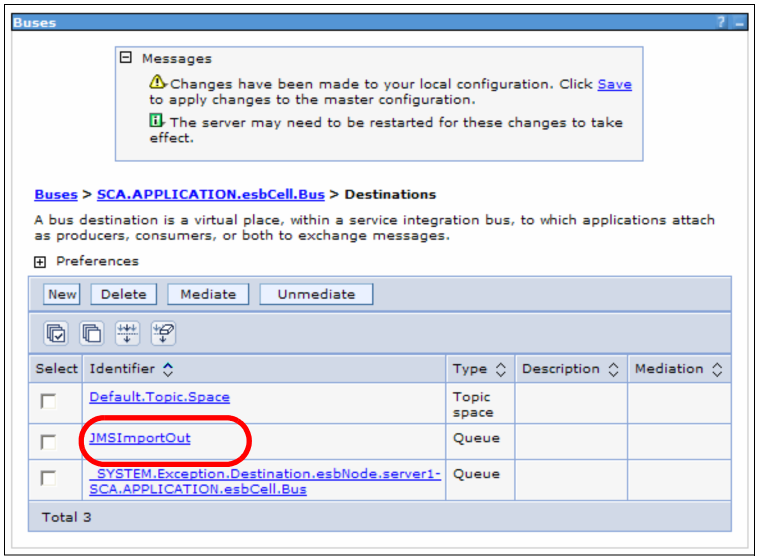

Buses
Create new queue
Create a new queue for point-to-point messaging
Step 1: Set queue attributes
Step 2: Assign the queue to a bus member
Step 3: Confirm queue creation
Set queue attributes
Configure the attributes of your new queue

* Identifier: JMSImportOut

Description:

Figure 11-60 Enter queue name

40. On the next screen, accept the queue assignments to a bus member and click **Next**.
41. On the confirmation screen, click **Finish** to create the new queue. The queue should now appear in the destinations list (Figure 11-61).



Messages

Changes have been made to your local configuration. Click Save to apply changes to the master configuration.

The server may need to be restarted for these changes to take effect.

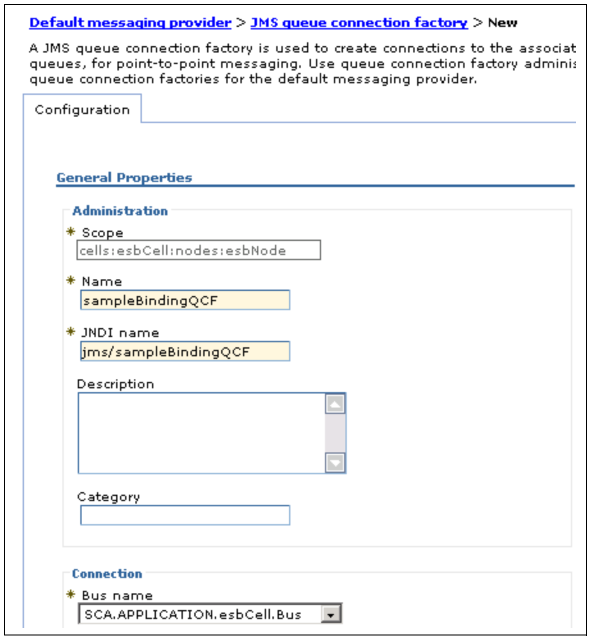
Buses > SCA.APPLICATION.esbCell.Bus > Destinations

A bus destination is a virtual place, within a service integration bus, to which applications attach as producers, consumers, or both to exchange messages.

Select	Identifier	Type	Description	Mediation
☐	Default.Topic.Space	Topic space		
☐	JMSImportOut	Queue		
☐	_SYSTEM.Exception.Destination.esbNode.server1-SCA.APPLICATION.esbCell.Bus	Queue		
Total 3				

Figure 11-61 Save new queue

42. In the navigation menu on the left, select **Resources** → **JMS Providers** → **Default messaging**.
43. Under Connection Factories, click **JMS queue connection factory**.
44. On the following screen, click **New**.
45. Fill in the properties of the queue connection factory (Figure 11-62).
    a. Set the Name as `sampleBindingQCF`.
    b. Set the JNDI Name to `jms/sampleBindingQCF`.
    c. From the drop down list for Bus name, choose **SCA.APPLICATION.esbCell.Bus**.
    d. Click **OK** at the bottom of the screen.


Default messaging provider > JMS queue connection factory > New

A JMS queue connection factory is used to create connections to the associat queues, for point-to-point messaging. Use queue connection factory adminis queue connection factories for the default messaging provider.

* Scope: cells:esbCell:nodes:esbNode

* Name: sampleBindingQCF

* JNDI name: jms/sampleBindingQCF

Description:

Category:

* Bus name: SCA.APPLICATION.esbCell.Bus

Figure 11-62 Create new queue connection factory

46.n the navigation menu on the left, select **Resources** → **JMS Providers** → **Default messaging**.

47.On the right, under Destinations, click **JMS queue** and then click **New**.

48.Create the queue.

a. Set the Name to `JMSImportOut.`

b. Set the JNDI Name to `jms/JMSImportOut`.

c. Select **SCA.APPLICATION.esbCell.Bus** from the Bus Name drop down menu.

d. Select **JMSImportOut** from the Queue Name drop down menu.

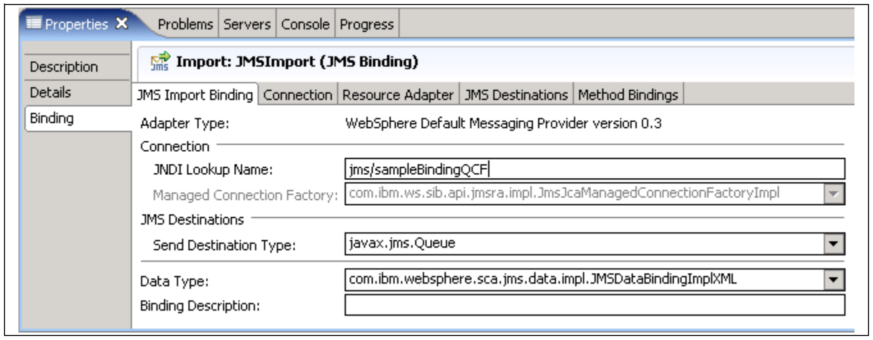
e. Click **OK** at the bottom of the screen

49.Click the **Save** link at the top of the console and confirm the changes to the master configuration by clicking the **Save** button.

50.Log out of the administrative console.

51.Switch back to the assembly diagram of the MapBindingSample1Module module. Select the **JMSImport** import. In the Properties view, select the **Binding** tab.

52. In the JNDI Lookup Name text field for the Connection enter `jms/sampleBindingQCF` (Figure 11-63).



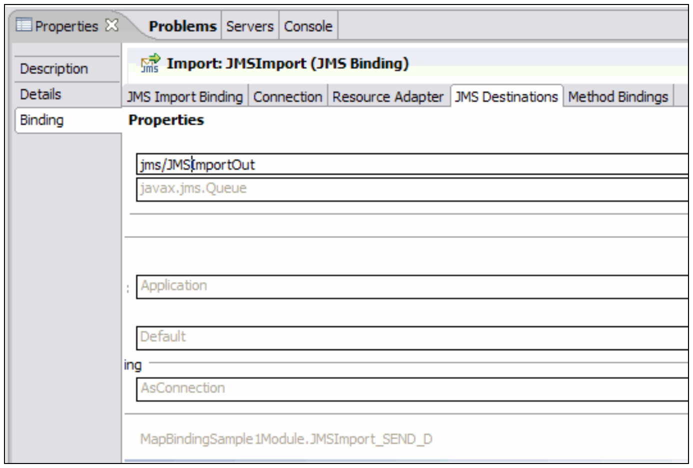
Import: JMSImport (JMS Binding)

Adapter Type: WebSphere Default Messaging Provider version 0.3
Connection
JNDI Lookup Name: jms/sampleBindingQCF
Managed Connection Factory: com.ibm.ws.sib.api.jmsra.impl.JmsJcaManagedConnectionFactoryImpl
JMS Destinations
Send Destination Type: javax.jms.Queue
Data Type: com.ibm.websphere.sca.jms.data.impl.JMSDataBindingImplXML
Binding Description:

Figure 11-63 Queue connection factory JNDI name

53. Still in the Properties view, select the **JMS Destinations** tab and expand the **Send Destination Properties**.

54. In the JNDI Lookup Name text field enter `jms/JMSImportOut` (Figure 11-64).



Import: JMSImport (JMS Binding)

Properties
jms/JMSImportOut
javax.jms.Queue
Application
Default
ing
AsConnection
MapBindingSample1Module.JMSImport_SEND_D

Figure 11-64 Queue JNDI name

55. Save the module.

56. Deploy the module to the server.

   a. Switch to the Servers view.

   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.

   c. Add MapBindingSample1ModuleApp.

d. Click **Finish**.

57. Right-click the **MapBindingSample1Module** in the Business Integration view and select **Test** → **Test Module**.

58. Right click the **Invoke** event in the Events view and click **Remove**.

59. Click the Attach icon in the test view to create an Attach, then click **Continue** (Figure 11-65).

60. Select your WebSphere Enterprise Service Bus server in the Deployment Location window and click **Finish**.

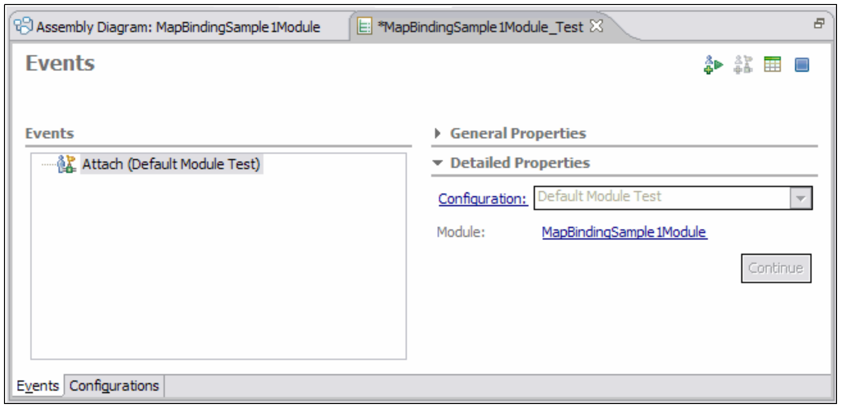
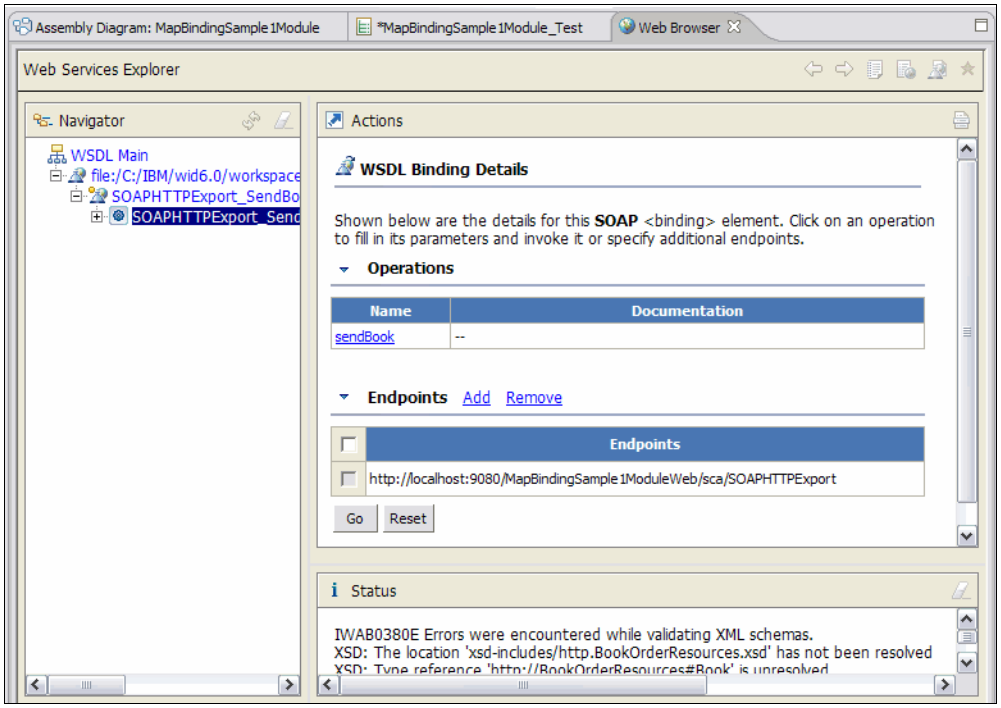


Figure 11-65 Attached test client

61. Open the Physical Resources view by right-clicking on the **MapBindingSample1Module** module and selecting **Show Files**.

62. Right-click the **SOAPHTTPExport_SendBookHttp_Service.wsdl** file in the MapBindingsSample1Module folder and select **Web Services** → **Test with Web Services Explorer**. The Web Service Explorer opens in a Web Browser view (Figure 11-66 on page 347).



WSDL Binding Details

Shown below are the details for this **SOAP** <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
sendBook	--

Endpoints Add Remove

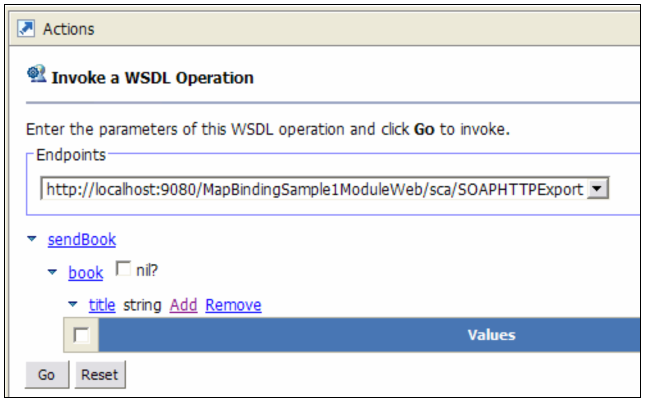
	Endpoints
☐	http://localhost:9080/MapBindingSample1ModuleWeb/sca/SOAPHTTPExport

IWAB0380E Errors were encountered while validating XML schemas.
XSD: The location 'xsd-includes/http.BookOrderResources.xsd' has not been resolved
XSD: Type reference 'http://BookOrderResources#Book' is unresolved

Figure 11-66 Web service explorer start view

63. Click the **sendBook** operation link.

64. Click the **Add** link next to title (Figure 11-67).



Invoke a WSDL Operation

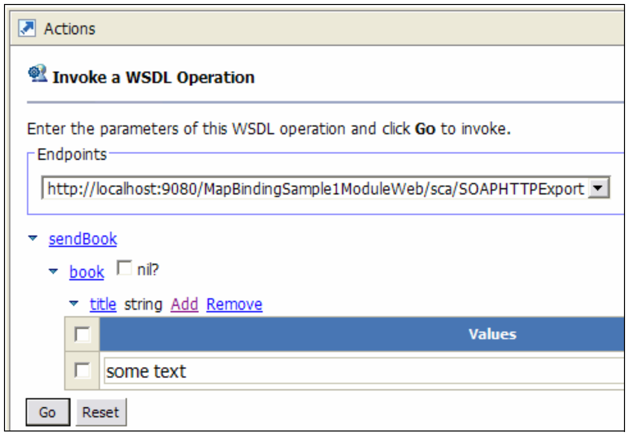
Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints

http://localhost:9080/MapBindingSample1ModuleWeb/sca/SOAPHTTPExport

Figure 11-67 Add input parameter in the Web service explorer

65. Enter any text in the Values text box and click **Go** (Figure 11-68 on page 348).


Actions

Invoke a WSDL Operation

Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints: http://localhost:9080/MapBindingSample1ModuleWeb/sca/SOAPHTTPExport

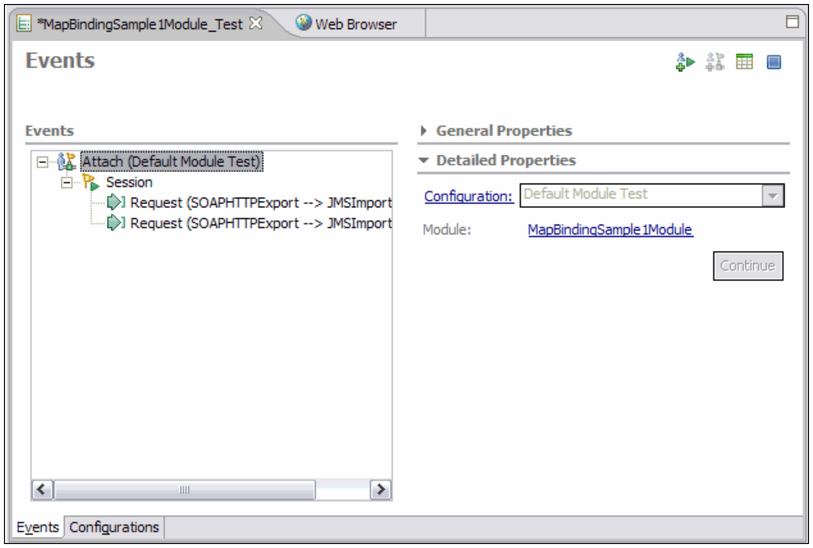
sendBook
book ☐ nil?
title string Add Remove

☐	Values
☐	some text

Go Reset

Figure 11-68 Web service explorer test settings

66. Switch to the Test view. You should see the request sent from the export to the import (Figure 11-69).


*MapBindingSample1Module_Test | Web Browser

Events

Events
- Attach (Default Module Test)
  - Session
    - Request (SOAPHTTPExport --> JMSImport
    - Request (SOAPHTTPExport --> JMSImport

General Properties

Detailed Properties

Configuration: Default Module Test

Module: MapBindingSample1Module

Continue

Events | Configurations

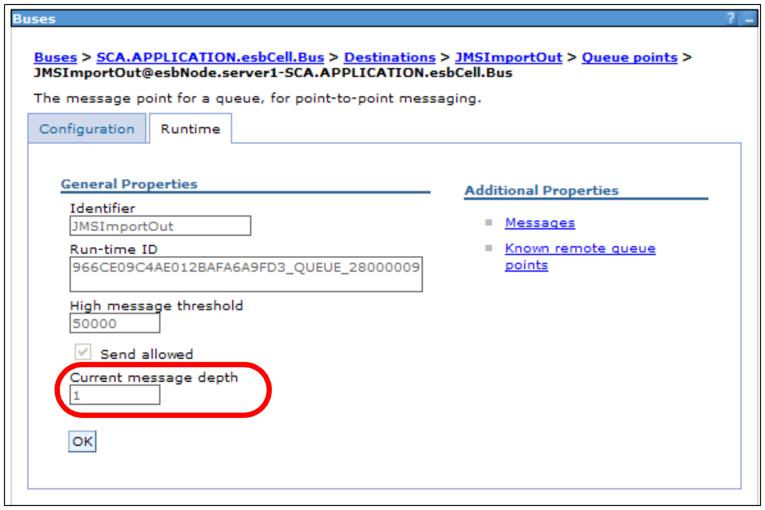
Figure 11-69 Test result

Attention: Although the request is only sent once from the export to the import, the attached test tool displays it twice.

67. In the Servers view, right-click your WebSphere Enterprise Service Bus server and select **Run administrative console** and log in.

68. Select **Service integration** → **Buses** → **SCA.APPLICATION.esbCell.Bus** → **Destinations** → **JMSImportOut** → **Queue points** → **JMSImportOut@esbNode.server1-SCA.APPLICATION.esbCell.Bus**.

69. Click on the **Runtime** tab. Current message depth should show 1 (Figure 11-70).



Buses

Buses > SCA.APPLICATION.esbCell.Bus > Destinations > JMSImportOut > Queue points > JMSImportOut@esbNode.server1-SCA.APPLICATION.esbCell.Bus

The message point for a queue, for point-to-point messaging.

Configuration | Runtime

General Properties

Identifier: JMSImportOut

Run-time ID: 966CE09C4AE012BAFA6A9FD3_QUEUE_28000009

High message threshold: 50000

Send allowed: ✓

Current message depth: 1

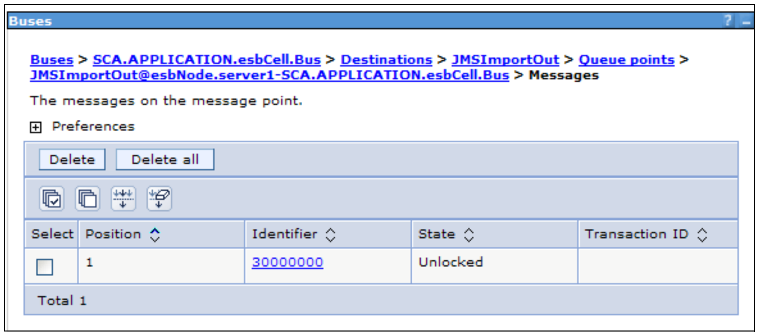
OK

Additional Properties

- Messages
- Known remote queue points

Figure 11-70 Output queue properties

70. Click on the **Messages** link under Additional Properties (Figure 11-71).



Buses

Buses > SCA.APPLICATION.esbCell.Bus > Destinations > JMSImportOut > Queue points > JMSImportOut@esbNode.server1-SCA.APPLICATION.esbCell.Bus > Messages

The messages on the message point.

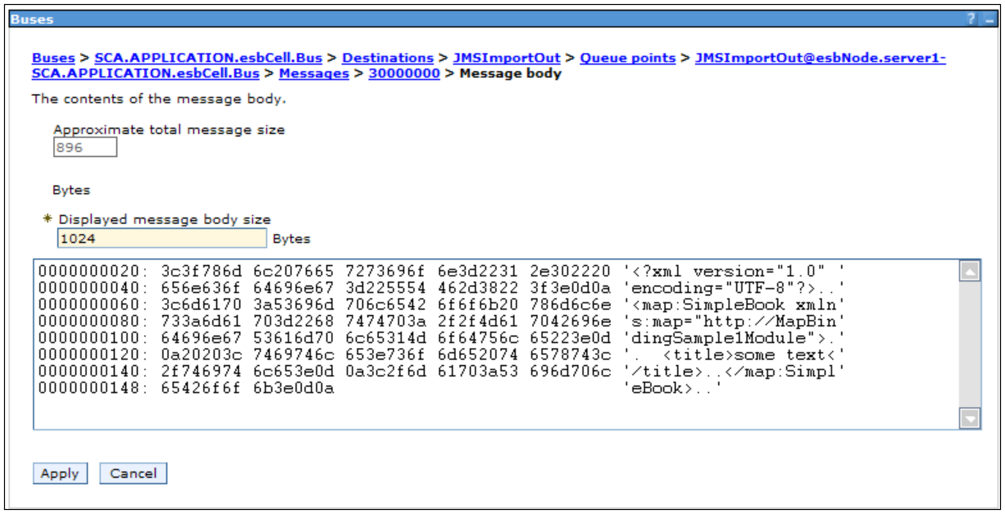
⊞ Preferences

Delete | Delete all

Select	Position	Identifier	State	Transaction ID
☐	1	30000000	Unlocked	
Total 1				

Figure 11-71 Output queue content

71. Click on the message identifier link. In the next screen click on the **Message body** link. You should see the serialized business object containing your text (Figure 11-72 on page 350).



Buses > SCA.APPLICATION.esbCell.Bus > Destinations > JMSImportOut > Queue points > JMSImportOut@esbNode.server1-SCA.APPLICATION.esbCell.Bus > Messages > 30000000 > Message body

The contents of the message body.

Approximate total message size
896
Bytes

* Displayed message body size
1024 Bytes

```
0000000020: 3c3f786d 6c207665 7273696f 6e3d2231 2e302220 '<?xml version="1.0" '
0000000040: 656e636f 64696e67 3d225554 462d3822 3f3e0d0a 'encoding="UTF-8"?>..'
0000000060: 3c6d6170 3a53696d 706c6542 6f6f6b20 786d6c6e '<map:SimpleBook xmln'
0000000080: 733a6d61 703d2268 7474703a 2f2f4d61 7042696e 's:map="http://MapBin'
0000000100: 64696e67 53616d70 6c65314d 6f64756c 65223e0d 'dingSample1Module">.'
0000000120: 0a20203c 7469746c 653e736f 6d652074 6578743c '.  <title>some text<'
0000000140: 2f746974 6c653e0d 0a3c2f6d 61703a53 696d706c '/title>..</map:Simpl'
0000000148: 65426f6f 6b3e0d0a                            'eBook>..'
```

Figure 11-72 Message content

72. Log out of the administrative console.
73. Switch to the test view. Right-click the **Attach** event and select **Detach**.
74. Remove the project from the server.

Congratulations you have successfully demonstrated how to build a module that converts a request message to use a different transport protocol.

11.3.2 Request and response flows

In this sample, we will create a module to demonstrate a request and response flow. The flows are implemented using mediation primitives.

The request flow is used to handle service request messages. In this example the request flow will store information about the order in the correlation context.

The response flow is used to handle service response messages. In this sample the response flow will log the order and confirmation ID.

This sample involves:

- ► Building a mediation module containing an import with a Web service binding.
- ► Implementing the request flow using an XSL Transform mediation.
- ► Implementing the response flow using a Message Logger mediation.

The completed sample invokes a book order Web service, which returns a confirmation ID. The mediation module is used to log the book order and the corresponding confirmation ID to a database.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `ReqResSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Rename Mediation1 to `BookOrderMediation`.
4. In the BookOrderResources project expand **Web Service Ports**. Drag and drop **BookOrderServiceSOAP** onto the assembly diagram. The Component Creation dialog will open (Figure 11-73).

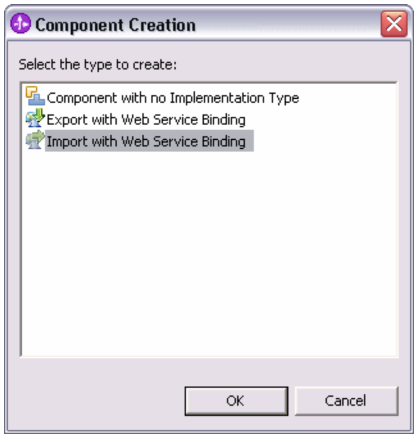

Component Creation
Select the type to create:
Component with no Implementation Type
Export with Web Service Binding
Import with Web Service Binding
OK
Cancel

Figure 11-73 Component creation

5. Select **Import with Web Service Binding** and click **OK**.
6. Rename Import1 to `BookOrderServiceImport`.
7. Right-click BookOrderMediation and select **Add -> Interface**.
8. Select the **BookOrderService** interface and click **OK**.
9. Wire BookOrderMediation to BookOrderServiceImport (Figure 11-74 on page 352). When asked whether to add a reference to the mediation flow component click **Yes**.

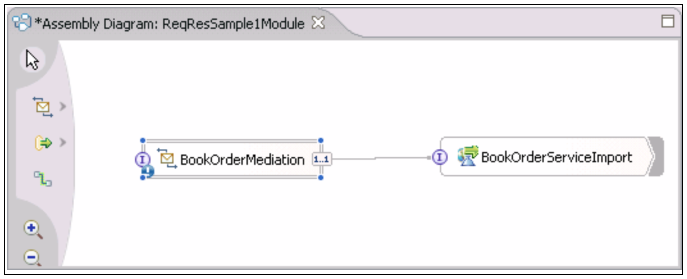


Figure 11-74 ReqResSample1Module

Now we are going to implement the BookOrderMediation flow component.

10.Right-click on **BookOrderMediation** and select **Generate Implementation**.

11.In the Generate Implementation dialog click **OK** to create the implementation in the default location.

The Mediation Flow Editor displays two panels, the first one is used to map operations in the interface to operations in the reference. The second panel is used to implement the request and response flows.

12.In the Operation connections panel wire the **order** operation on the BookOrderService interface to the **order** operation on the BookOrderServicePartner reference (Figure 11-75).

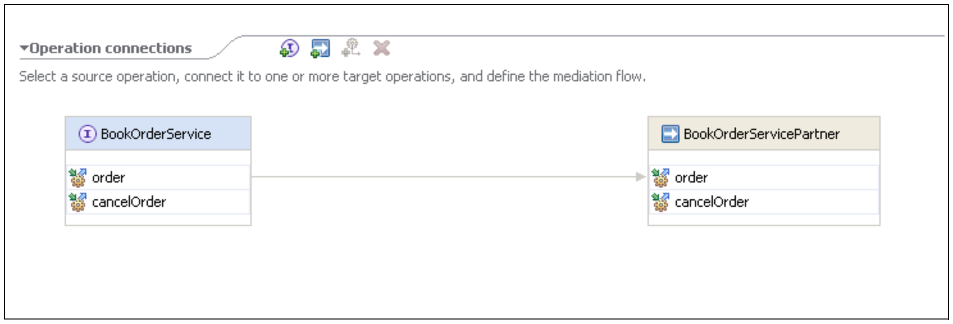
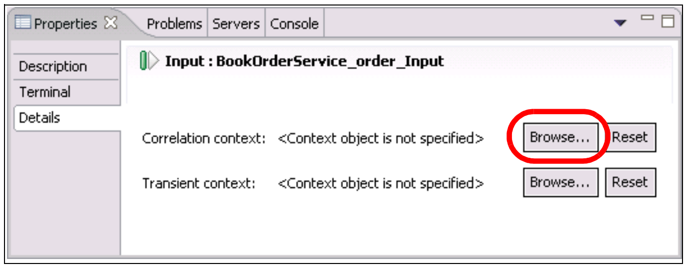

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
BookOrderService
order
cancelOrder
BookOrderServicePartner
order
cancelOrder

Figure 11-75 Operation connections

The request flow will use the correlation context to store the book order from the request message. Using the correlation context makes the book order available to the response flow.

13.Click on **BookOrderService_order_Input**, and click the **Details** tab in the Properties view.

14. Next to Correlation context click the **Browse** button (Figure 11-76 on page 353).



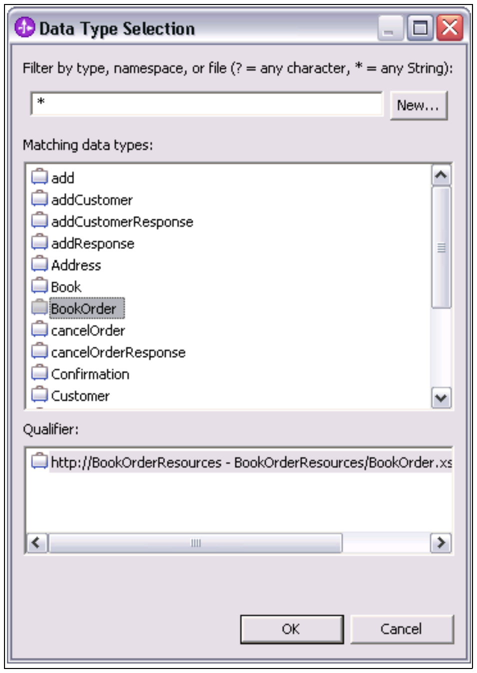
Correlation context: <Context object is not specified>

Transient context: <Context object is not specified>

Figure 11-76 Correlation context browse

Now we need to choose what type of business object we will store in the correlation context.

15. In the Data Type Selection dialog choose the **BookOrder** data type and click **OK** (Figure 11-77).



Filter by type, namespace, or file (? = any character, * = any String):

*

http://BookOrderResources - BookOrderResources/BookOrder.xs

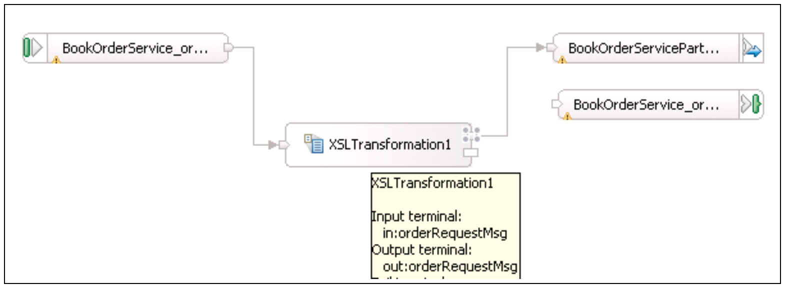
Figure 11-77 Data type selection

16. Add an XSLTransform mediation primitive to the request flow by using .

17. Wire the request flow (Figure 11-78 on page 354).

a. Wire **BookOrderService_order_Input** to the **in** terminal of XSLTransform1.

b. Wire the **out** terminal of XSLTransform1 **BookOrderServicePartner_order_Callout**.



XSLTransformation1

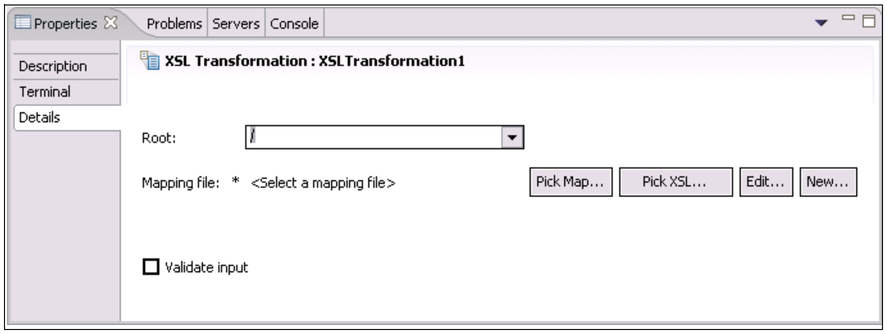
Input terminal:
 in:orderRequestMsg
Output terminal:
 out:orderRequestMsg

Figure 11-78 Request flow

18. Click on XSLTransform1 and click the **Details** tab in the Properties view.

19. In the Root drop down menu select **/** (Figure 11-79).

20. Click the **New** button.



XSL Transformation : XSLTransformation1

Root: /

Mapping file: * <Select a mapping file>

☐ Validate input

Figure 11-79 Creating an new mapping

21. In the New XSLT Mapping dialog click **Finish**.

22. In the XML Map Editor create the mapping (Figure 11-80 on page 355).

a. Expand the source and target **tns:smo** tree.

b. Select **headers** in both the source and target panels.

c. Right-click on **headers** in the source panel and select **Match Mapping**.

d. Select **body** in both the source and target panels.

e. Right-click on **body** in the source panel and select **Match Mapping**.

f. In the source panel select **body -> order -> bookOrder -> book**.

g. In the target panel select **context -> correlation -> book**.

h. Right-click on **book** in the source panel and select **Match Mapping**.

i. In the source panel select **body -> order -> bookOrder -> quantity.**

j. In the target panel select **context -> correlation -> quantity.**

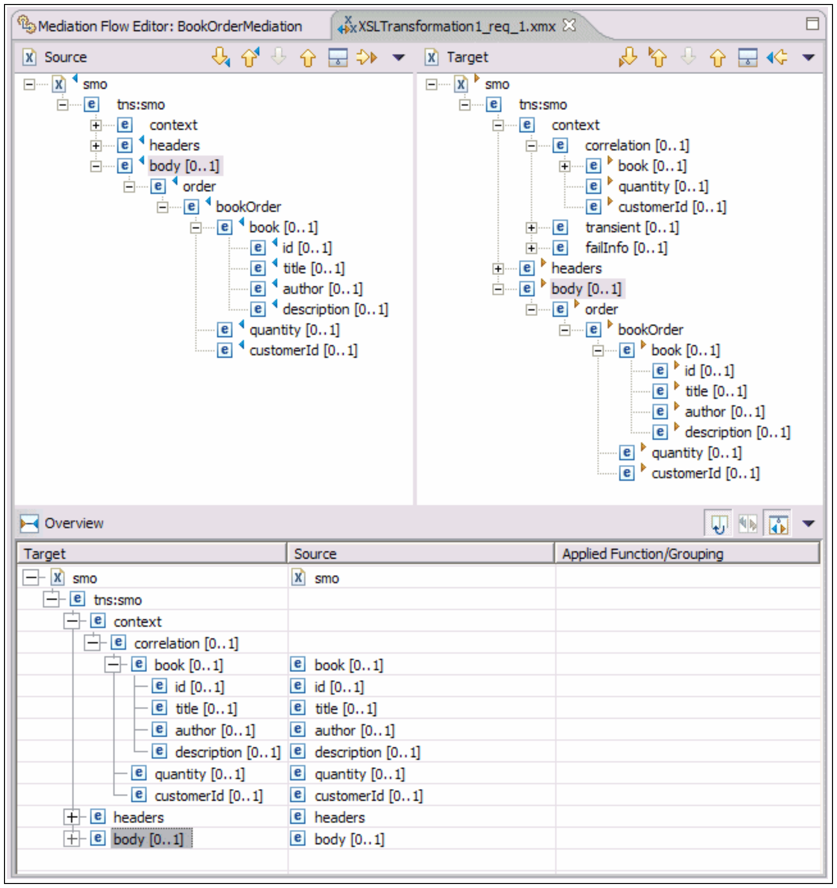
k. Right-click on **quantity** in the source panel and select **Create Mapping.**

l. In the source panel select **body -> order -> bookOrder -> customerId.**

m. In the target panel select **context -> correlation -> customerId.**

n. Right-click on **customerId** in the source panel and select **Create Mapping.**

o. Save and close the mapping file.



Target	Source	Applied Function/Grouping
smo	smo	
tns:smo		
context		
correlation [0..1]		
book [0..1]	book [0..1]	
id [0..1]	id [0..1]	
title [0..1]	title [0..1]	
author [0..1]	author [0..1]	
description [0..1]	description [0..1]	
quantity [0..1]	quantity [0..1]	
customerId [0..1]	customerId [0..1]	
headers	headers	
body [0..1]	body [0..1]	

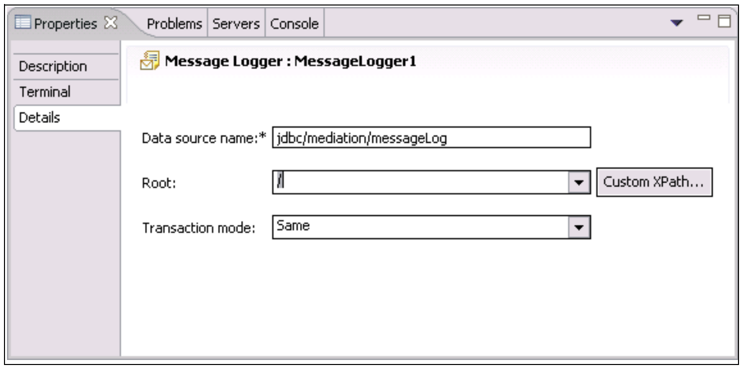
Figure 11-80 Mapping book order to the context

23. In the Properties view click the **Regenerate XSL** button.
24. When the XSL regenerated dialog appears click **OK**.

The mapping file we created mapped the headers and body of the incoming message directly to headers and body of the outgoing message. We also mapped the contents of the book order in the incoming message to the correlation context.

Now we create the response flow that will retrieve the book order object from the correlation context and log it with the confirmation id from the response message.

25. In the Mediation Flow Editor, click on the **Response** tab.
26. Add a Message Logger mediation primitive to the response flow using .
27. Click MessageLogger1 and select the **Details** tab in the Properties view.
28. From the Root drop down menu select **/** (Figure 11-81).



Message Logger : MessageLogger1

Data source name:* jdbc/mediation/messageLog

Root: /

Transaction mode: Same

Figure 11-81 Message logger properties

29. Wire the response flow (Figure 11-82).
    a. Wire **BookOrderServicePartner_order_CalloutResponse** to the **in** terminal of MessageLogger1.
    b. Wire the **out** terminal of MessageLogger1 to **BookOrderService_order_InputResponse**.

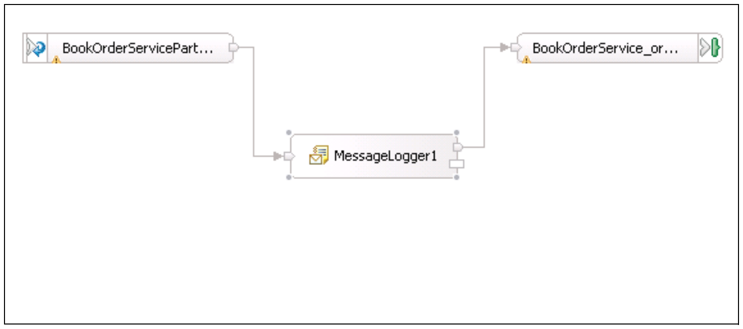
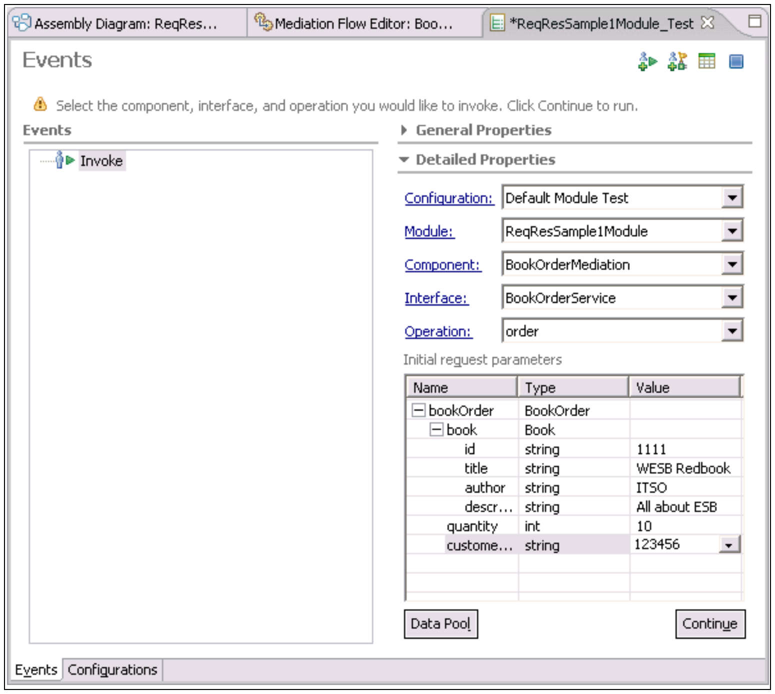


Figure 11-82 Response flow

30. Save the mediation flow and the module.

The module is now complete. The Integration Test Client is used to test the module.

31. Deploy the module and the Web service to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add BookOrderServiceEAR and ReqResSample1ModuleApp.
    d. Click **Finish**.
32. Right-click on **ReqResSample1Module** and select **Test -> Test Module**.
33. Ensure **BookOrderMediation** is selected in the Component drop down menu.
34. Select **order** from the Operation drop down menu.
35. Enter values for the BookOrder and click **Continue** (Figure 11-83).


Assembly Diagram: ReqRes... | Mediation Flow Editor: Boo... | *ReqResSample1Module_Test

Events

Select the component, interface, and operation you would like to invoke. Click Continue to run.

Events: Invoke

General Properties

Detailed Properties

Configuration: Default Module Test
Module: ReqResSample1Module
Component: BookOrderMediation
Interface: BookOrderService
Operation: order

Initial request parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	1111
title	string	WESB Redbook
author	string	ITSO
descr...	string	All about ESB
quantity	int	10
custome...	string	123456

Data Pool | Continue

Events | Configurations

Figure 11-83 Integration test client

36. In the Deployment Location dialog select the WebSphere Enterprise Service Bus server and click **Finish**.

37. A confirmation ID will be returned and displayed

The module has been tested, now lets check the logging database for the message.

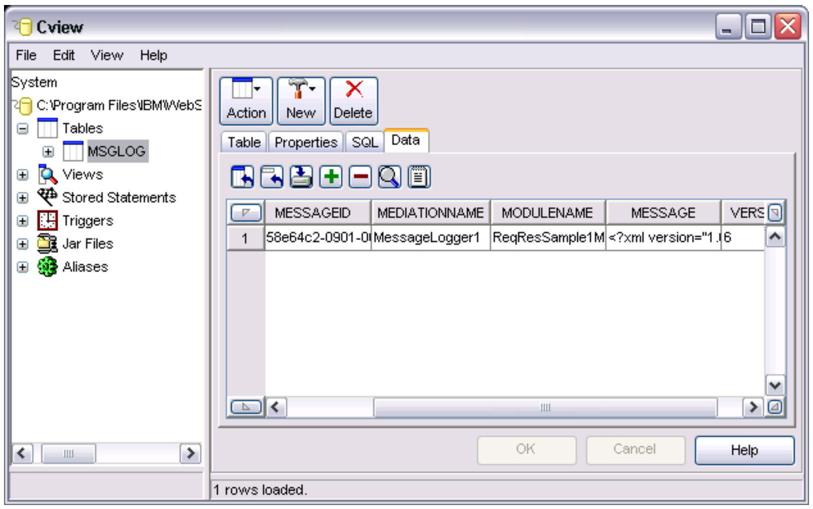
38. In the Servers view right click the WebSphere Enterprise Service Bus server and click **Stop**. This unlocks the logging database so we can open it for viewing.

39. Run the Cloudscape viewer **cview.bat** which is available in the <WID_INSTALL>\runtimes\bi_v6\cloudscape\bin\embedded directory.

40. Click on **File -> Open** and open the CloudScape database **EsbLogMedDB** which is in the directory <WID_INSTALL>\pf\esb\databases.

41. Expand **Tables** and select **MSGLOG.**

42. Click on **Data** tab to show the records in the table (Figure 11-84).

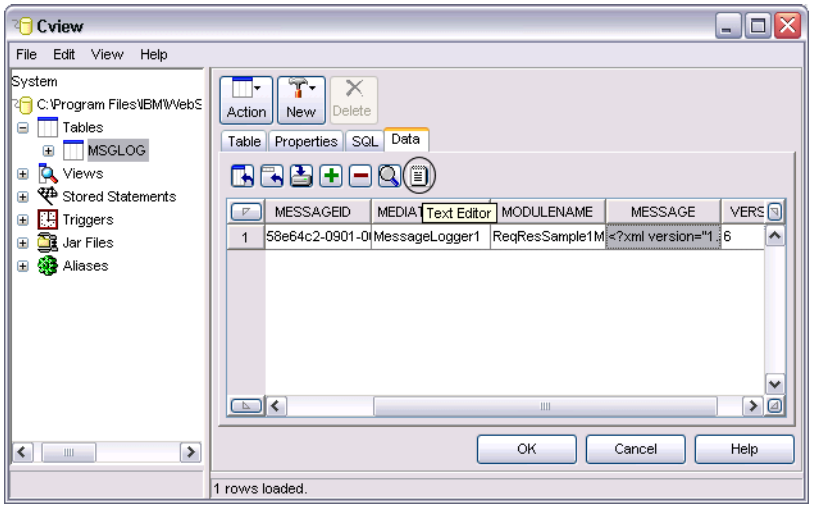


	MESSAGEID	MEDIATIONNAME	MODULENAME	MESSAGE	VERS
1	58e64c2-0901-0	MessageLogger1	ReqResSample1M	<?xml version="1.	6

1 rows loaded.

Figure 11-84 CloudScape database MSGLOG table

43. Select the message in the **MESSAGE** column, and click on the **Text Editor** icon (Figure 11-85).

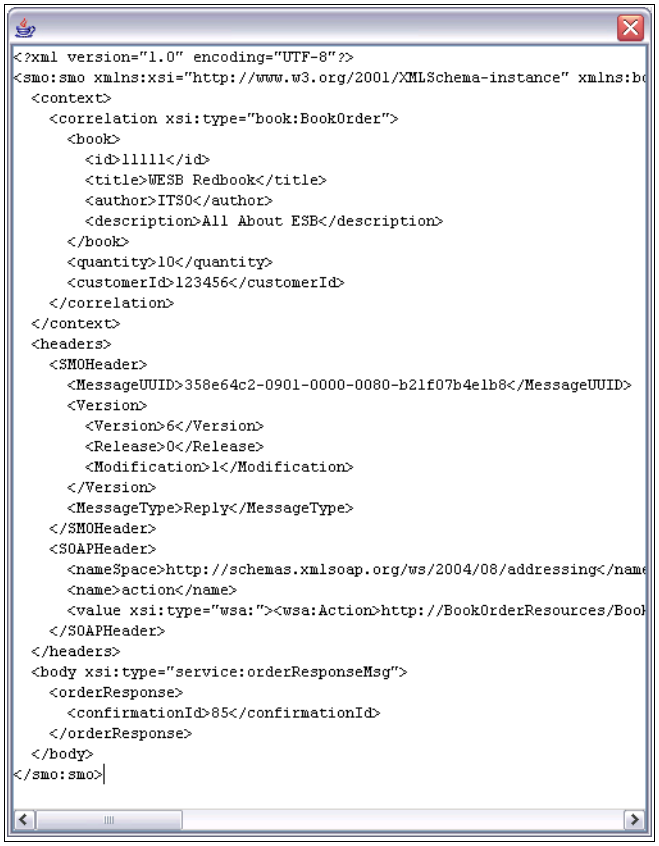


	MESSAGEID	MEDIAT	MODULENAME	MESSAGE	VERS
1	58e64c2-0901-0	MessageLogger1	ReqResSample1M	<?xml version="1.	6

1 rows loaded.

Figure 11-85 Text Editor to view message

44. You should see the context of the Service Message Object contains the BookOrder and the body contains the confirmationId (Figure 11-86).



```
<?xml version="1.0" encoding="UTF-8"?>
<smo:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:bo
  <context>
    <correlation xsi:type="book:BookOrder">
      <book>
        <id>11111</id>
        <title>WESB Redbook</title>
        <author>ITSO</author>
        <description>All About ESB</description>
      </book>
      <quantity>10</quantity>
      <customerId>123456</customerId>
    </correlation>
  </context>
  <headers>
    <SMOHeader>
      <MessageUUID>358e64c2-0901-0000-0080-b21f07b4e1b8</MessageUUID>
      <Version>
        <Version>6</Version>
        <Release>0</Release>
        <Modification>1</Modification>
      </Version>
      <MessageType>Reply</MessageType>
    </SMOHeader>
    <SOAPHeader>
      <nameSpace>http://schemas.xmlsoap.org/ws/2004/08/addressing</name
      <name>action</name>
      <value xsi:type="wsa:"><wsa:Action>http://BookOrderResources/Book
    </SOAPHeader>
  </headers>
  <body xsi:type="service:orderResponseMsg">
    <orderResponse>
      <confirmationId>85</confirmationId>
    </orderResponse>
  </body>
</smo:smo>
```

Figure 11-86 Message logged in the database

Congratulations, you have successfully created a module that demonstrates a request and a response flow.

45. Remove the projects from the test server and close Cview.

11.3.3 Fault handling

This sample demonstrates how to handle faults in a mediation flow.

A fault occurs when an exception is raised and thrown in a service. We need two basic elements to illustrate fault handling:

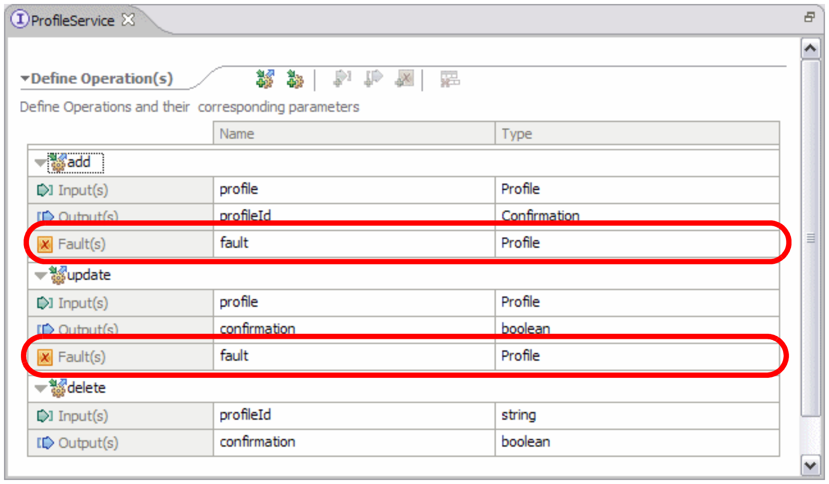
1. An interface that includes a fault as part of its definition
2. A component that raises an exception of the kind defined by the fault.

This sample involves:

- Building a mediation module containing an import with a Web service binding.
- Implementing a mediation flow component to handle fault messages.
- Modifying a Web service to throw an exception, creating a fault message.

The completed sample demonstrates an attempt to call a service that creates a new profile. The request fails and the fault condition is handled by the mediation module.

1. First expand **BookOrderResources**, and **Interfaces** and open the **ProfileService** interface by double-clicking on it. Notice that both the add and update operations define a fault as part of the interface, as shown in Figure 11-87. Thus, our first requirement is met, and we will be using this interface and the ProfileService Web service in the remainder of this sample.



	Name	Type
add		
Input(s)	profile	Profile
Output(s)	profileId	Confirmation
Fault(s)	fault	Profile
update		
Input(s)	profile	Profile
Output(s)	confirmation	boolean
Fault(s)	fault	Profile
delete		
Input(s)	profileId	string
Output(s)	confirmation	boolean

Figure 11-87 Profile service interface includes fault definitions

2. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `FaultSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
3. Open the module in the Assembly Editor by double clicking .
4. Rename Mediation1 to `HandleFaultMediation`, as shown in Figure 11-88.

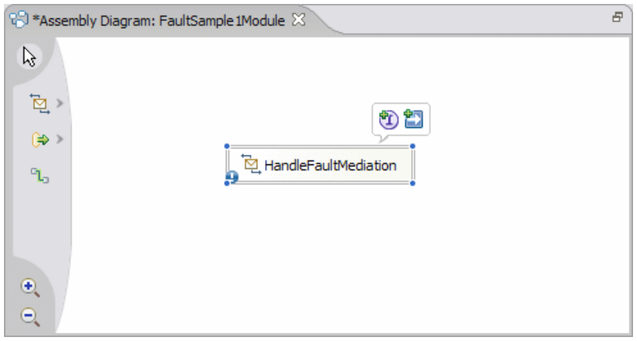
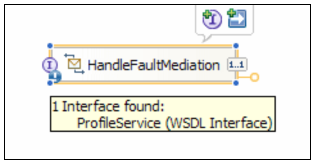


Figure 11-88 HandleFaultMediation

5. Right click **HandleFaultMediation** and select **Add -> Interface**.
6. Choose the **ProfileService** interface and click **OK**.
7. Right click **HandleFaultMediation** and select **Add -> Reference**.
8. Choose the **ProfileService** interface and click **OK**. The HandleFaultMediation should now appear as shown in Figure 11-89 on page 362.



1 Interface found:
ProfileService (WSDL Interface)

Figure 11-89 HandleFaultMediation component

9. Save the module.

Now we have a mediation flow component with an interface and a reference lets generate the mediation flow.

1. Right click **HandleFaultMediation** and select **Generate Implementation**.
2. Click **OK** to store the implementation in the default folder. This opens the Mediation Flow editor.
3. In the Operation connections section of this view, wire the **add** method on the ProfileService Interface to the **add** method of the ProfileServicePartner Reference (Figure 11-90).

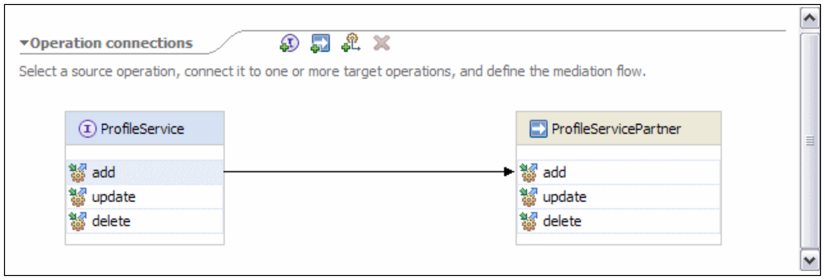

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
ProfileService
add
update
delete
ProfileServicePartner
add
update
delete

Figure 11-90 Operation connection

4. In the request flow, wire ProfileService_add_Input to the input terminal of ProfileServicePartner_add_Callout, as shown in Figure 11-91 on page 363.

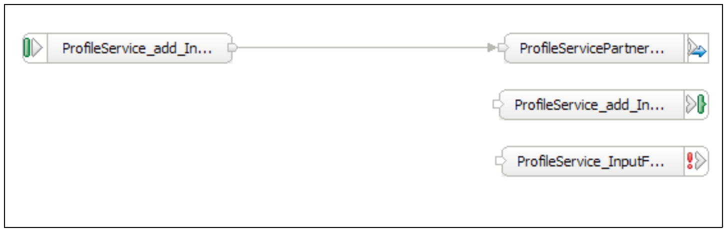


Figure 11-91 Request flow wiring

5. Click the **Response** tab, and in the response flow wire ProfileServicePartner_add_CalloutResponse to the input terminal of ProfileService_add_InputResponse.
6. Wire ProfileServicePartner_CalloutFault to the input terminal of ProfileService_InputFault. Your response flow should look like Figure 11-92

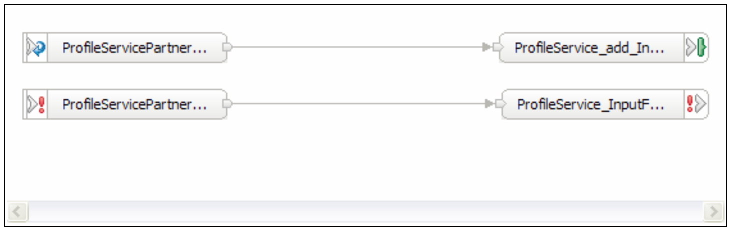
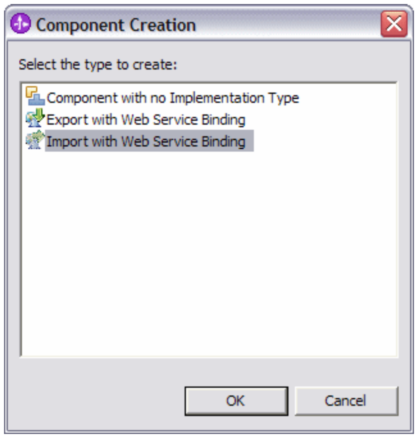


Figure 11-92 Response flow wiring

7. Save the mediation flow.
8. Expand the **BookOrderResources** library in the Business Integration view, and the **Web Service Ports**.

9. Drag and drop the **ProfileServiceSOAP** from the Business Integration view into the assembly diagram of the FaultSample1Module module.
10. This will open the Component Creation window. Select **Import with Web Service Binding** and click **OK**.


Component Creation

Select the type to create:

Component with no Implementation Type
Export with Web Service Binding
Import with Web Service Binding

Figure 11-93 Component creation selection

11. Select the newly created import **Import1** in the assembly diagram and rename it to `ProfileServiceImport`.
12. Wire the WSDL reference from the HandleFaultMediation to the interface on ProfileServiceImport. Your assembly diagram should match Figure 11-94.

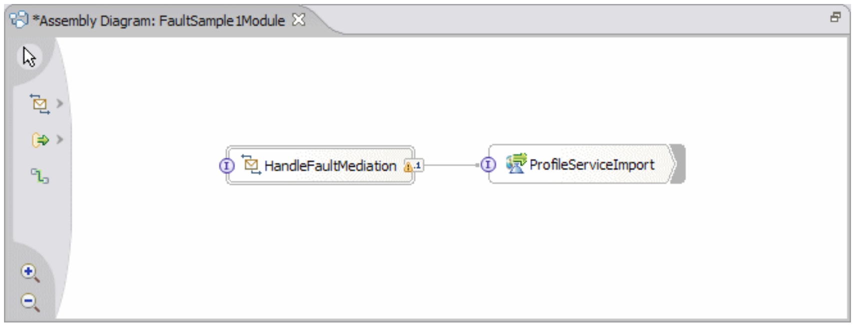
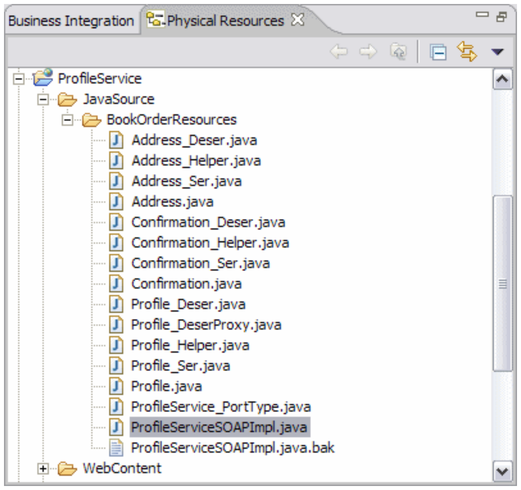


Figure 11-94 FaultSample1Module assembly diagram

13. Save the mediation module.
14. Open the Physical Resources view in the Business Integration perspective by right-clicking on any module and selecting **Show Files**.

15. Expand **ProfileService**, then **JavaSource**, then **BookOrderResources**, as shown in Figure 11-95 on page 365.



```
ProfileService
  JavaSource
    BookOrderResources
      Address_Deser.java
      Address_Helper.java
      Address_Ser.java
      Address.java
      Confirmation_Deser.java
      Confirmation_Helper.java
      Confirmation_Ser.java
      Confirmation.java
      Profile_Deser.java
      Profile_DeserProxy.java
      Profile_Helper.java
      Profile_Ser.java
      Profile.java
      ProfileService_PortType.java
      ProfileServiceSOAPImpl.java
      ProfileServiceSOAPImpl.java.bak
  WebContent
```

Figure 11-95 Select the web service java implementation

16. Open **ProfileServiceSOAPImpl.java**, by double-clicking on it.

17. Look at the code, and notice the add method throws two exceptions. The one we are interested in is the BookOrderResources.Profile exception.

18. At the bottom of the add method, comment out the following line of code:

```
return confirmation;
```

19. Add the following line of code just before the return statement:

```
throw profile;
```

20. Make sure the code matches that in Example 11-4.

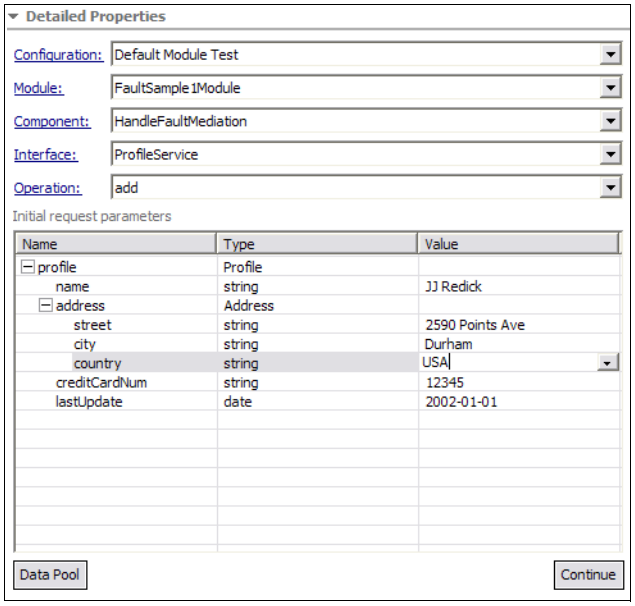
Example 11-4 add method of ProfileServiceSOAPImpl

```
public BookOrderResources.Confirmation add(BookOrderResources.Profile profile)
throws java.rmi.RemoteException, BookOrderResources.Profile {
    String id = Integer.toString(new Random().nextInt(100));
        while (profiles.containsKey(id)) {
        id = Integer.toString(new Random().nextInt(100));
        }
        profiles.put(id, profile);
        Confirmation confirmation = new Confirmation();
        confirmation.setId(id);
        throw profile;
        //return confirmation;
    }
```

21. Save the updated Java class.

We are now ready to test the mediation module. We use the Integration Test Client to perform the test. Follow these steps:

1. Start your WebSphere Enterprise Service Bus server.
2. Deploy the module and the Web service to the server.
   a. Switch to the Servers view.
   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
   c. Add ProfileServiceEAR and FaultSample1ModuleApp.
   d. Click **Finish**.
3. In the Business Integration view, right click **FaultSample1Module** and select **Test -> Test Module**.
4. Set the Component to **HandleFaultMediation**.
5. Select the **add** operation from the Operation drop down menu.
6. Enter any test data that you like as shown in Figure 11-96.



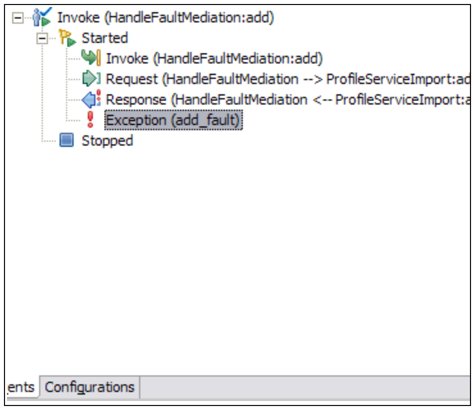
Configuration:	Default Module Test
Module:	FaultSample1Module
Component:	HandleFaultMediation
Interface:	ProfileService
Operation:	add

Initial request parameters

Name	Type	Value
profile	Profile	
name	string	JJ Redick
address	Address	
street	string	2590 Points Ave
city	string	Durham
country	string	USA
creditCardNum	string	12345
lastUpdate	date	2002-01-01

Figure 11-96 Test input

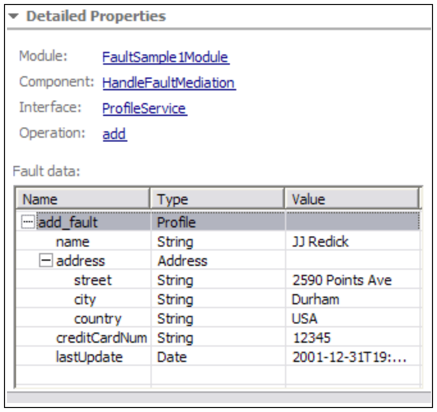
7. Click **Continue**.
8. Select your WebSphere Enterprise Service Bus server and click **Finish** in the Deployment Location dialog
9. You will see the mediation flow returns the add_fault as shown in Figure 11-97.



```
Invoke (HandleFaultMediation:add)
  Started
    Invoke (HandleFaultMediation:add)
    Request (HandleFaultMediation --> ProfileServiceImport:ad
    Response (HandleFaultMediation <-- ProfileServiceImport:a
    Exception (add_fault)
  Stopped
```

Figure 11-97 Fault returned

10. Notice the fault data returned is identical to the input data that was supplied, as shown in Figure 11-98 on page 367.



Detailed Properties

Module: FaultSample1Module
Component: HandleFaultMediation
Interface: ProfileService
Operation: add

Fault data:

Name	Type	Value
add_fault	Profile	
name	String	JJ Redick
address	Address	
street	String	2590 Points Ave
city	String	Durham
country	String	USA
creditCardNum	String	12345
lastUpdate	Date	2001-12-31T19:...

Figure 11-98 Fault data

Congratulations you have successfully built and tested a mediation module that properly handles a fault condition from a Web service.

11. Remove the projects from the test server.

Note: In this sample when we modified the Web service to throw the exception, we took a very simple approach. More commonly the logic in the Web service would be such that it would try the add operation, and if it failed, the exception would be thrown.

Note: In this sample the fault data was not very descriptive as to the cause of the error. More commonly another business object would be built to contain detailed information about the fault condition, and would be used in the interface definition.

12

Developing mediation logic using mediation primitives

This chapter provides step-by-step instructions for using each of the mediation primitives provided by WebSphere Enterprise Service Bus. These are:

- XSL Transformation mediation primitive
- Database Lookup mediation primitive
- Message Filter mediation primitive
- Message Logger mediation primitive
- Stop mediation primitive
- Fail mediation primitive
- Custom mediation primitive

These development examples assume you have configured your WebSphere Integration Developer workspace as described in Chapter 10, “Preparing for the development examples” on page 271.

You may find it useful to refer to Chapter 7, “WebSphere Integration Developer key concepts and common tasks” on page 153 for more detailed information on how to perform specific tasks in WebSphere Integration Developer.

You can import completed Project Interchange projects for each mediation primitive development example from the additional material supplied with this redbook in the `\MediationPrimitives\Solutions` directory.

12.1 XSL Transformation mediation primitive

This sample demonstrates how to use an XSL Transformation mediation primitive to map messages between two incompatible interfaces.

The XSL Transformation mediation primitive uses XSL stylesheets to transform the Service Message Object received on its input terminal. The transformed message is then output to the output terminal.

This sample involves:

- Creating a mediation module to communicate with a Web service.
- Implementing a mediation flow component using an XSL Transformation mediation primitive.
- Testing the module using the Integration Test Client.

The completed sample demonstrates a client making a request to create a new customer profile. The mediation module converts this request to the required format and invokes a Web service that creates a profile. The client receives confirmation that a customer profile has been created.

Perform the following:

1. Create a new Mediation Module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `XSLTSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog tick the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double-clicking .
3. Rename the component Mediation1 to `CustomerMediation` (Figure 12-1).

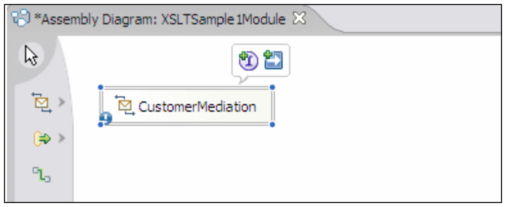


Figure 12-1 CustomerMediation component

4. Add the CustomerService interface to CustomerMediation.
   a. Right-click **CustomerMediation** and select **Add -> Interface**.

b. Select the **CustomerService** interface and click **OK** (Figure 12-2).

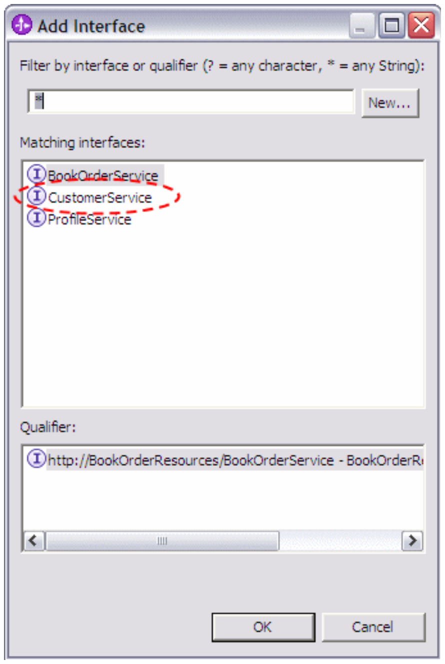

Add Interface
Filter by interface or qualifier (? = any character, * = any String):
*
New...
Matching interfaces:
BookOrderService
CustomerService
ProfileService
Qualifier:
http://BookOrderResources/BookOrderService - BookOrderR
OK
Cancel

Figure 12-2 Adding CustomerService interface to mediation flow component

5. Add the ProfileService interface as a reference to CustomerMediation.
   a. Right-click **CustomerMediation** and select **Add -> Reference**.
   b. Select the **ProfileService** interface and click **OK** (Figure 12-3).

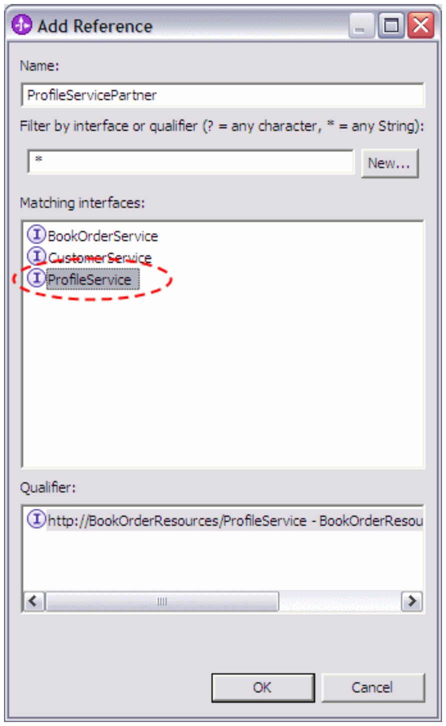


Figure 12-3 Adding ProfileService reference to mediation flow component

6. Right-click **CustomerMediation** and select **Generate Implementation**. In the Generate Implementation dialog, click **OK**. This will open the Mediation Flow editor.
7. Wire the **addCustomer** operation on the CustomerService interface to the **add** operation on the ProfileService reference (Figure 12-4).

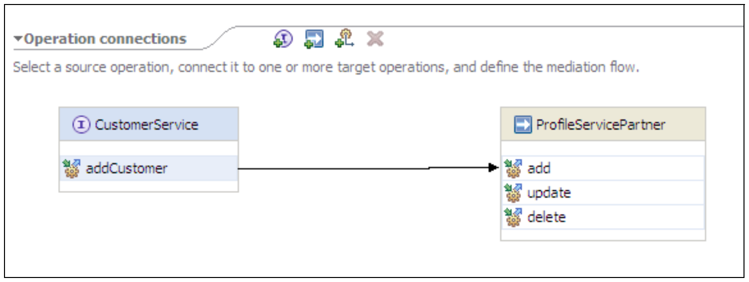

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
CustomerService
addCustomer
ProfileServicePartner
add
update
delete

Figure 12-4 Wiring operations

8. Add an XSL Transformation primitive to the request flow (Figure 12-5).

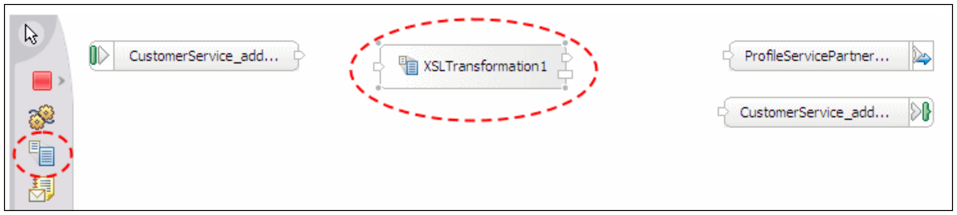


Figure 12-5 XSL Transformation primitive

9. Wire the request flow (Figure 12-6).
   a. Wire **CustomerService_addCustomer_Input** to the XSLTransformation1 **in** terminal.
   b. Wire the XSLTransformation1 **out** terminal to **ProfileServicePartner_add_Callout**.

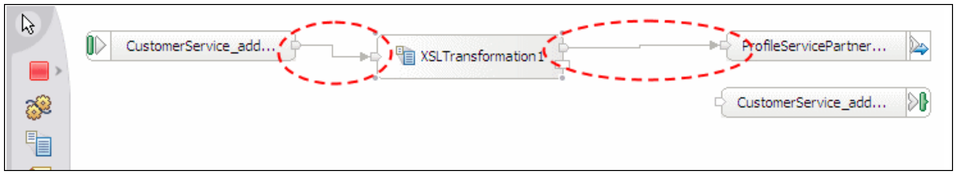


Figure 12-6 Wiring primitive terminals

10. Select XSLTransformation1 and click the **Details** tab in the Properties view.
11. Click the **New** button to create a new mapping file (Figure 12-7).

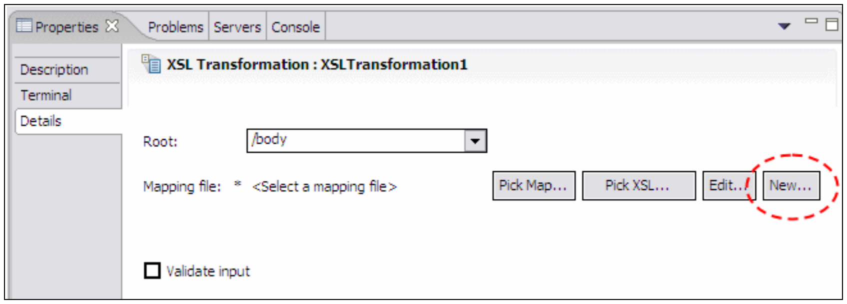
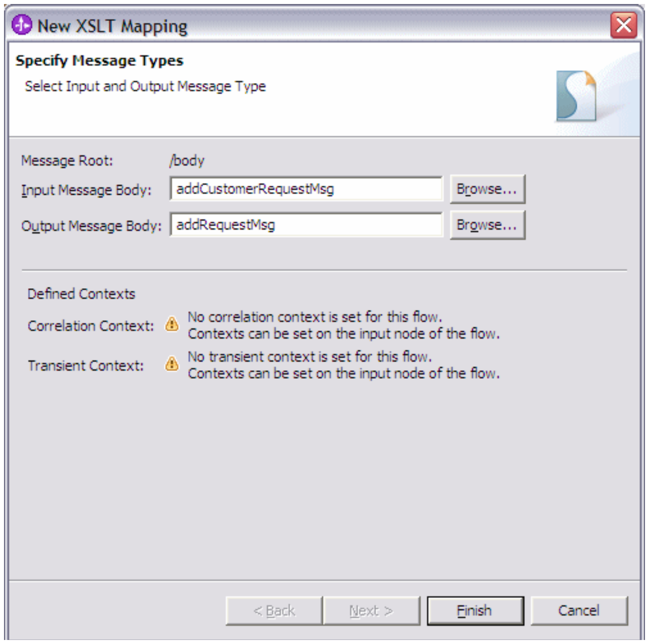


Figure 12-7 Create new mapping file

12. Leave the message types unchanged (Figure 12-8). We do not need to specify them because the terminals in the mediation primitive were already wired and their message types assigned. Click **Finish**.


New XSLT Mapping

Specify Message Types

Select Input and Output Message Type

Message Root: /body

Input Message Body: addCustomerRequestMsg

Output Message Body: addRequestMsg

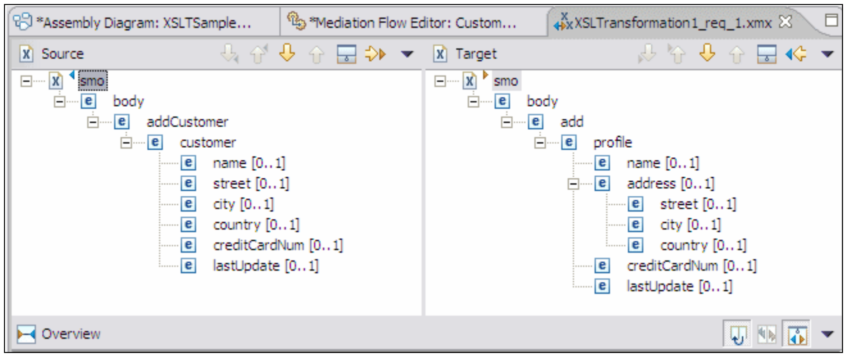
Defined Contexts

Correlation Context: No correlation context is set for this flow. Contexts can be set on the input node of the flow.

Transient Context: No transient context is set for this flow. Contexts can be set on the input node of the flow.

Figure 12-8 Specify message types

13. Now we will graphically develop the XSL Transformation by mapping elements from the source SMO to the target SMO. Expand the source and target messages in order to see all individual components (Figure 12-9).


*Assembly Diagram: XSLTSample... *Mediation Flow Editor: Custom... XSLTransformation1_req_1.xmx

Source
```
smo
  body
    addCustomer
      customer
        name [0..1]
        street [0..1]
        city [0..1]
        country [0..1]
        creditCardNum [0..1]
        lastUpdate [0..1]
```

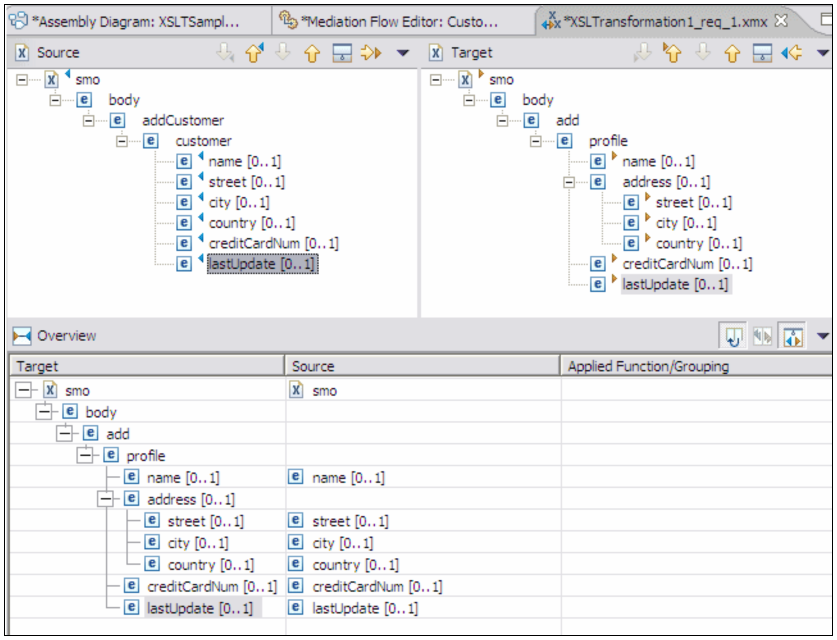
Target
```
smo
  body
    add
      profile
        name [0..1]
        address [0..1]
          street [0..1]
          city [0..1]
          country [0..1]
        creditCardNum [0..1]
        lastUpdate [0..1]
```

Figure 12-9 Source and target message types

14. Drag the **name** attribute of the customer element to the **name** attribute of the profile element.

15. Drag the **street** attribute of the customer element to the **street** attribute of the address element. The address element is a child of the profile element.

16. Drag all other attributes from the customer source element until the mapping is complete and all attributes have an associated source and target (Figure 12-10).


*Assembly Diagram: XSLTSampl... *Mediation Flow Editor: Custo... *XSLTransformation1_req_1.xmx

Source
```
smo
  body
    addCustomer
      customer
        name [0..1]
        street [0..1]
        city [0..1]
        country [0..1]
        creditCardNum [0..1]
        lastUpdate [0..1]
```

Target
```
smo
  body
    add
      profile
        name [0..1]
        address [0..1]
          street [0..1]
          city [0..1]
          country [0..1]
        creditCardNum [0..1]
        lastUpdate [0..1]
```

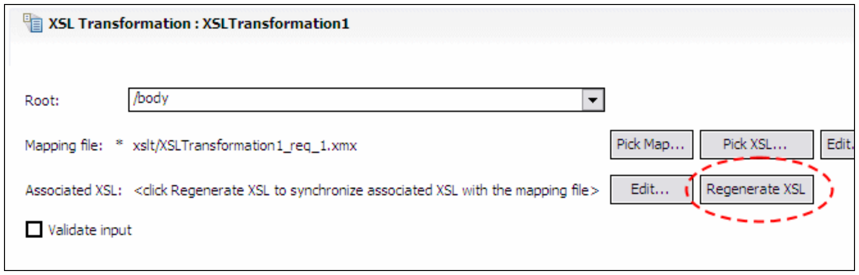
Overview

Target	Source	Applied Function/Grouping
smo	smo	
body		
add		
profile		
name [0..1]	name [0..1]	
address [0..1]		
street [0..1]	street [0..1]	
city [0..1]	city [0..1]	
country [0..1]	country [0..1]	
creditCardNum [0..1]	creditCardNum [0..1]	
lastUpdate [0..1]	lastUpdate [0..1]	

Figure 12-10 Complete mapping

17. Save and close the mapping file.

18. Back in the Mediation Flow editor, on the Details tab of the Properties view click the **Regenerate XSL** button (Figure 12-11).


XSL Transformation : XSLTransformation1

Root: /body

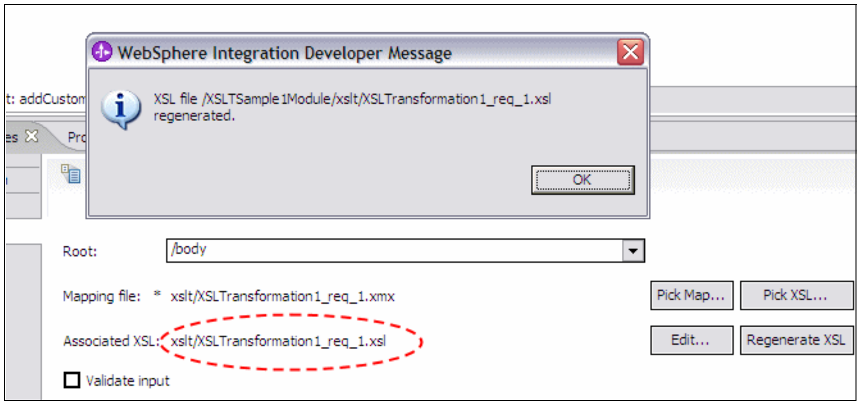
Mapping file: * xslt/XSLTransformation1_req_1.xmx

Associated XSL: <click Regenerate XSL to synchronize associated XSL with the mapping file>

☐ Validate input

Figure 12-11 Regenerate XSL

19. Click **OK** on the confirmation pop-up window. The Associated XSL field will be populated (Figure 12-12).


WebSphere Integration Developer Message

XSL file /XSLTSample1Module/xslt/XSLTransformation1_req_1.xsl regenerated.

Root: /body

Mapping file: * xslt/XSLTransformation1_req_1.xmx

Associated XSL: xslt/XSLTransformation1_req_1.xsl

Figure 12-12 XSL Generation complete

Important: Every time you edit the XSL mapping you must click the **Regenerate XSL** button.

20. Now click the **Response** tab of the Mediation Flow editor's center pane to compose the response flow.
21. Add an XSL Transformation primitive to the response flow.
22. Wire the response flow (Figure 12-13).
    a. Wire **ProfileServicePartner_add_CalloutResponse** to the XSLTransformation1 **in** terminal.
    b. Wire the **out** terminal of XSLTransformation1 to **CustomerService_addCustomer_InputResponse**.

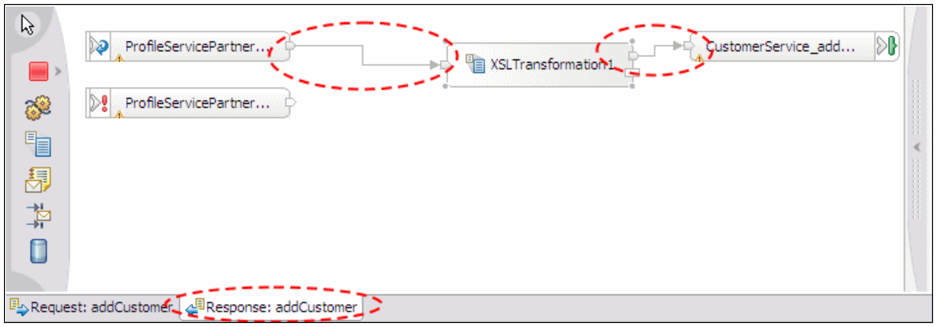
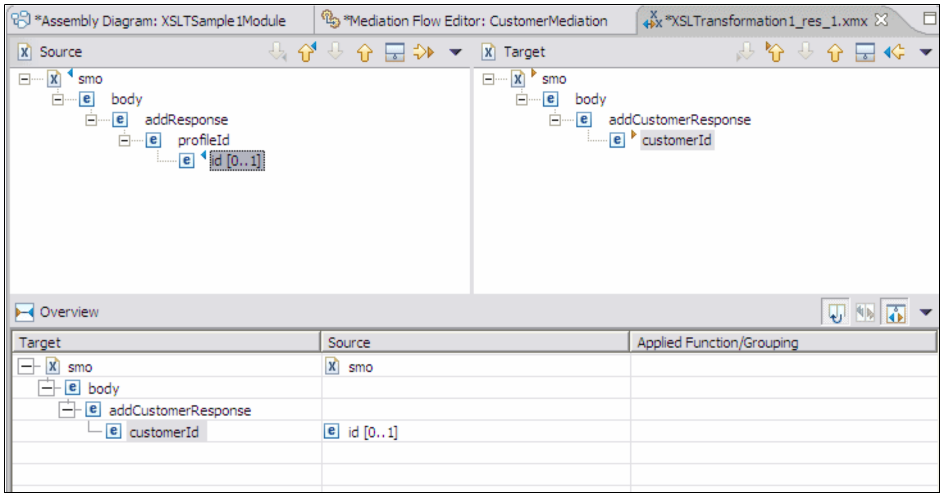


Figure 12-13 Wiring the response flow

23. Click on **XSLTransformation1** and in the Properties view select the **Details** tab.

24. Create a new mapping file by clicking on the **New** button, leave the default message types, and click **Finish**.
25. Map the **id** attribute of the profileId element on the source SMO to the **customerId** attribute on the target SMO (Figure 12-14).


*Assembly Diagram: XSLTSample1Module
*Mediation Flow Editor: CustomerMediation
*XSLTransformation1_res_1.xmx
Source
smo
body
addResponse
profileId
id [0..1]
Target
smo
body
addCustomerResponse
customerId
Overview

Target	Source	Applied Function/Grouping
smo	smo	
body		
addCustomerResponse		
customerId	id [0..1]	

Figure 12-14 Response flow mapping

26. Save and close the XSL Transformation.
27. Regenerate the XSL mapping file by clicking the **Regenerate XSL** button.
28. Save and close the mediation flow editor.
29. Add the ProfileService Web service as an import component on the XSLT1Sample1Module assembly diagram, and wire it to the CustomerMediation component as follows:
    a. In the Business Integration view, expand the BookOrderResources library. Locate **ProfileServiceSOAP** under Web Service Ports and drag it into the assembly diagram.
    b. In the Component Creation dialog box choose **Import with Web Service Binding** and click **OK**.
    c. Rename the import to `ProfileServiceImport`.
    d. Wire the reference on the CustomerMediation component to the interface on the ProfileServiceImport component (Figure 12-15).
    e. Save the assembly diagram.

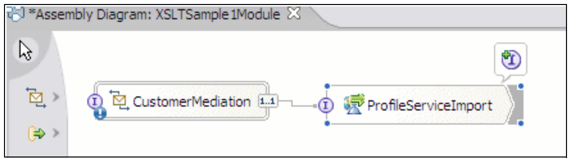


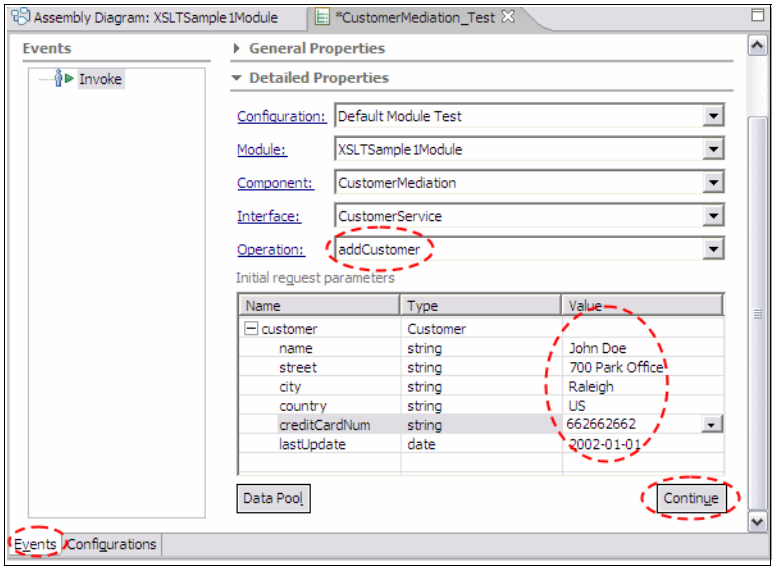
Figure 12-15 Mediation flow component wired to web service import

Now we are going to test our mediation using the Integration Test Client.

30. Deploy the module and Web service to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add ProfileServiceEAR and XSLTSample1ModuleApp.
    d. Click **Finish**.
31. In the Business Integration view, right-click the **XSLTSample1Module** project and select **Test** → **Test Module**.

> **Note:** By testing the module instead of the component we suppress all emulation.

32. On the Events tab of the Integration Test Client verify the Detailed Properties are correct (Figure 12-16).
    a. Ensure the Component is set to CustomerMediation.
    b. Ensure the Interface is set to CustomerService.
    c. Select **addCustomer** from the Operation drop down menu.
33. Populate the message data (Figure 12-16) and click **Continue**.



Configuration: Default Module Test
Module: XSLTSample1Module
Component: CustomerMediation
Interface: CustomerService
Operation: addCustomer

Initial request parameters

Name	Type	Value
customer	Customer	
name	string	John Doe
street	string	700 Park Office
city	string	Raleigh
country	string	US
creditCardNum	string	662662662
lastUpdate	date	2002-01-01

Figure 12-16 Test component event

34. In the Deployment Location dialog select **WebSphere ESB Server v6.0** and click **Finish** (Figure 12-17).

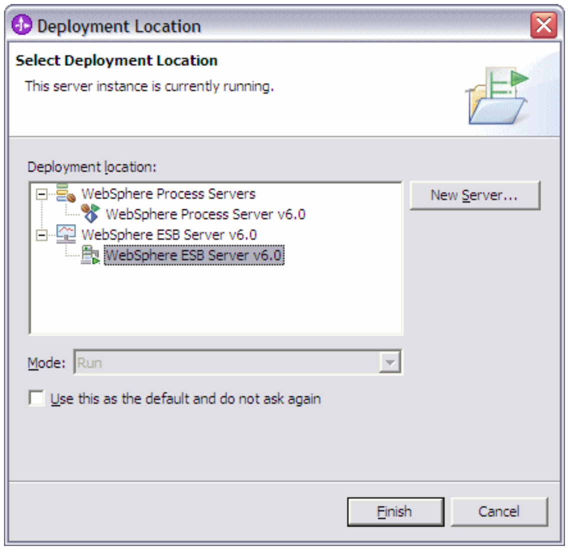
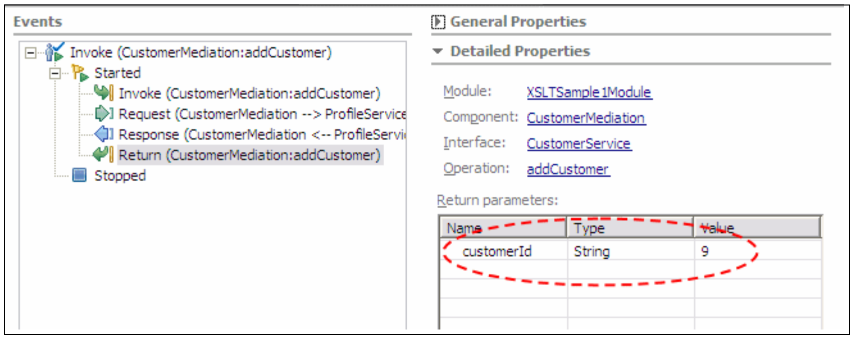

Deployment Location
Select Deployment Location
This server instance is currently running.
Deployment location:
WebSphere Process Servers
WebSphere Process Server v6.0
WebSphere ESB Server v6.0
WebSphere ESB Server v6.0
New Server...
Mode: Run
Use this as the default and do not ask again
Finish
Cancel

Figure 12-17 Deployment location

35. The response from the ProfileService Web service is a Confirmation message containing an id. We mapped this id attribute to our customerId string. We will see the response containing this customerId in the Return parameters pane (Figure 12-18).


Events
- Invoke (CustomerMediation:addCustomer)
  - Started
    - Invoke (CustomerMediation:addCustomer)
    - Request (CustomerMediation --> ProfileService
    - Response (CustomerMediation <-- ProfileServi
    - Return (CustomerMediation:addCustomer)
  - Stopped

General Properties

Detailed Properties

Module: XSLTSample1Module
Component: CustomerMediation
Interface: CustomerService
Operation: addCustomer

Return parameters:

Name	Type	Value
customerId	String	9

Figure 12-18 Return parameters

Congratulations! You successfully built and tested a mediation flow that transforms both the request and the response messages between two incompatible interfaces.

36. Once the testing is complete remove the projects from the server.

12.2 Database Lookup mediation primitive

This sample demonstrates how to use a Database Lookup mediation primitive to update a message with a value from a database.

The Database Lookup mediation primitive is used to search for values in a database. It takes a key which will be part of the Service Message Object and searches for that key in a database. If the key is found then a value associated with it is returned, and this can then be used to update the Service Message Object before forwarding it on.

This sample involves:

- Creating a Cloudscape database containing keys and values.
- Building a mediation module containing an import with a Web service binding.
- Implementing a mediation flow component using a Database Lookup mediation primitive.
- Testing the module using the Integration Test Client.

The completed sample invokes a book order Web service, which returns a confirmation ID. The mediation module is used to add a country to the description of the book, contained in the book order, before forwarding it to the Web service.

Perform the following:

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `DatabaseLookupSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double-clicking .
3. Rename mediation1 to `RareBookLookupMediation`. (Figure 12-19 on page 381)

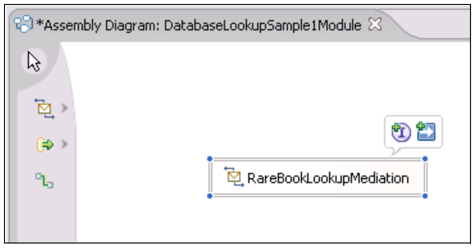


Figure 12-19 RareBookLookupMediation

4. Right-click on **RareBookLookupMediation**, and select **Add -> Interface**.
5. Choose the **BookOrderService** interface then click **OK**.
6. Right-click on **RareBookLookupMediation**, select **Add -> Reference.**
7. Choose the **BookOrderService** reference then click **OK**.
8. In the BookOrderResources project, under Web Service Ports, drag **BookOrderServiceSOAP** onto the Assembly Editor palette.
9. Select an **Import with Web service binding** and click **OK**.
10. Rename the import to `BookOrderService`.
11. Wire RareBookLookupMediation to BookOrderService (Figure 12-20 on page 382).

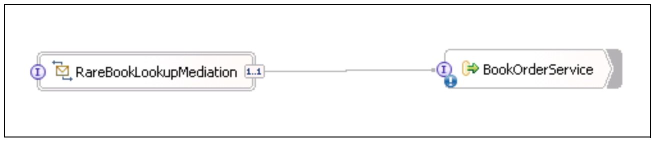


Figure 12-20 RareBookLookup Mediation Flow Component

12. Save the module.

We need to create a database of rare books for the DatabaseLookup mediation primitive to search.

13. Run the **`Cview.bat`** tool, this can be found in

    `<WID_INSTALL>/runtimes/bi_v6/cloudscape/bin/embedded`

14. Create a new database called `BookOrderDatabase` (Figure 12-21 on page 383) and add a table to it.

    a. Click **File** -> **New** -> **Database**, name the database `BookOrderDatabase` and click **Directory** to specify where the database will be created. Click **OK**.

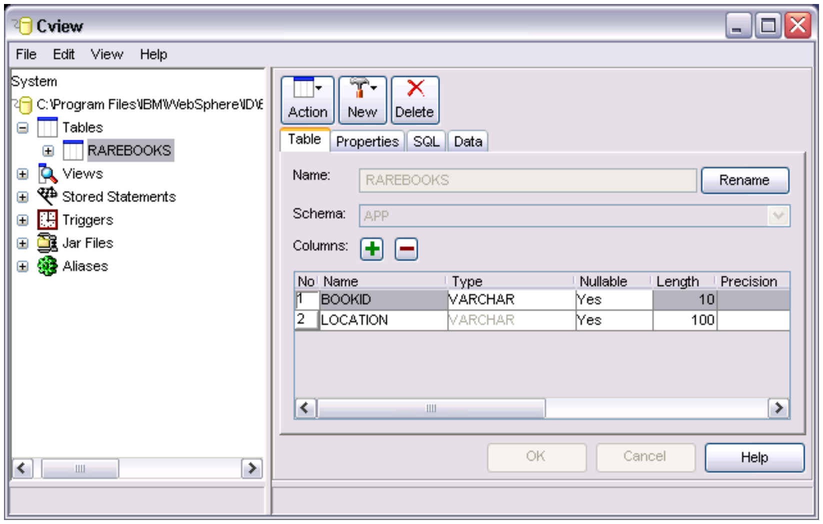
Note: Remember where you create the database as we will need to refer to it later.

    b. Create a new table called `RAREBOOKS`

    c. Create a column called `BOOKID` of type VARCHAR and length 10.

    d. Create a column called `LOCATION` of type VARCHAR and length 100.

    e. Click **OK** to save the table.



Name: RAREBOOKS

Schema: APP

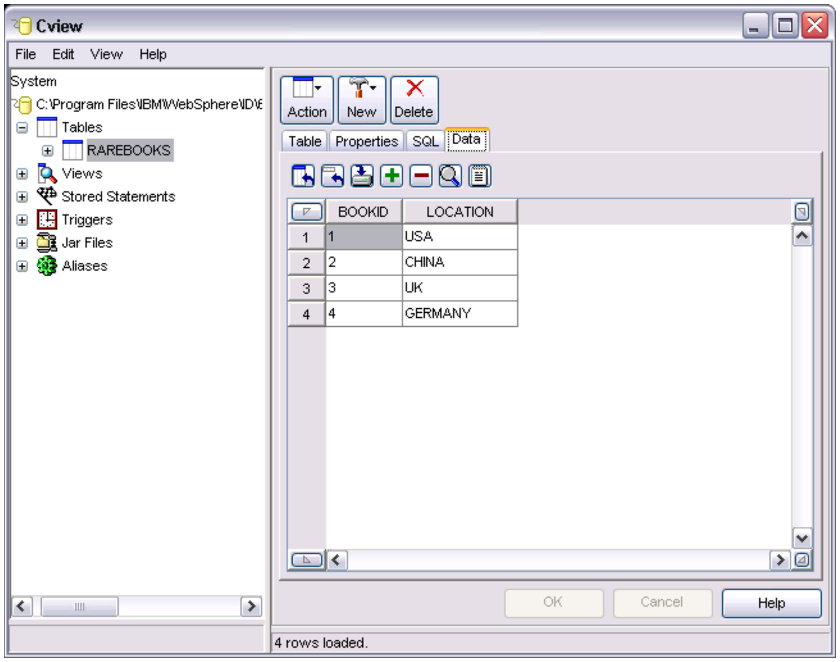
Columns:

No	Name	Type	Nullable	Length	Precision
1	BOOKID	VARCHAR	Yes	10	
2	LOCATION	VARCHAR	Yes	100	

Figure 12-21 RAREBOOKS Database Table

15. Switch to the **Data** tab.

16. Add BookId’s `1`, `2`, `3`, `4` with locations of `USA`, `CHINA`, `GERMANY`, `UK` respectively (Figure 12-22 on page 384).



	BOOKID	LOCATION
1	1	USA
2	2	CHINA
3	3	UK
4	4	GERMANY

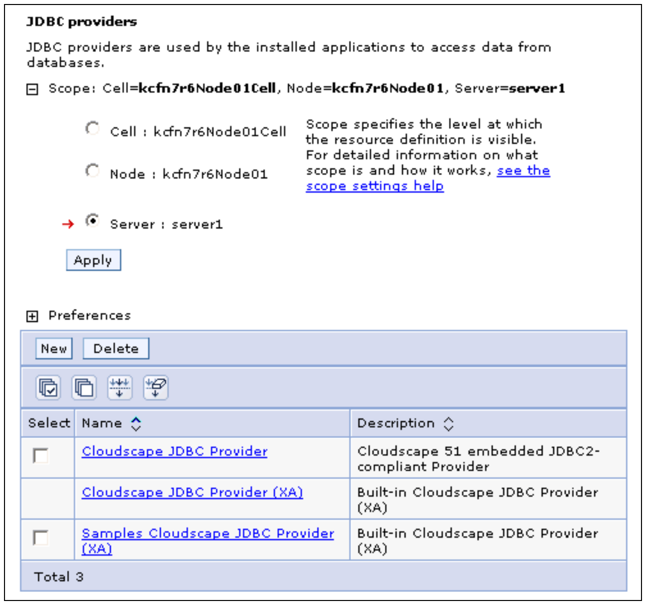
4 rows loaded.

Figure 12-22 BookOrderDatabase

17. Click **OK** and close Cview.

For the Database Lookup primitive to access the database we need to define a data source on the server.

18. In WebSphere Integration Developer, in the Server view, right-click your WebSphere Enterprise Service Bus server and select **Run administrative console**.
19. Log in to the console and click **Resources** -> **JDBC Providers**.
20. Choose the **Server** radio button and click **Apply**. This will show the JDBC providers defined on our server (Figure 12-23 on page 385).



JDBC providers

JDBC providers are used by the installed applications to access data from databases.

Scope: Cell=kcfn7r6Node01Cell, Node=kcfn7r6Node01, Server=server1

Cell : kcfn7r6Node01Cell

Node : kcfn7r6Node01

Server : server1 [✓]

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help

Select	Name	Description
☐	Cloudscape JDBC Provider	Cloudscape 51 embedded JDBC2-compliant Provider
	Cloudscape JDBC Provider (XA)	Built-in Cloudscape JDBC Provider (XA)
☐	Samples Cloudscape JDBC Provider (XA)	Built-in Cloudscape JDBC Provider (XA)

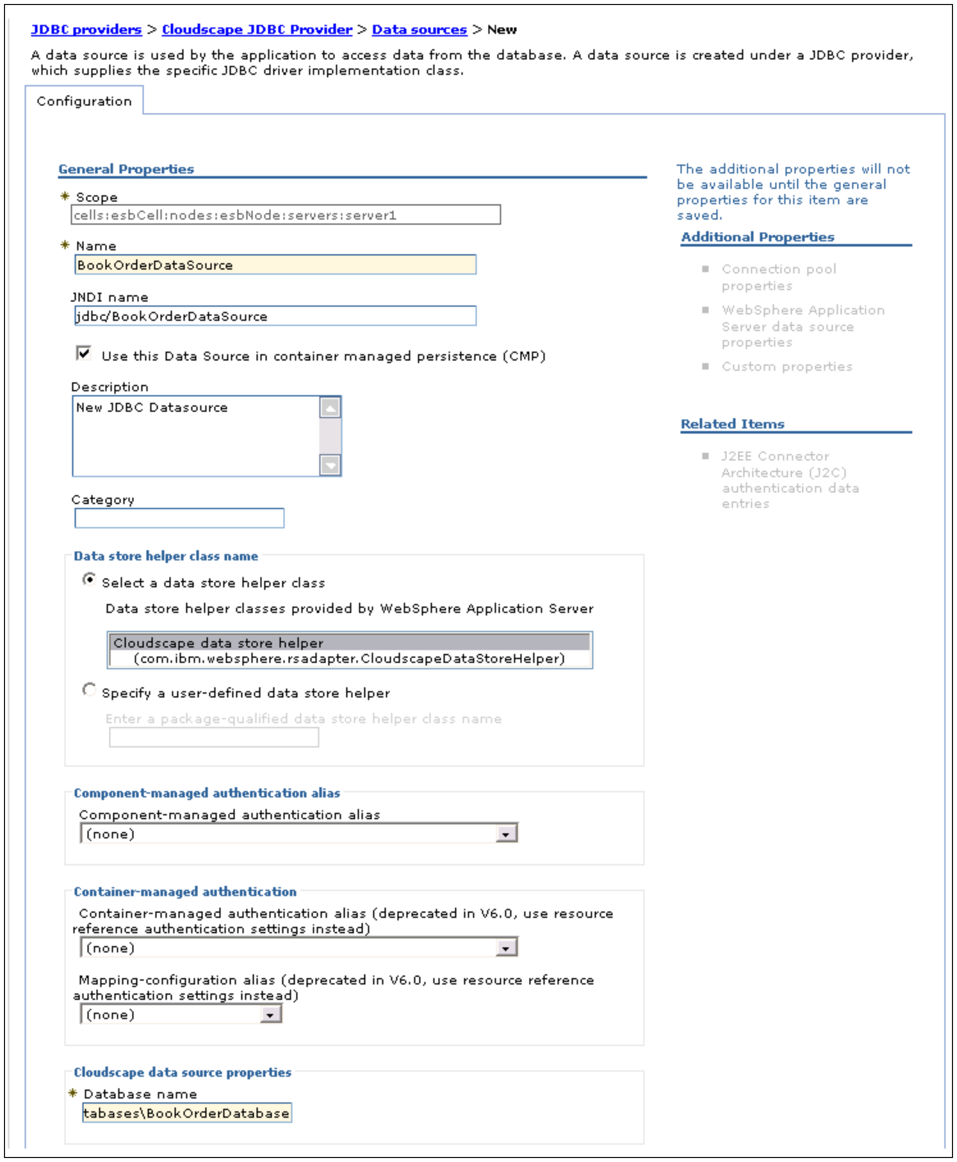
Total 3

Figure 12-23 JDBC Providers

21. Click **Cloudscape JDBC Provider**.
22. Click **Data sources**.
23. Press the **New** button to create a new data source (Figure 12-24 on page 386).
    a. Set Name to `BookOrderDataSource`.
    b. Set JNDI name to `jdbc/BookOrderDataSource`.
    c. Set Database name to the location of your database, for example:

    `C:\WID\runtimes\bi_v6\cloudscape\databases\BookOrderDatabase`

•


JDBC providers > Cloudscape JDBC Provider > Data sources > New

A data source is used by the application to access data from the database. A data source is created under a JDBC provider, which supplies the specific JDBC driver implementation class.

Configuration

General Properties

* Scope: cells:esbCell:nodes:esbNode:servers:server1

* Name: BookOrderDataSource

JNDI name: jdbc/BookOrderDataSource

[✓] Use this Data Source in container managed persistence (CMP)

Description: New JDBC Datasource

Category:

Data store helper class name

(●) Select a data store helper class

Data store helper classes provided by WebSphere Application Server

Cloudscape data store helper (com.ibm.websphere.rsadapter.CloudscapeDataStoreHelper)

( ) Specify a user-defined data store helper

Enter a package-qualified data store helper class name

Component-managed authentication alias

Component-managed authentication alias: (none)

Container-managed authentication

Container-managed authentication alias (deprecated in V6.0, use resource reference authentication settings instead): (none)

Mapping-configuration alias (deprecated in V6.0, use resource reference authentication settings instead): (none)

Cloudscape data source properties

* Database name: tabases\BookOrderDatabase

The additional properties will not be available until the general properties for this item are saved.

Additional Properties

- Connection pool properties
- WebSphere Application Server data source properties
- Custom properties

Related Items

- J2EE Connector Architecture (J2C) authentication data entries

Figure 12-24 BookOrder DataSource

24.Click **OK** and save the changes.

Now we have the database and data source, we can create the mediation flow.

25.In the Assembly Diagram editor, right-click on **RareBookLookupMediation** and select **Generate Implementation**.

26. Click **OK** to store the mediation flow in the default location. The Mediation Flow editor will open.

27. Under Operation connections, connect the **order** operation on the BookOrderService interface to the **order** operation on the BookOrderServicePartner reference (Figure 12-25 on page 387).

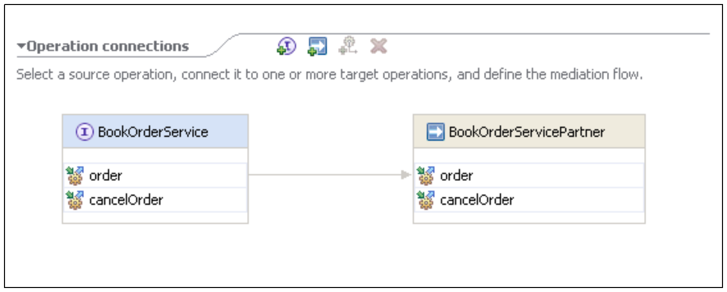

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
BookOrderService
order
cancelOrder
BookOrderServicePartner
order
cancelOrder

Figure 12-25 Operation connections

We use a DatabaseLookup mediation primitive to check for a key in the database. The key can be defined as any part of the message. If the key is found, we update the message with the value associated with that key and the message is passed to the default output terminal. If the key is not found the message is unmodified and passed to the KeyNotFound output terminal.

28. Add a Database Lookup mediation primitive to the palette by using .

29. Rename it to `RareBookLookup`.

30. Wire the request flow (Figure 12-26).

  a. Wire **BookOrderService_order_Input** to the **in** terminal of RareBookLookup.

  b. Wire the **out** terminal and the **KeyNotFound** terminal of RareBookLookup to **BookOrderServicePartner_order_Callout**.

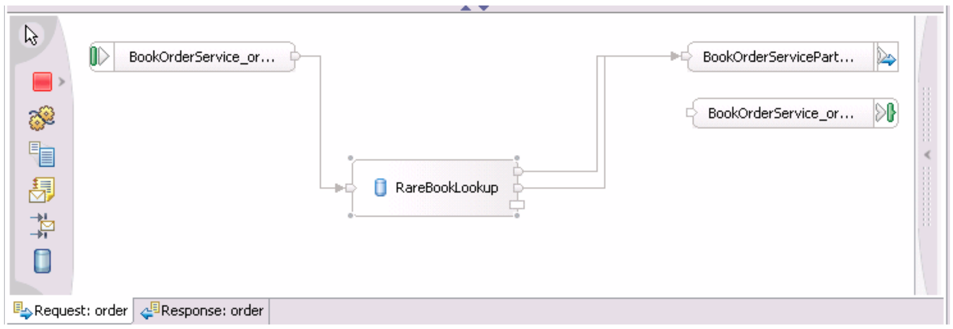
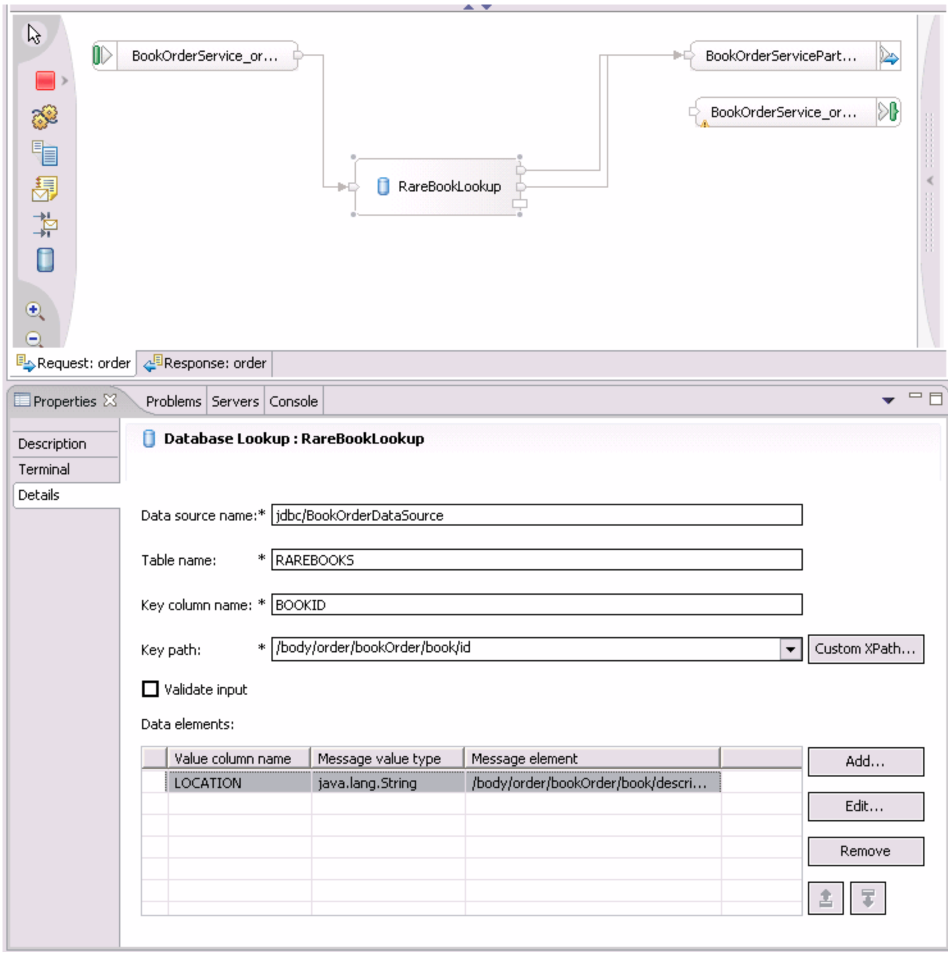


Figure 12-26 RareBookLookup mediation flow

31. On the Details tab of RareBookLookup properties panel enter the following settings (Figure 12-27 on page 389).
    a. Data source name: `jdbc/BookOrderDataSource`
    b. Table name: `RAREBOOKS`
    c. Key column name: `BOOKID`
    d. Click the **Custom XPath** button, enter Key path `/body/order/bookOrder/book/id` and click **OK.**
    e. Add the Data element:
        i. Value column name: `LOCATION`
        ii. Message value type: `java.lang.String`
        iii. Message element: `/body/order/bookOrder/book/description`



Database Lookup : RareBookLookup

Data source name:* jdbc/BookOrderDataSource

Table name: * RAREBOOKS

Key column name: * BOOKID

Key path: * /body/order/bookOrder/book/id

Custom XPath...

☐ Validate input

Data elements:

Value column name	Message value type	Message element
LOCATION	java.lang.String	/body/order/bookOrder/book/descri...

Add... Edit... Remove

Figure 12-27 Database Lookup mediation primitive properties

32. Click the **Response** tab, and in the response flow wire BookOrderServicePartner_order_CalloutResponse directly to BookOrderService_order_InputResponse (Figure 12-28 on page 390).

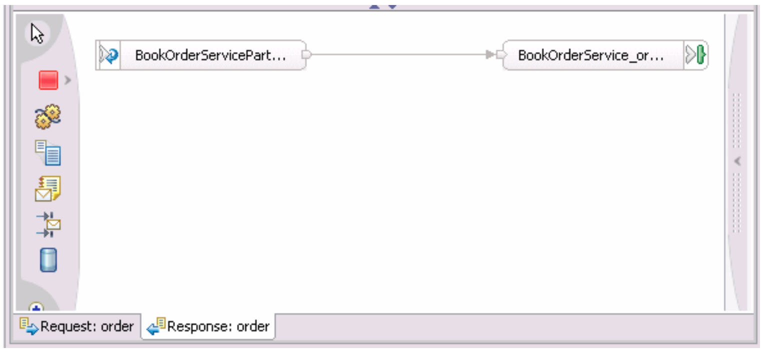


Figure 12-28 Repsonse Flow

33. Save the mediation flow component and the module.

The development of the mediation module is complete. Now we test the module using the Integration Test Client.

34. In the Request view, right-click on **BookOrderServicePartner_order_Callout** and select **Add Breakpoint** (Figure 12-29).

> **Note:** By adding a breakpoint to BookOrderServicePartner_order_Callout the flow will stop when the message reaches this point and the contents will be displayed. This allows us to view the changes made by the DatabaseLookup mediation primitive.

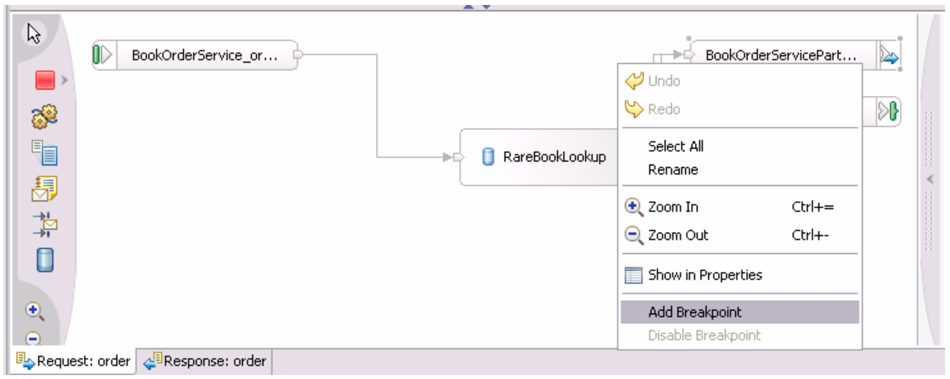
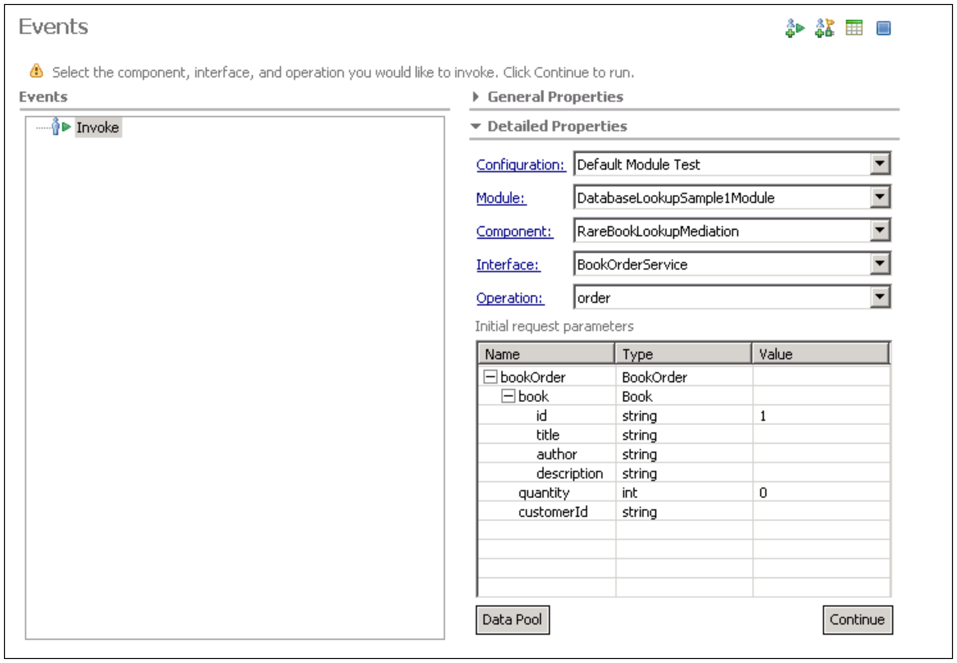


Figure 12-29 Adding a breakpoint

Note: If the server is running you will need to stop it now.

35. Right-click on the WebSphere Enterprise Service Bus server and select **Debug**.
36. Deploy the module and the Web service.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add BookOrderServiceEAR and DatabaseLookupSample1Module.
    d. Click **Finish**.
37. In the Business Integration view, right-click on **DatabaseLookupSample1Module** and click **Test** -> **Test Module** to open the Integration Test Client.
38. Select the **RareBookLookupMediation** component and the **order** operation from the drop down menus.
39. Enter a book ID of **1** (Figure 12-30 on page 391).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: DatabaseLookupSample1Module
Component: RareBookLookupMediation
Interface: BookOrderService
Operation: order

Initial request parameters

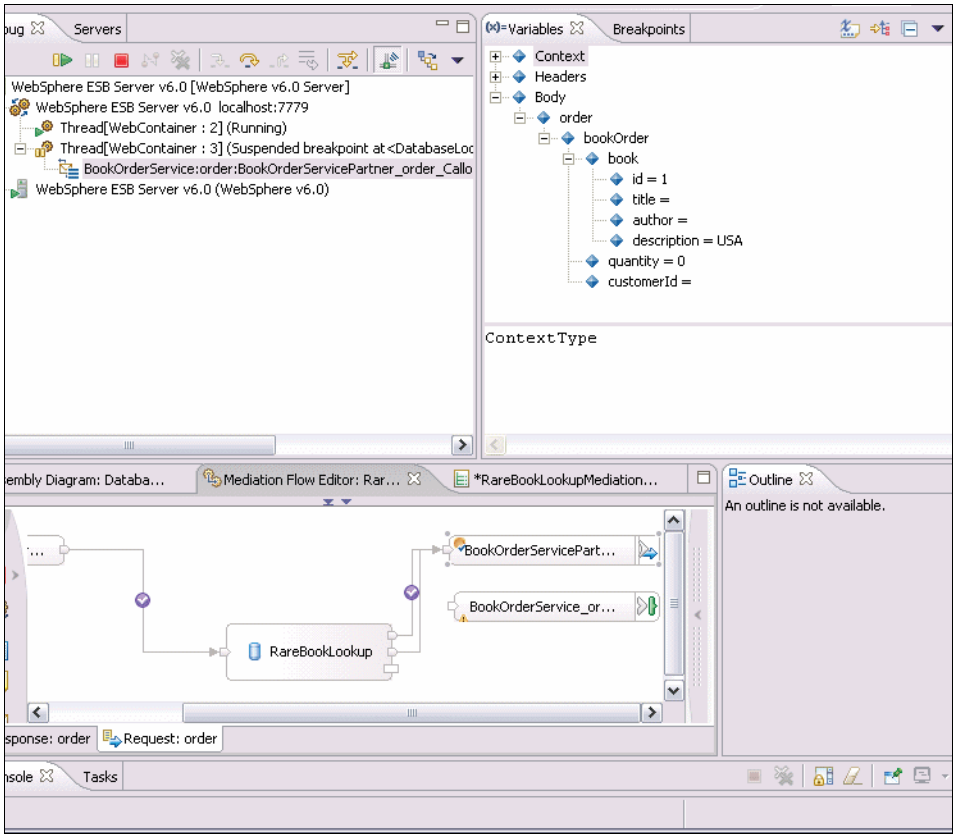
Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	1
title	string	
author	string	
description	string	
quantity	int	0
customerId	string	

Figure 12-30 Testing DatabaseLookupSample1Module

40. Press **Continue**.

41. Select the WebSphere Enterprise Service Bus server and press **Finish**.

42. This will prompt open the Debug perspective. In the variables view expand the body of the message to check the value from the database has been added to the description (Figure 12-31 on page 392).



```
WebSphere ESB Server v6.0 [WebSphere v6.0 Server]
  WebSphere ESB Server v6.0 localhost:7779
    Thread[WebContainer : 2] (Running)
    Thread[WebContainer : 3] (Suspended breakpoint at <DatabaseLoc
      BookOrderService:order:BookOrderServicePartner_order_Callo
  WebSphere ESB Server v6.0 (WebSphere v6.0)
```

```
Context
Headers
Body
  order
    bookOrder
      book
        id = 1
        title =
        author =
        description = USA
      quantity = 0
      customerId =
ContextType
```

An outline is not available.

Figure 12-31 Debugging DatabaseLookupSample1module

43. Click the resume button to continue the flow. A confirmation message is returned.

Try using values `2`, `3` and `4` for the book ID to see other values from the database.

Try setting the book ID to a value we know isn't in the database such as `x` and running the test again. You will see the purple tick on the wire from the KeyNotFound output terminal instead of the default output terminal (Figure 12-32).

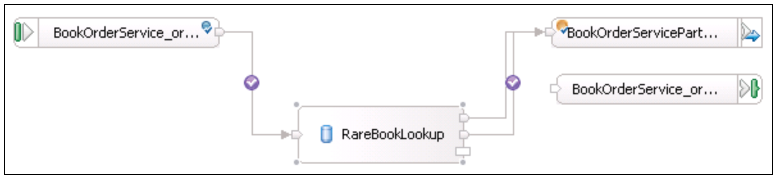


Figure 12-32 Database Lookup KeyNotFound

44.Once the testing is complete remove the projects from the server.

Congratulations, you have successfully built and tested a mediation module containing a DatabaseLookup mediation primitive.

Note: The Database Lookup mediation primitive is commonly used in conjunction with the Message Filter mediation primitive. Using the DatabaseLookup a value can be obtained from a database and stored in the transient or correlation context. The MessageFilter can then filter messages using this value by defining an XPath expression to the context.

12.3 Message Filter mediation primitive

This sample demonstrates how to create a Message Filter mediation primitive to route messages based on message content.

The Message Filter mediation primitive routes messages based on conditions that are defined on it's out terminals. Multiple out terminals can be defined and each has an XPath expression associated with it. If the Service Message Object entering the primitive satisfies the XPath expression the message may be routed to that terminal. Messages may be routed to the first terminal to match it's expression or to all terminals that match their expressions.

This sample involves:

- Building a mediation module containing two imports with a Web service bindings.
- Implementing the request flow using a Message Filter mediation primitive.
- Testing the module using the Integration Test Client.

The completed sample invokes a book order Web service, which returns a confirmation ID. The mediation module is used to check for a specific bookid and route book orders using that id to a rare-book ordering service.

1. Create a new mediation module.

   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `MessageFilterSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog tick the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double-clicking .
3. Rename Mediation1 to `FilterRareOrdersMediation` (Figure 12-33).

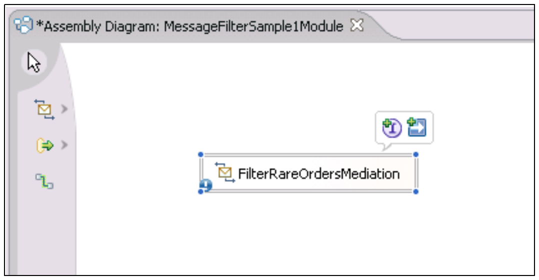

*Assembly Diagram: MessageFilterSample1Module
FilterRareOrdersMediation

Figure 12-33 FilterRareOrdersMediation

4. Right-click **FilterRareOrdersMediation** and select **Add -> Interface**.
5. Select the **BookOrderService** interface and click **OK**.
6. In this example we are going to use Java components to represent Web services. They will return a unique string so we can identify which service was invoked. Add a Java component to the assembly editor by using .
7. Rename it to `StandardBookOrderService`.
8. Add another Java component to the palette and rename it to `RareBookOrderService` (Figure 12-34).

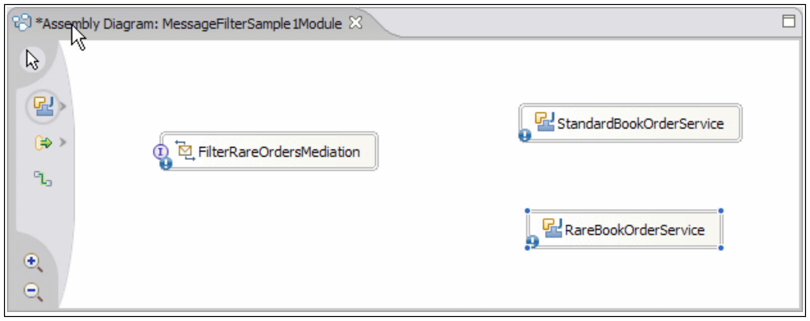
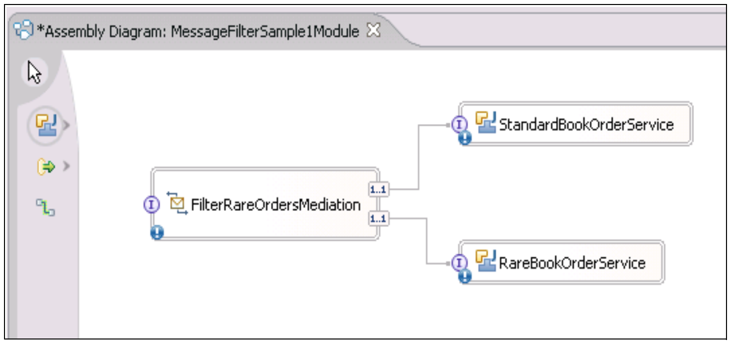

*Assembly Diagram: MessageFilterSample1Module
FilterRareOrdersMediation
StandardBookOrderService
RareBookOrderService

Figure 12-34 Adding Java components to MessageFilterSample1Module

9. Wire FilterRareOrdersMediation to StandardBookOrderService.

10. In the Add Reference dialog select **BookOrderService** and click **OK**.
11. Wire FilterRareOrdersMediation to RareBookOrderService.
12. On the Add Reference dialog select **BookOrderService** and click **OK** (Figure 12-35).

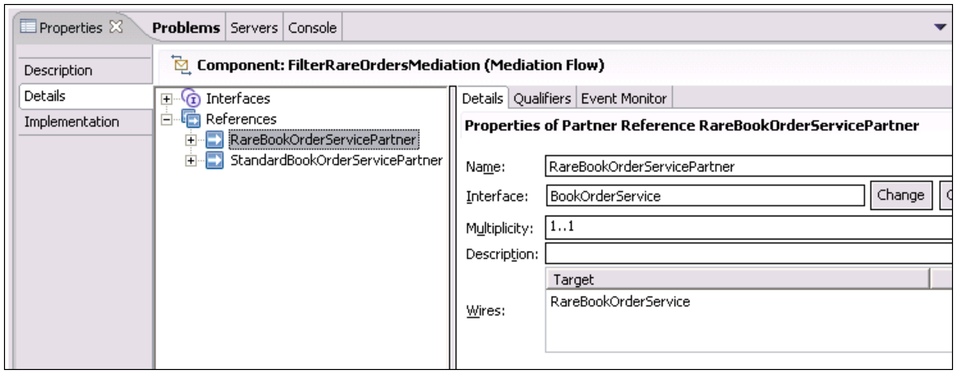


*Assembly Diagram: MessageFilterSample1Module
StandardBookOrderService
FilterRareOrdersMediation
1..1
1..1
RareBookOrderService

Figure 12-35 Wired Java Components in MessageFilterSample1Module

13. Click FilterRareOrdersMediation and in the Properties view, select the **Details** tab.
14. Expand the **References** tree.
15. Click on the **BookOrderServicePartner** reference and set its name to `StandardBookOrderServicePartner`.
16. Click on the **BookOrderServicePartner1** reference and set its name to `RareBookOrderServicePartner` (Figure 12-36 on page 396).

Note: Renaming the references makes it clearer when wiring the mediation flow.



Properties of Partner Reference RareBookOrderServicePartner

Name: RareBookOrderServicePartner
Interface: BookOrderService
Multiplicity: 1..1
Description:
Wires: Target
RareBookOrderService

Figure 12-36 Renaming references

17. In the assembly diagram right-click on **StandardBookOrderService** and select **Generate Implementation**.
18. In the Generate Implementation dialog click **OK** to use the default package.
19. The Java editor displays the content of StandardBookOrderImpl.java. Find the order method and replace it with Example 12-1.

Example 12-1 Java code for order method in StandardBookOrderImpl.java

```
public String order(DataObject bookOrder) {
   return "Confirmation of Standard book order";
}
```

20. Save and close StandardBookOrderImpl.java.
21. Right-click on **RareBookOrderService** and select **Generate Implementation**.
22. In the Generate Implementation dialog click **OK** to use the default package.
23. The Java editor displays the content of RareBookOrderImpl.java. Find the order method and replace it with Example 12-2.

Example 12-2 Java code for order method in RareBookOrderImpl.java

```
public String order(DataObject bookOrder) {
   return "Confirmation of Rare book order";
}
```

24. Save and close RareBookOrderImpl.java.

We have two book ordering services but only need to invoke one. To decide which service handles the request a Message Filter mediation is used.

25. Right-click **FilterRareOrdersMediation** and select **Generate Implementation**.

26. In the Generate Implementation dialog click **OK** to create the implementation in the default location. The Mediation Flow Editor is displayed.

27. In the Operation connections panel wire the **order** operation on BookOrderService to the **order** operation on StandardBookOrderServicePartner.

28. Also wire the **order** operation on BookOrderService to the **order** operation on RareBookOrderServicePartner (Figure 12-37).

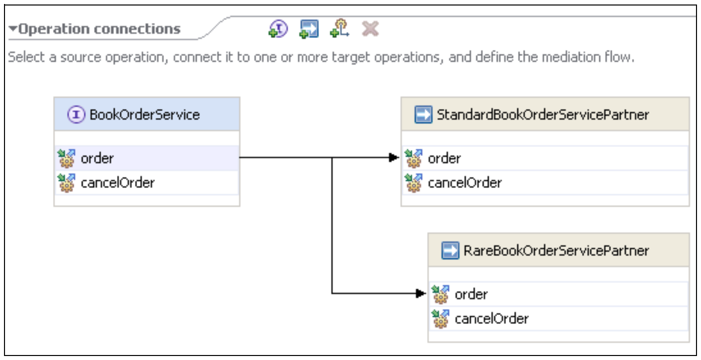

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
BookOrderService
order
cancelOrder
StandardBookOrderServicePartner
order
cancelOrder
RareBookOrderServicePartner
order
cancelOrder

Figure 12-37 Wiring operations

29. Add a Message Filter mediation primitive to the mediation flow by using .

30. Rename the Message Filter to `FilterRareBooks`.

When first created the Message Filter mediation primitive only has one output terminal. This default terminal is fired when a message does not match the requirements specified on the other output terminals. Therefore the default behavior of the Message Filter mediation primitive is to forward messages. We need to check whether to use the standard or rare book service so another output terminal is required.

31. Right-click on **FilterRareBooks** and select **Add Output Terminal**.

32. On the New Dynamic Terminal dialog set the Terminal name to `RareBook` and click **OK**.

33. Wire the request flow (Figure 12-38 on page 398).
    a. Wire **BookOrderService_order_Input** to the **in** terminal of FilterRareBooks.
    b. Wire the **default** terminal of FilterRareBooks to **StandardBookOrderServicePartner_order_Callout**.

c. Wire the **RareBook** terminal of FilterRareBooks to **RareBookOrderServicePartner_order_Callout**.

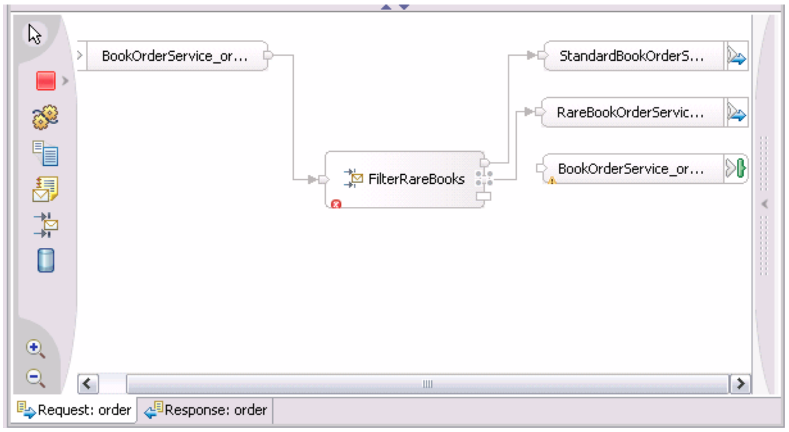


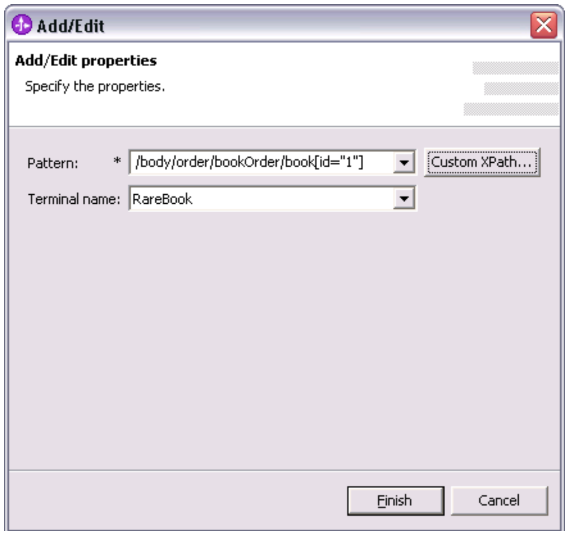
Figure 12-38 Wiring the request mediation flow

34. Save the mediation flow.

Note: The error on FilterRareBooks indicates we need to define a filter for the RareBook output terminal.

35. Click **FilterRareBooks** and in the Properties view, select the **Details** tab.
36. The default distribution mode is set to First. This will send the message to the first output terminal that satifies its filter. Click the **Add** button, to add a filter.
37. Fill in the Add/Edit properties panel. (Figure 12-39)
    a. Select the Terminal name to be **RareBook**
    b. Set the **Pattern** field to `/body/order/bookOrder/book[id="1"]`
    c. Click **Finish**.

Note: This expression states that the RareBook output terminal will fire if the id attribute of the book, contained in the BookOrder is equal to 1.


Add/Edit

Add/Edit properties

Specify the properties.

Pattern: * /body/order/bookOrder/book[id="1"]

Terminal name: RareBook

Figure 12-39 Adding a filter

38. Switch to the **Response** tab.

39. Wire the response flow (Figure 12-40 on page 399).

   a. Wire **StandardBookOrderServicePartner_order_CalloutResponse** to **BookOrderService_order_InputResponse**.

   b. Wire **RareBookOrderServicePartner_order_CalloutResponse** to **BookOrderService_order_InputResponse**.

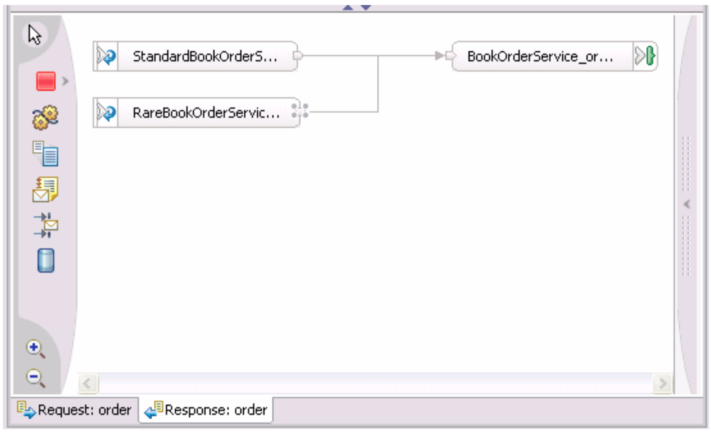
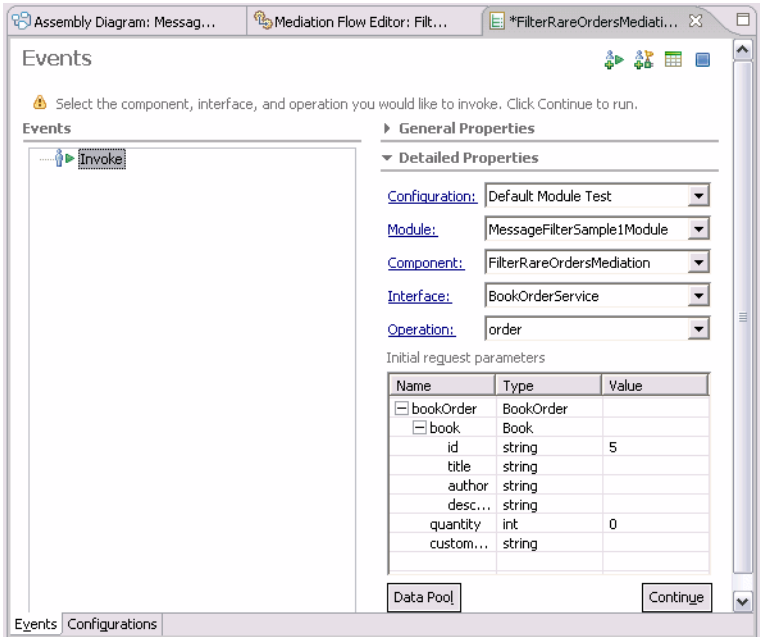


Figure 12-40 Wiring the response flow

40. Save the mediation flow and the module.

The development of the mediation module is complete. Now we test the module using the Integration Test Client.

41. Deploy the module to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add MessageFiterSample1Module.
    d. Click **Finish**.
42. Right-click **MessageFiterSample1Module** and select **Test Module**. This opens the Integration Test Client.
43. Ensure the Component is FilterRareOrdersMediation.
44. Select the **order** operation from the Operation drop down menu.
45. Set the book id to 5 (Figure 12-41 on page 400).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: MessageFilterSample1Module
Component: FilterRareOrdersMediation
Interface: BookOrderService
Operation: order

Initial request parameters

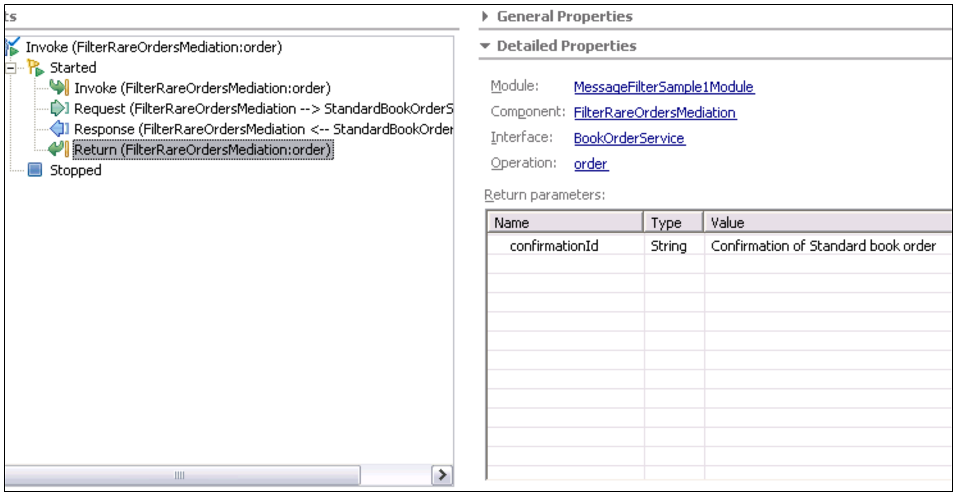
Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	5
title	string	
author	string	
desc...	string	
quantity	int	0
custom...	string	

Figure 12-41 Testing MessageFilterSample1Module

Note: By entering a book ID of 5 we cause the expression on RareBook terminal on the MessageFilter mediation primitive to be false. Therefore the message is sent to the standard book order service.

46. Press the **Continue** button.

47. Select the WebSphere Enterprise Service Bus server and click **Finish**. The response should look like (Figure 12-42 on page 401).



```
Invoke (FilterRareOrdersMediation:order)
Started
    Invoke (FilterRareOrdersMediation:order)
    Request (FilterRareOrdersMediation --> StandardBookOrderS
    Response (FilterRareOrdersMediation <-- StandardBookOrder
    Return (FilterRareOrdersMediation:order)
Stopped
```

Module: MessageFilterSample1Module
Component: FilterRareOrdersMediation
Interface: BookOrderService
Operation: order

Return parameters:

Name	Type	Value
confirmationId	String	Confirmation of Standard book order

Figure 12-42 Response from bookid of 5

48. Repeat the test but use 1 as the book Id. You should see the response coming back from the RareBookOrderService.

49. Once the testing is complete remove the project from the server.

Congratulations you have successfully built and tested a mediation module that demonstrates the Message Filter mediation.

Note: On the Message Filter mediation primitive we can set the distribution mode to All. This would fire all matching output terminals resulting in requests being sent to multiple services. This in turn would result in multiple responses being returned, and this would need to be handled by the response flow. Remember only one response can be returned to the service consumer otherwise a ServiceRuntimeExcpetion occurs.

12.4 Message Logger mediation primitive

This sample demonstrates how the Message Logger mediation primitive is used to store a message in a database.

The Message Logger mediation primitive is used to log Service Message Objects to a database.

This sample involves:

- Building a mediation module containing an import with a Web service binding.
- Implementing the request flow using a Message Logger mediation primitive.
- Testing the module using the Integration Test Client.

The completed sample invokes a book order Web service, which returns a confirmation ID. The mediation module is used to log the book order sent to the Web service.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `MessageLoggerSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog tick the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Rename Mediation1 to **LogMessageMediation** (Figure 12-43).

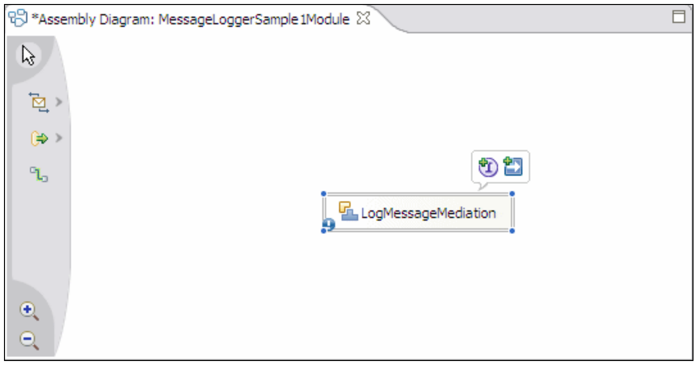
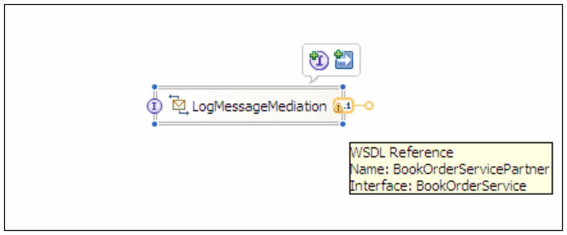


Figure 12-43 LogMessageMediation

4. Right-click **LogMessageMediation** and select **Add -> Interface.**
5. Select the **BookOrderService** interface and click **OK.**
6. Right-click **LogMessageMediation** and select **Add -> Reference**.
7. Select the **BookOrderService** interface and click **OK.** (Figure 12-44 on page 403).



WSDL Reference
Name: BookOrderServicePartner
Interface: BookOrderService

Figure 12-44 LogMessageMediation flow component

8. Expand the **BookOrderResources** library in the Business Integration view and select **BookOrderServiceSOAP** from Web Service Ports (Figure 12-45 on page 403).

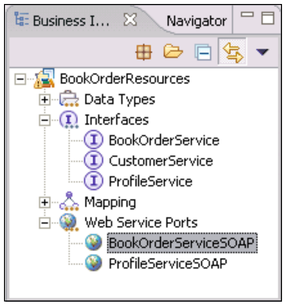


Figure 12-45 Select BookOrderServiceSOAP

9. Drag and drop the **BookOrderServiceSOAP** into the assembly diagram of the MessageLoggerSample1Module.

10. This will open the Component Creation window. Select **Import with Web Service Binding** and click **OK.**

11. Rename Import1 to `BookOrderServiceImport`.

12. Wire the reference BookOrderServicePartner on LogMessageMediation to the BookOrderServerInterface on the import BookOrderServiceImport (Figure 12-46 on page 403).

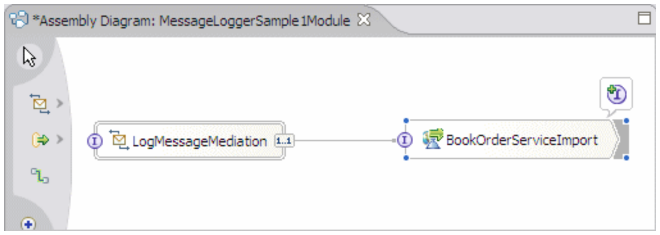


Figure 12-46 Assembly Diagram

13. Save the module.

Now we have a mediation flow component with an interface and a reference, lets generate the mediation flow.

14. Right-click **LogMessageMediation** and select **Generate Implementation**.

15. Click **OK** to store the implementation in the default folder. This opens the mediation flow editor.

16. In the Operation connections section of this view, wire the **order** method on the BookOrderService Interface to the **order** method of the BookOrderServicePartner reference, as shown in Figure 12-47.

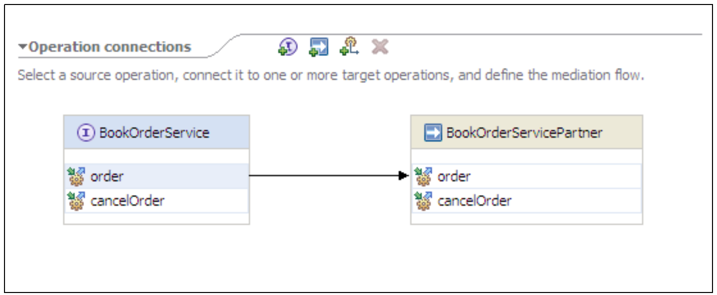

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
BookOrderService
order
cancelOrder
BookOrderServicePartner
order
cancelOrder

Figure 12-47 Operation connection

17. Add a Message Logger mediation primitive to the assembly editor by using .

18. Rename it to `LogMessage`.

19. Wire the request flow (Figure 12-48).

   a. Wire **BookOrderService_order_Input** to the **in** terminal of LogMessage.

   b. Wire the **out** terminal of LogMessage to **BookOrderServicePartner_order_Callout**.

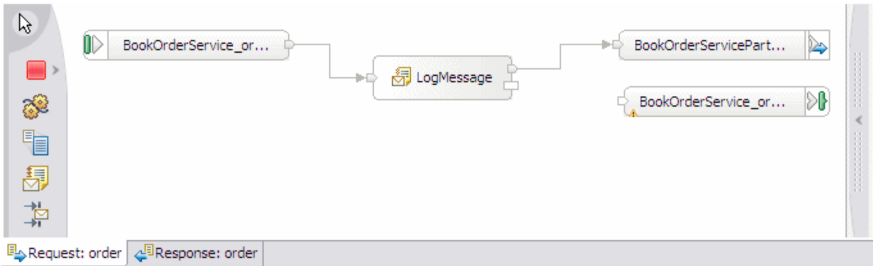
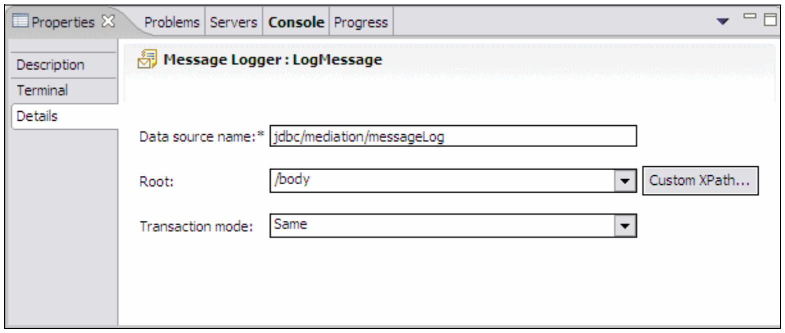


Figure 12-48 Messagelogger mediation flow

20. Select **LogMessage** and in the Properties view, select the **Details** tab.

21. View the Data source name, the XPath expression selected to log and the Transaction mode (Figure 12-49).



Message Logger : LogMessage

Data source name:* jdbc/mediation/messageLog

Root: /body

Transaction mode: Same

Figure 12-49 Messagelogger properties

This sample uses the default Cloudscape database **EsbLogMedDB** that is configured with the complete installation. It is possible to log messages to other relational databases using the Message Logger primitive. The jdbc/mediation/messageLog data source will be already defined in WebSphere Enterprise Service Bus.

The Root shows the XPath expression defining the data from the Service Message Object to log to the database. This can be customized using the Custom XPath button.

The default Transaction mode is set to Same. This will commit the message to the database within the flow's transaction. Setting the Transaction mode to New will commit the message to the database immediately using a new transaction.

22. Click the **Response** tab in the mediation flow editor.

23. Wire **BookOrderServicePartner_order_CalloutResponse** to **BookOrderService_order_InputResponse** (Figure 12-50).

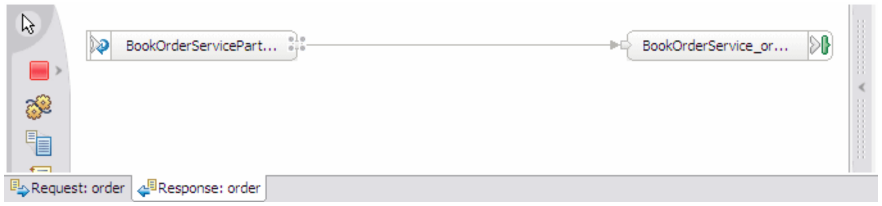
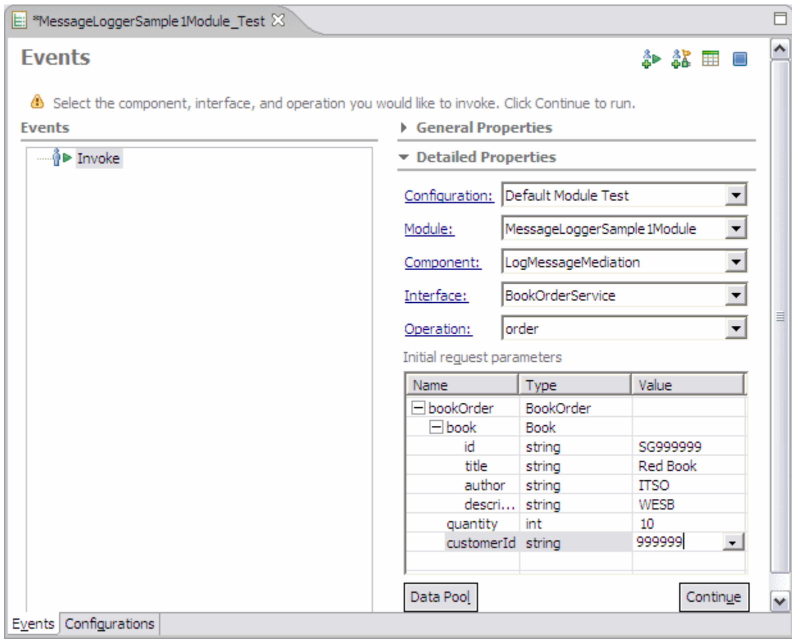


Figure 12-50 Response flow

24. Save the mediation flow and the mediation module.

To test the flow we use the Integration Test Client to show the flow logging a message to the database EsbLogMedDB.

25. Deploy the module and the Web service to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add BookOrderServiceEAR and MessageLoggerSample1ModuleApp.
    d. Click **Finish**.
26. Right-click **MessageLoggerSample1Module** and select **Test -> Test Module**. This opens the Integration Test Client.
27. Set the Component to **LogMessageMediation**.
28. Select **order** from the Operation drop down menu.
29. Enter values for all the fields in the order operation (Figure 12-51).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

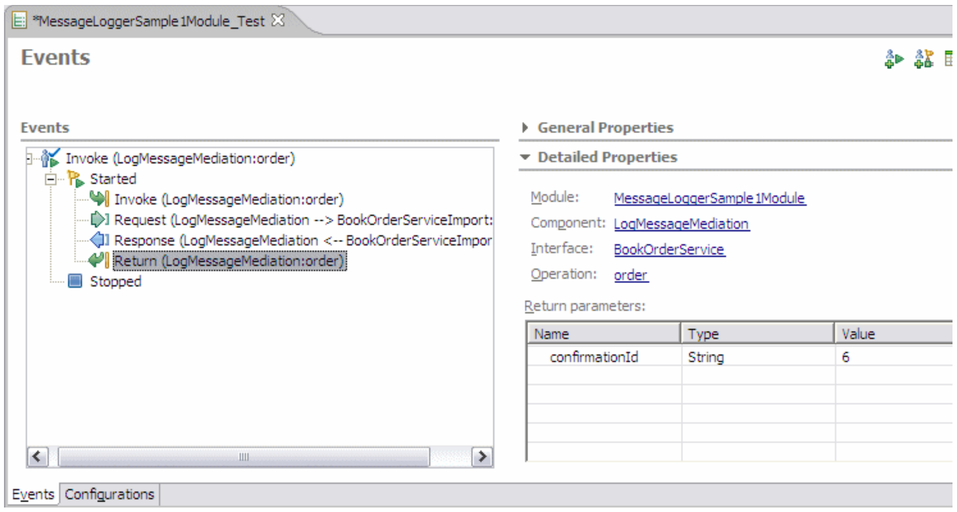
Configuration: Default Module Test
Module: MessageLoggerSample1Module
Component: LogMessageMediation
Interface: BookOrderService
Operation: order

Initial request parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	SG999999
title	string	Red Book
author	string	ITSO
descri...	string	WESB
quantity	int	10
customerId	string	999999

Figure 12-51 Enter values for the order operation in the Integration Test Client

30. Click **Continue**.
31. Select the **WebSphere ESB Server v6.0** server as deployment location and click **Finish**.
32. The test should finish returning a confirmationId (Figure 12-52 on page 407).



Invoke (LogMessageMediation:order)
Started
Invoke (LogMessageMediation:order)
Request (LogMessageMediation --> BookOrderServiceImport:
Response (LogMessageMediation <-- BookOrderServiceImpor
Return (LogMessageMediation:order)
Stopped

Module:	MessageLoggerSample1Module
Component:	LogMessageMediation
Interface:	BookOrderService
Operation:	order

Return parameters:

Name	Type	Value
confirmationId	String	6

Figure 12-52 Test result

33. Stop the WebSphere Enterprise Service Bus server to release the lock on the EsbLogMedDB database.
34. Check the Cloudscape database to ensure the message is logged. This can be done using a utility called **cview.bat** which is available in the directory:

```
<WID_INSTALL>/runtimes/bi_v6/cloudscape/bin/embedded
```

35. Run the utility cview.bat. Figure 12-53 shows the Cview utility startup.

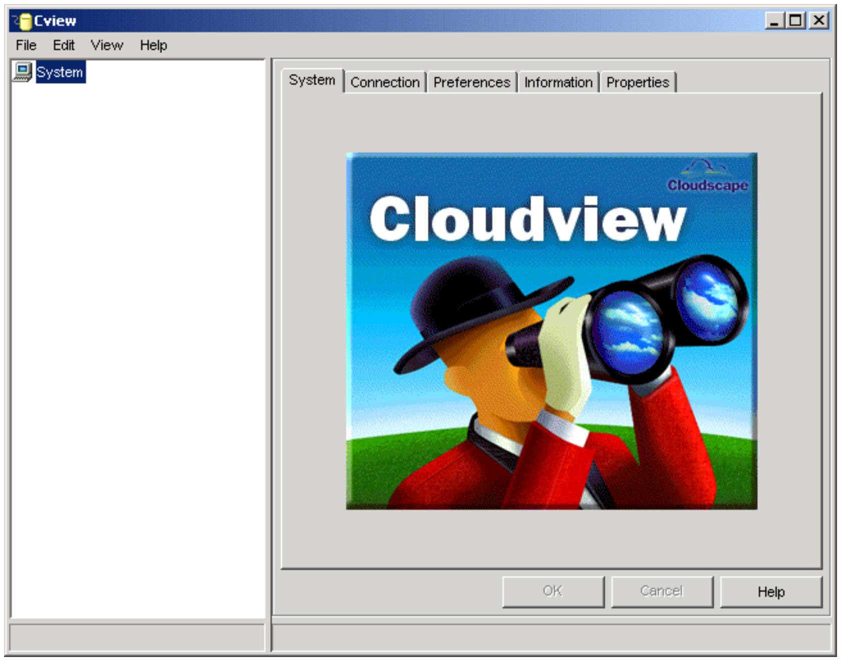
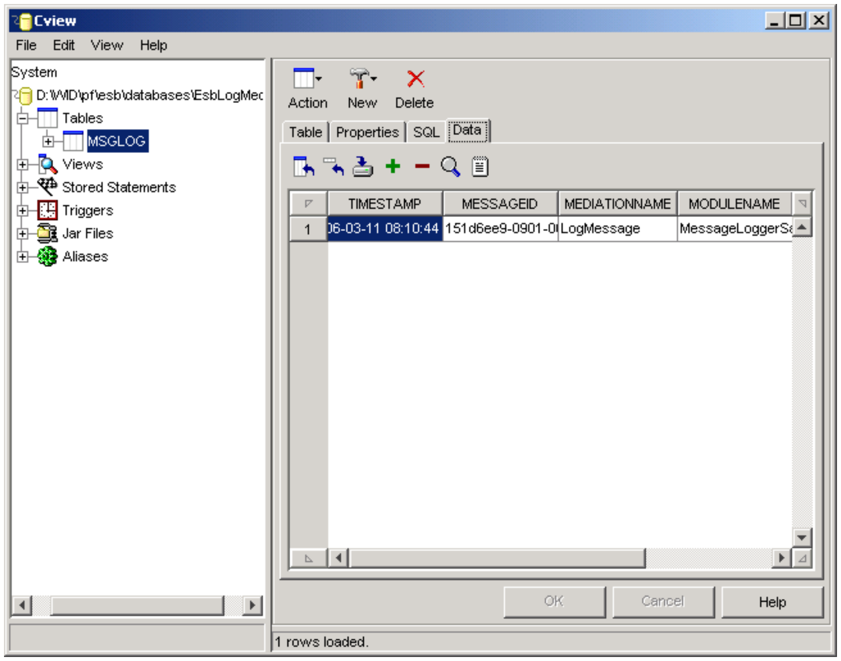


Figure 12-53 Cview utility

36. Click **File -> Open** and open the Cloudscape database **EsbLogMedDB** which is in the directory <WID_INSTALL>/pf/esb/databases.
37. This will open the database. Expand **Tables** and select **MSGLOG.**
38. Click on the **Data** tab to show the records in the table (Figure 12-54).



	TIMESTAMP	MESSAGEID	MEDIATIONNAME	MODULENAME
1	)6-03-11 08:10:44	151d6ee9-0901-0(	LogMessage	MessageLoggerS

1 rows loaded.

Figure 12-54 CloudScape database MSGLOG table

39. Select the message in the **MESSAGE** column and click on the **Text Editor** icon (Figure 12-55).

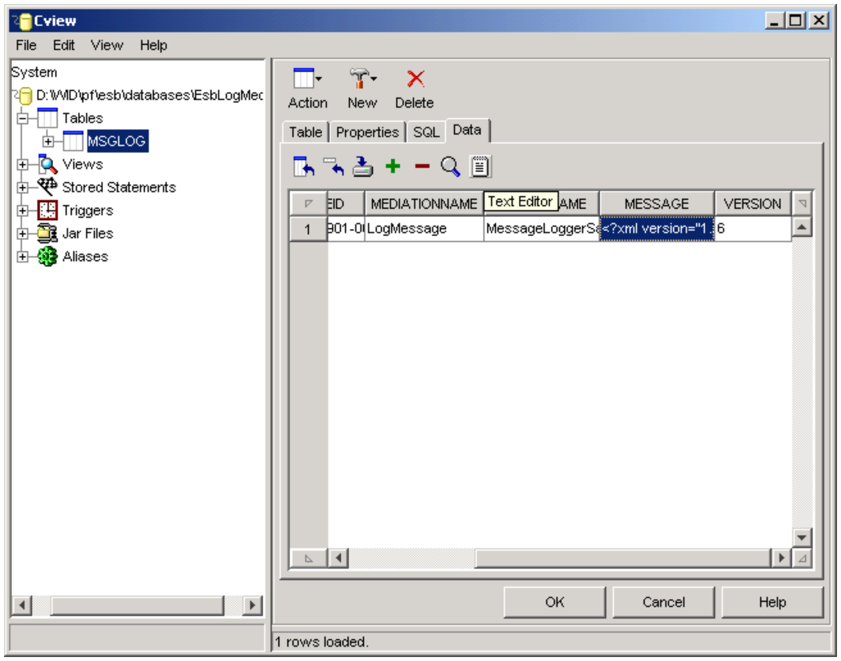
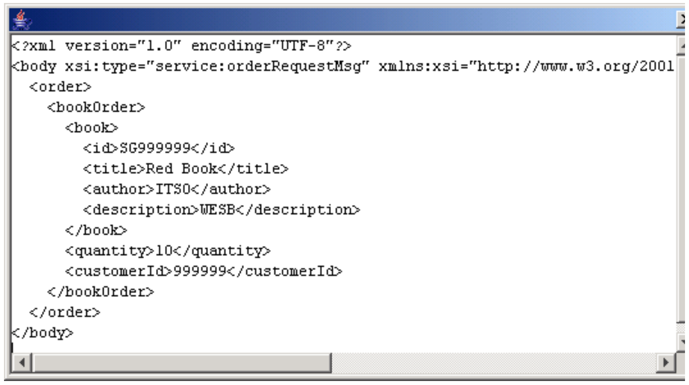

Cview
File Edit View Help
System
D:\WID\pf\esb\databases\EsbLogMed
Tables
MSGLOG
Views
Stored Statements
Triggers
Jar Files
Aliases
Action New Delete
Table Properties SQL Data
EID MEDIATIONNAME Text Editor AME MESSAGE VERSION
1 901-0 LogMessage MessageLoggerS <?xml version="1 6
OK Cancel Help
1 rows loaded.

Figure 12-55 Text Editor to view message

40. You should see the contents of the message logged to the database (Figure 12-56).



```
<?xml version="1.0" encoding="UTF-8"?>
<body xsi:type="service:orderRequestMsg" xmlns:xsi="http://www.w3.org/2001
  <order>
    <bookOrder>
      <book>
        <id>SG999999</id>
        <title>Red Book</title>
        <author>ITSO</author>
        <description>WESB</description>
      </book>
      <quantity>10</quantity>
      <customerId>999999</customerId>
    </bookOrder>
  </order>
</body>
```

Figure 12-56 Message logged in the database

41. Close the Cview tool.

42. Once the testing is complete remove the projects from the server.

Congratulations you have successfully built and tested a mediation module that uses a Message Logger primitive.

12.5 Stop mediation primitive

This sample demonstrates how the stop mediation primitive is used to stop a flow, by consuming messages sent from an output or fault terminal of another mediation primitive.

The Stop mediation primitive stops the flow through a mediation flow component. If connected to a fault terminal of another primitive it will stop the flow and suppress exceptions.

This sample involves:

- Building a mediation module containing an import with a Web service binding.
- Implementing the request flow using the Message Filter and Stop mediation primitives.
- Testing the module using the Integration Test Client.

The completed sample invokes a profile creation Web service, which returns a profile ID. The mediation module is used to only send valid profiles to the Web service while ignoring invalid profiles.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `StopSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Rename Mediation1 to `FilterValidProfilesMediation` (Figure 12-57).

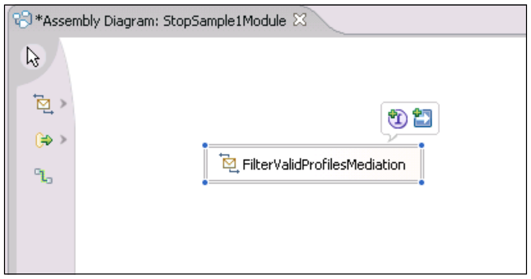
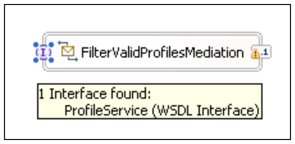

*Assembly Diagram: StopSample1Module
FilterValidProfilesMediation

Figure 12-57 FilterValidProfilesMediation

4. Right-click **FilterValidProfilesMediation** and select **Add -> Interface**.
5. Select the **ProfileService** interface and click **OK**.
6. Right-click **FilterValidProfilesMediation** and select **Add -> Reference**.
7. Choose the **ProfileService** interface and click **OK** (Figure 12-58).



1 Interface found:
ProfileService (WSDL Interface)

Figure 12-58 FilterValidProfiles Mediation Flow Component

8. In the BookOrderResources project expand Web Service Ports. Drag and drop **ProfileServiceSOAP** onto the assembly editor. The Component Creation dialog will open. Select **Import with Web Service Binding** and click **OK**.
9. Wire FilterValidProfilesMediation to the import (Figure 12-59).

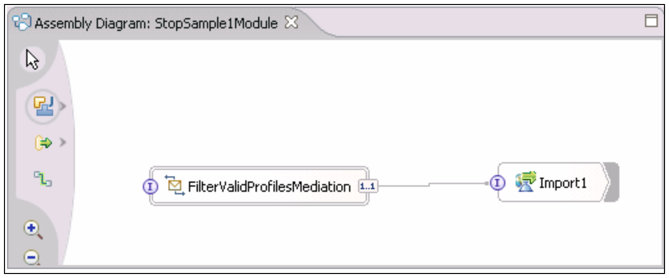
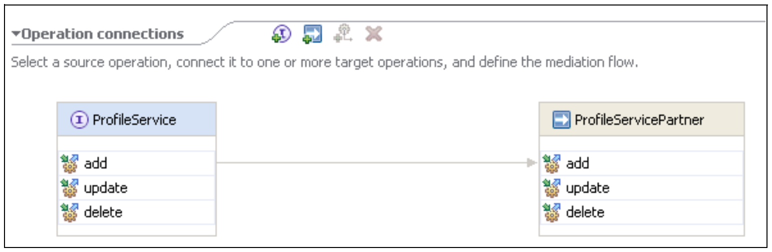

Assembly Diagram: StopSample1Module
FilterValidProfilesMediation
1..1
Import1

Figure 12-59 Stop Sample Module

10. Save the module.

Now we have a mediation flow component with an interface and a reference lets generate the mediation flow.

11. Right-click **FilterValidProfilesMediation** and select **Generate Implementation**.
12. Click **OK** to store the implementation in the default folder. This opens the mediation flow editor.
13. In the Operation connections section of this view, wire the **add** method on the ProfileService interface to the **add** method of the ProfileServicePartner reference. (Figure 12-60)


Operation connections

Select a source operation, connect it to one or more target operations, and define the mediation flow.

Figure 12-60 Operation connections

14. Add a Message Filter mediation primitive to the request flow using .
15. Rename the Message Filter mediation primitive to `ValidProfileFilter`.
16. Wire the request flow (Figure 12-61 on page 413).
    a. Wire **ProfileService_add_Input** to the **in** terminal of ValidProfileFilter.
    b. Wire the **default** terminal of ValidProfileFilter to **ProfileServicePartner_add_Callout**.

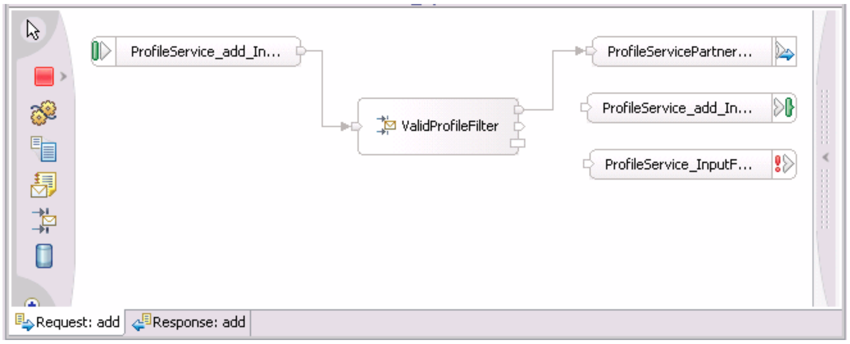
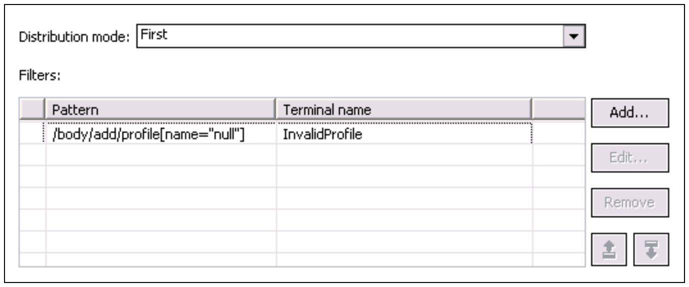


Figure 12-61 Stop sample mediation flow

17. Right-click **ValidProfileFilter** and select **Add Output Terminal**.

18. Name the terminal `InvalidProfile`. Click **OK**.
19. Click on the new terminal and select the **Details** tab in the Properties view.
20. Click the **Add** button.
21. Fill in the filter properties (Figure 12-62).
    a. Set Pattern to `/body/add/profile[name=”null”]`.
    b. Select **InvalidProfile** from the Terminal name drop down menu.
    c. Click **Finish**.


Distribution mode: First

Filters:

Pattern	Terminal name
/body/add/profile[name="null"]	InvalidProfile

Figure 12-62 InvalidProfile output terminal

22. Add a Stop mediation primitive to the request flow using 🟥.
23. Wire the InvalidProfile **out** terminal of ValidProfileFilter to the Stop mediation primitive (Figure 12-63).

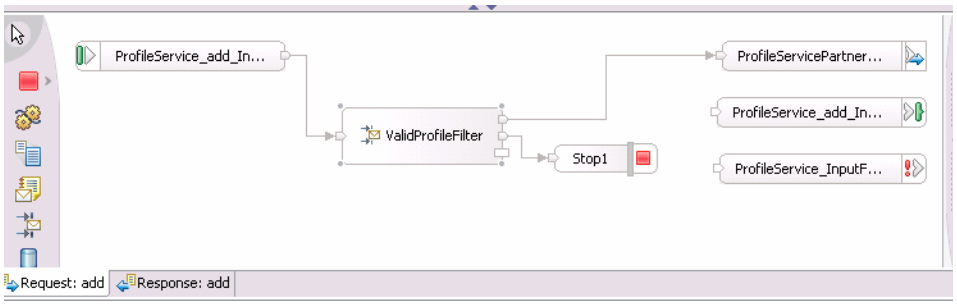


Figure 12-63 Completed Mediation Flow for Stop Sample 1

24. Click the **Response** tab in the mediation flow editor.
25. Wire the response flow (Figure 12-64 on page 415).
    a. Wire **ProfileServicePartner_add_CalloutResponse** to **ProfileService_add_InputResponse**.
    b. Wire **ProfileServicePartner_CalloutFault** to **ProfileService_InputFault**.

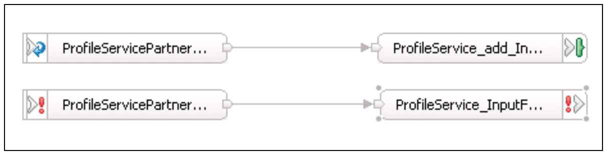
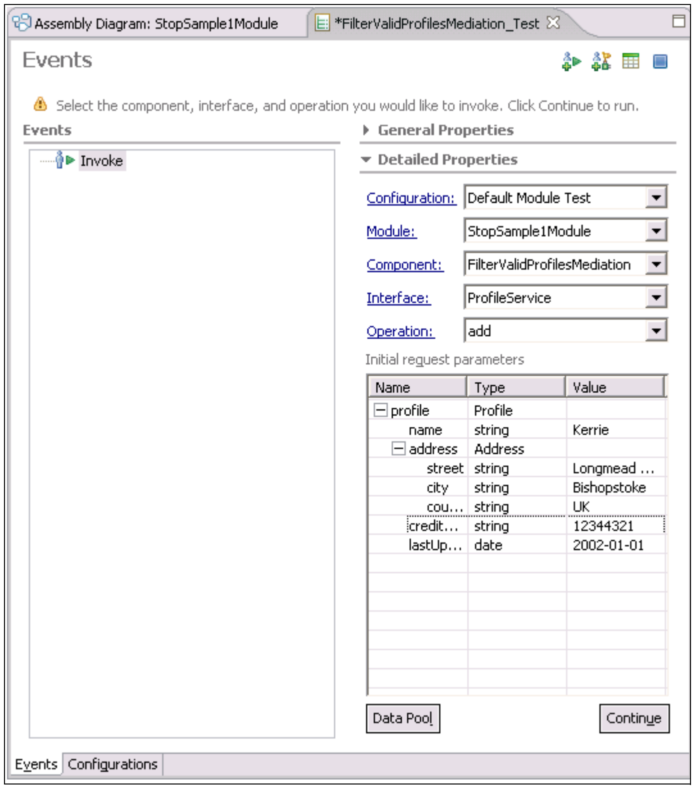


Figure 12-64 Stop sample response flow

26. Save the mediation flow and mediation module.

To test the flow we use the Integration Test Client. This will show the flow being stopped by the Stop mediation primitive.

27. Deploy the module and the Web service to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add ProfileServiceEAR and StopSample1ModuleApp.
    d. Click **Finish**.
28. Right-click the **StopSample1Module** project and select **Test -> Test Module**.
29. Set the Component to **FilterValidProfilesMediation**.
30. Select the **add** operation from the Operation drop down menu.
31. Enter values into the request parameters table (Figure 12-65 on page 416).


Assembly Diagram: StopSample1Module
*FilterValidProfilesMediation_Test
Events

Select the component, interface, and operation you would like to invoke. Click Continue to run.

Events
Invoke
General Properties
Detailed Properties

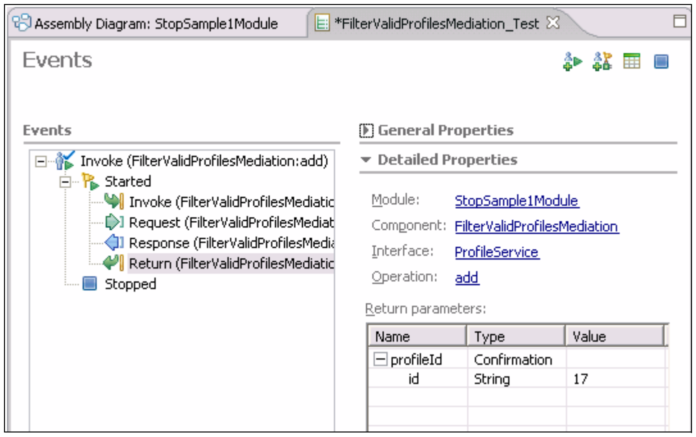
Configuration: Default Module Test
Module: StopSample1Module
Component: FilterValidProfilesMediation
Interface: ProfileService
Operation: add

Initial request parameters

Name	Type	Value
profile	Profile	
name	string	Kerrie
address	Address	
street	string	Longmead ...
city	string	Bishopstoke
cou...	string	UK
credit...	string	12344321
lastUp...	date	2002-01-01

Figure 12-65 Entering request parameters

32. Click **Continue**.
33. Select the **WebSphere ESB Server v6.0** server as deployment location and click **Finish**.
34. You will see a confirmation ID returned from the Web service (Figure 12-66 on page 417).


Assembly Diagram: StopSample1Module
*FilterValidProfilesMediation_Test
Events
Events
Invoke (FilterValidProfilesMediation:add)
Started
Invoke (FilterValidProfilesMediatic
Request (FilterValidProfilesMediat
Response (FilterValidProfilesMedi
Return (FilterValidProfilesMediatic
Stopped
General Properties
Detailed Properties

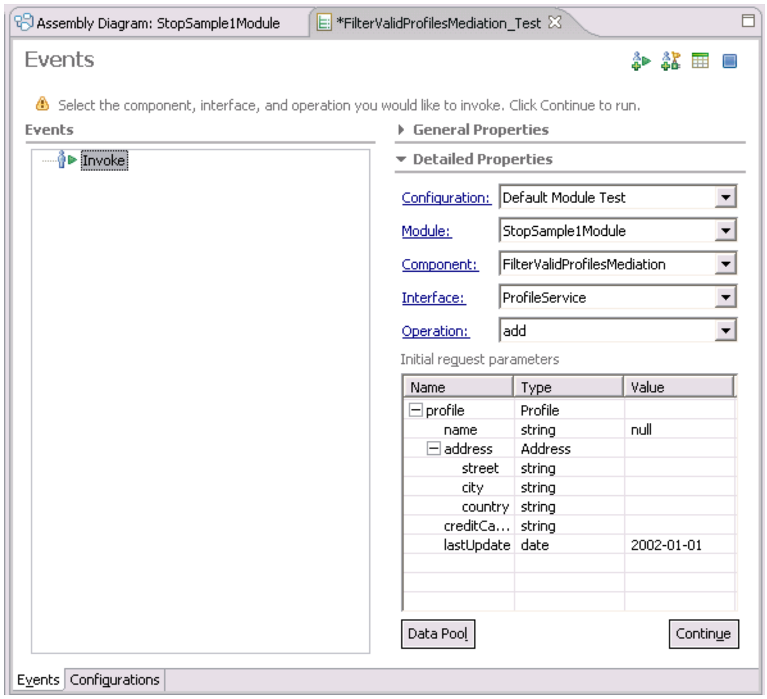
Module: StopSample1Module
Component: FilterValidProfilesMediation
Interface: ProfileService
Operation: add

Return parameters:

Name	Type	Value
profileId	Confirmation	
id	String	17

Figure 12-66 Returned confirmation ID

35. Re-run the test but this time enter `null` into the name field (Figure 12-67 on page 417).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: StopSample1Module
Component: FilterValidProfilesMediation
Interface: ProfileService
Operation: add

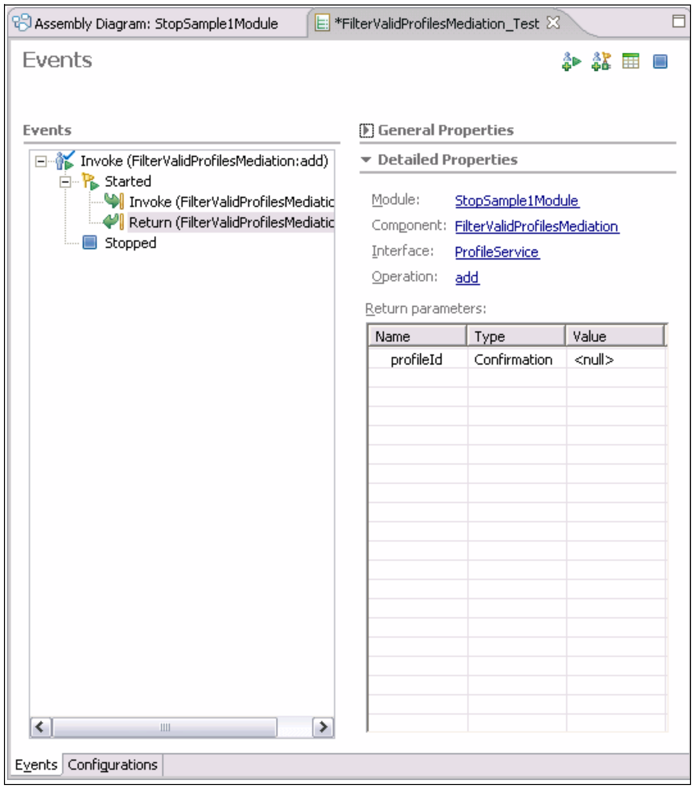
Initial request parameters

Name	Type	Value
profile	Profile	
name	string	null
address	Address	
street	string	
city	string	
country	string	
creditCa...	string	
lastUpdate	date	2002-01-01

Figure 12-67 Entering a null name into the Integration Test Client

36. Click **Continue**.

37. You will see the mediation flow is stopped and the profileId is null (Figure 12-68 on page 418). The ProfileService Web service was not invoked.


Assembly Diagram: StopSample1Module
*FilterValidProfilesMediation_Test
Events
Events
General Properties
Detailed Properties
Events
Configurations

- Invoke (FilterValidProfilesMediation:add)
  - Started
    - Invoke (FilterValidProfilesMediatic
    - Return (FilterValidProfilesMediatic
  - Stopped

Module: StopSample1Module
Component: FilterValidProfilesMediation
Interface: ProfileService
Operation: add

Return parameters:

Name	Type	Value
profileId	Confirmation	<null>

Figure 12-68 Null profileId

38. Once the testing is complete remove the projects from the server.

Congratulations, you have successfully built and tested a mediation module that uses a Stop mediation primitive.

Note: In this sample, if the default terminal of ValidProfileFilter is not connected to a Stop mediation primitive the flow would behave in the same way. However, using the Stop mediation primitive clarifies the intention of the flow. It also removes the *output terminal not connected* warnings from WebSphere Integration Developer.

Note: The Stop mediation primitive can also be wired to the fail terminal of another primitive. In this case, if an exception occurs, the Stop primitive will suppress it and the flow will stop cleanly.

12.6 Fail mediation primitive

This sample demonstrates how the Fail mediation primitive is used to raise a FailFlowException.

The Fail mediation primitive stops the flow through a mediation flow component and raises an exception. Any existing transaction will be rolled back and the module throws a FailFlowException.

This sample involves:

- Building a mediation module containing an import with a Web service binding.
- Implementing the request flow using the Message Filter and Fail mediation primitives.
- Testing the module using the Integration Test Client.

The completed sample invokes a profile creation Web service, which returns a profile ID. The mediation module is used to only send valid profiles to the Web service while raising an exception when a profile is invalid.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `FailSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Rename Mediation1 to `CatchInvalidProfilesMediation` (Figure 12-69).

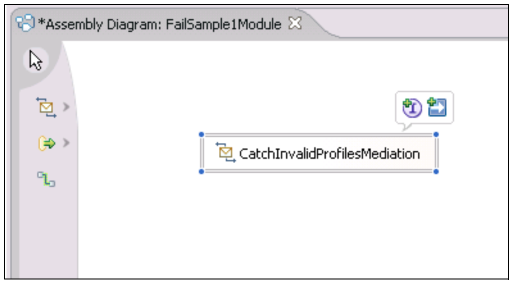
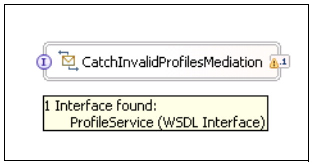

*Assembly Diagram: FailSample1Module
CatchInvalidProfilesMediation

Figure 12-69 CatchInvalidProfilesMediation flow component

4. Right-click **CatchInvalidProfilesMediation** and select **Add -> Interface**.
5. Select the **ProfileService** interface and click **OK**.
6. Right-click **CatchInvalidProfilesMediation** and select **Add -> Reference**.
7. Select the **ProfileService** interface and click **OK** (Figure 12-70).



1 Interface found:
ProfileService (WSDL Interface)

Figure 12-70 Added interface and reference

8. In the BookOrderResources project expand Web Service Ports. Drag and drop **ProfileServiceSOAP** onto the palette. The Component Creation dialog will open. Select **Import with Web Service Binding** and click **OK**.
9. Wire CatchInvalidProfilesMediation to the import (Figure 12-59).

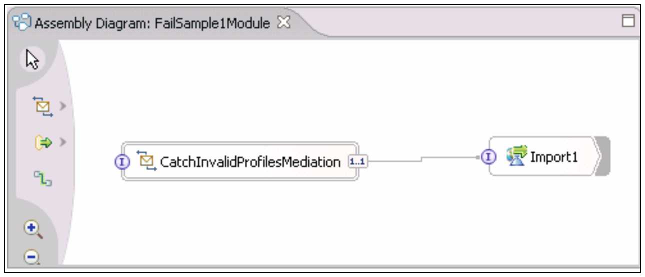

Assembly Diagram: FailSample1Module
CatchInvalidProfilesMediation
Import1

Figure 12-71 Wired Import

10. Save the module.

Now we have a mediation flow component with an interface and a reference, let's generate the mediation flow.

11. Right-click **CatchInvalidProfilesMediation** and select **Generate Implementation**.
12. Click **OK** to store the implementation in the default folder. This opens the Mediation Flow Editor.
13. In the Operation connections section of this view, wire the **add** method on the ProfileService interface to the **add** method of the ProfileServicePartner reference (Figure 12-72 on page 421).

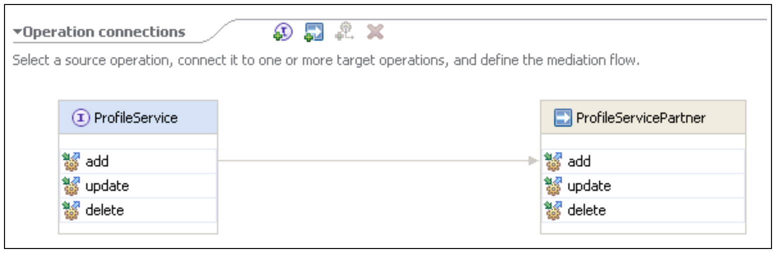

Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
ProfileService
add
update
delete
ProfileServicePartner
add
update
delete

Figure 12-72 Operation connections

14. Add a Message Filter mediation primitive to the request flow using .
15. Rename the Message Filter mediation primitive to `InvalidProfileFilter`.
16. Wire the request flow (Figure 12-73).
    a. Wire **ProfileService_add_Input** to the **in** terminal of InvalidProfileFilter.
    b. Wire the **default** terminal of InvalidProfileFilter to **ProfileServicePartner_add_Callout**.

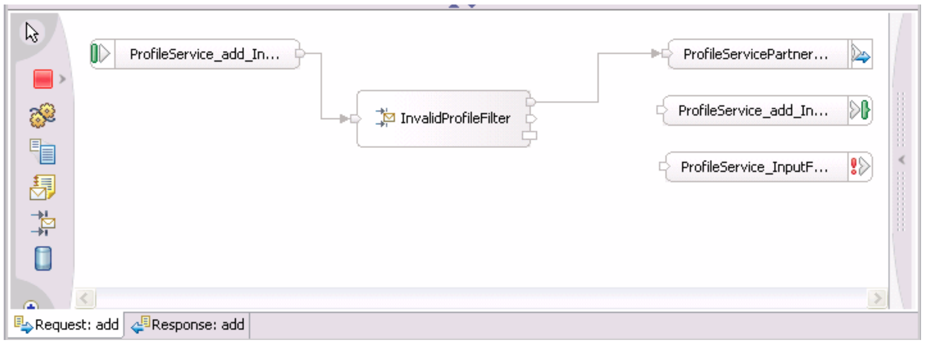
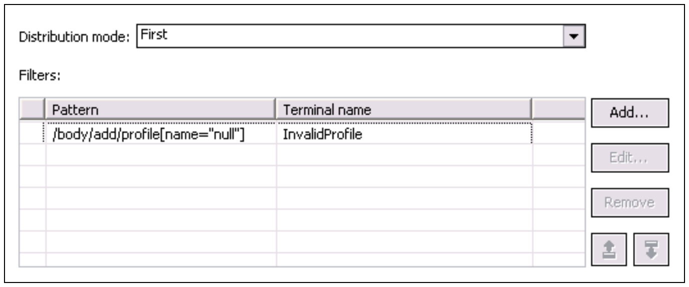


Figure 12-73 Request flow

17. Right-click **InvalidProfileFilter** and select **Add Output Terminal**. Name the terminal `InvalidProfile`. Click **OK**.

18. Click on the new InvalidProfile terminal and select the **Details** tab in the Properties view.
19. Click on the **Add** button.
20. Fill in the filter properties (Figure 12-74)
    a. Set Pattern to `/body/add/profile[name=”null”]`.
    b. Select **InvalidProfile** from the Terminal name drop down menu.
    c. Click **Finish**.


Distribution mode: First

Filters:

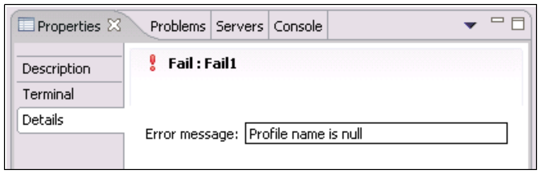
Pattern	Terminal name
/body/add/profile[name="null"]	InvalidProfile

Figure 12-74 InvalidProfile output terminal

21. Add a Fail mediation primitive to the request flow using ❗.

> **Note:** The Fail mediation primitive is found by expanding the Stop mediation primitive icon.

22. Click on the Fail mediation primitive and in the Properties view, select the **Details** tab.
23. Set the error message to `Profile name is null` (Figure 12-75 on page 422).



Fail : Fail1

Error message: Profile name is null

Figure 12-75 Setting the error message on a fail mediation primitive

24. Wire the **InvalidProfile** terminal to the **in** terminal of the Fail mediation primitive. (Figure 12-76 on page 423)

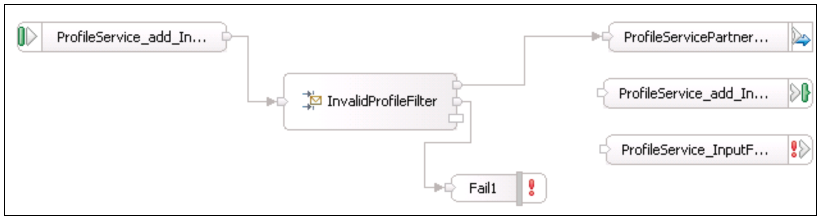


Figure 12-76 Completed mediation flow for the Fail sample

25. Click the **Response** tab in the mediation flow editor.

26. Wire the response flow (Figure 12-64 on page 415).

   a. Wire **ProfileServicePartner_add_CalloutResponse** to **ProfileService_add_InputResponse**.

   b. Wire **ProfileServicePartner_CalloutFault** to **ProfileService_InputFault**.

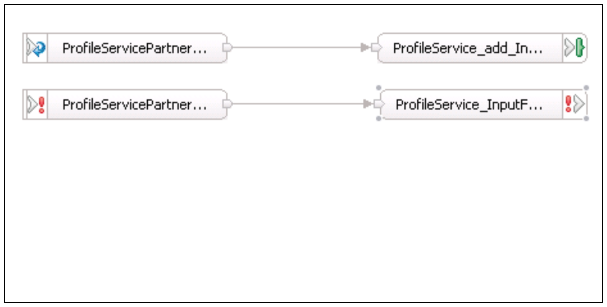


Figure 12-77 Response flow

27. Save the mediation flow and the mediation module.

To test the flow we use the Integration Test Client to check that when the name field in the profile is set to null a FailFlowException is raised.

28. Deploy the module and the Web service to the server.

   a. Switch to the Servers view.

   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.

   c. Add ProfileServiceEAR and FailSample1ModuleApp.

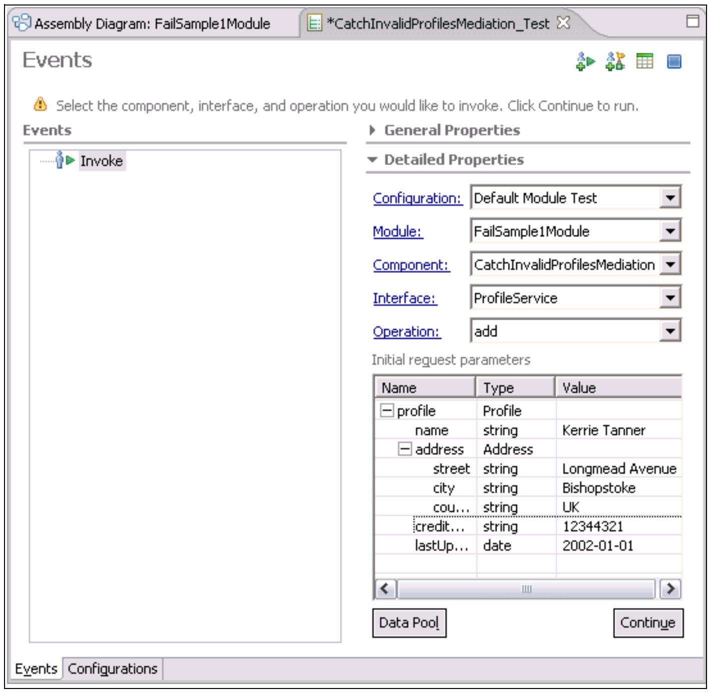
   d. Click **Finish**.

29. Right-click the **FailSample1Module** and select **Test** -> **Test Module**.

30. Set the Component to **CatchInvalidProfilesMediation**.

31. Select the **add** operation from the Operation drop down menu.

32. Enter values into the request parameters table (Figure 12-78 on page 424).


Assembly Diagram: FailSample1Module
*CatchInvalidProfilesMediation_Test
Events
Select the component, interface, and operation you would like to invoke. Click Continue to run.
Events
Invoke
General Properties
Detailed Properties
Data Pool
Continue
Events
Configurations

Configuration: Default Module Test
Module: FailSample1Module
Component: CatchInvalidProfilesMediation
Interface: ProfileService
Operation: add

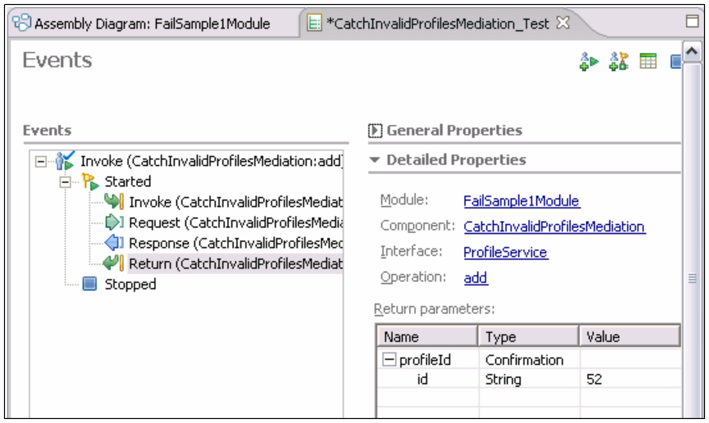
Initial request parameters

Name	Type	Value
profile	Profile	
name	string	Kerrie Tanner
address	Address	
street	string	Longmead Avenue
city	string	Bishopstoke
cou...	string	UK
credit...	string	12344321
lastUp...	date	2002-01-01

Figure 12-78 Enter values into request parameters

33. Click **Continue**.

34. You will see a profileId returned by the Web service (Figure 12-79 on page 424).


Assembly Diagram: FailSample1Module
*CatchInvalidProfilesMediation_Test
Events
Events
Invoke (CatchInvalidProfilesMediation:add)
Started
Invoke (CatchInvalidProfilesMediat
Request (CatchInvalidProfilesMedi
Response (CatchInvalidProfilesMed
Return (CatchInvalidProfilesMediat
Stopped
General Properties
Detailed Properties

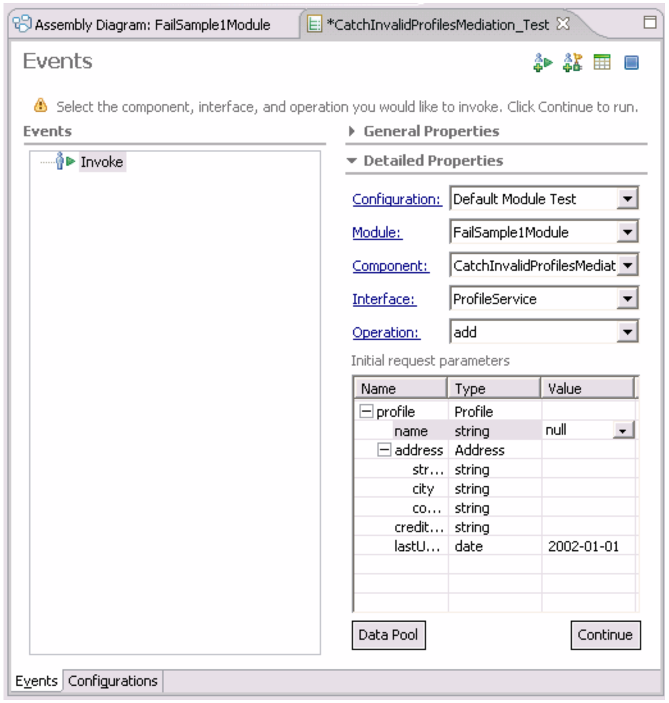
Module: FailSample1Module
Component: CatchInvalidProfilesMediation
Interface: ProfileService
Operation: add

Return parameters:

Name	Type	Value
profileId	Confirmation	
id	String	52

Figure 12-79 Returned confirmation ID

35. Re-run the test, but this time enter `null` into the name field (Figure 12-80 on page 425).



Select the component, interface, and operation you would like to invoke. Click Continue to run.

Configuration: Default Module Test
Module: FailSample1Module
Component: CatchInvalidProfilesMediat
Interface: ProfileService
Operation: add

Initial request parameters

Name	Type	Value
profile	Profile	
name	string	null
address	Address	
str...	string	
city	string	
co...	string	
credit...	string	
lastU...	date	2002-01-01

Figure 12-80 Entering a null name in the Integration Test Client

36. Click **Continue**.

37. You will see a FailFlowException is raised, containing the message we defined (Figure 12-81 on page 426).

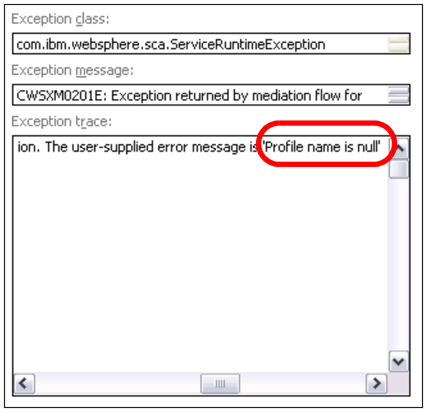

Exception class:
com.ibm.websphere.sca.ServiceRuntimeException
Exception message:
CWSXM0201E: Exception returned by mediation flow for
Exception trace:
ion. The user-supplied error message is 'Profile name is null'

Figure 12-81 Testing the Fail mediation primitive

38.Once the testing is complete remove the projects from the server.

Congratulations, you have successfully created and tested a module containing a Fail mediation primitive.

Note: If a transaction is in progress, when the mediation flow is stopped by a fail mediation primitive, the transaction is rolled back and the FailFlowException is stored in the transient context.

12.7 Custom mediation primitive

This sample demonstrates Custom mediation primitives. It also introduces the use of the mediation flow's correlation context.

The Custom mediation primitive allows the user to implement their own mediate method using Java. The Custom mediation, like the other primitives, receives a Service Message Object and returns a Service Message Object. It can be used to perform tasks that cannot be performed by using the other mediation primitives.

This sample involves:

- Building a mediation module containing an import.
- Implementing the request flow using an XSL Transform mediation primitive.
- Implementing the response flow using a Custom mediation primitive.
- Implementing the Custom mediation primitive using the JavaMail™ API and a properties file.

- Testing the module using the Integration Test Client.

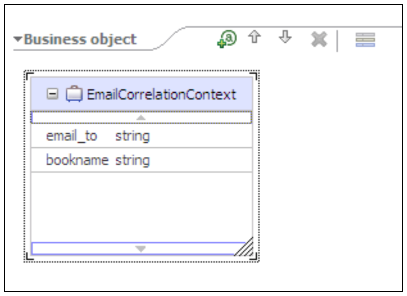
The completed sample emulates a book order Web service, which returns a confirmation ID. The mediation module is used to persist the book order in the correlation context and then once a response is received, a copy of the order along with the confirmation ID is sent in an email.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `CustomSample1Module` and click **Next.**
   c. In the Select Required Libraries dialog check the **BookOrderResources** library and click **Finish**.
2. Open the module in the Assembly Editor by double clicking .
3. Create a properties file to store the SMTP server host name.
   a. In the Business Integration view, right-click the mediation module and select **New** → **Other** from the context menu.
   b. Expand **Simple**, select **File** and click **Next**.
   c. Select **CustomSample1Module** as the parent folder, name the file `smtp_host.properties` and click **Finish**.
   d. The text editor will open. Type the SMTP server hostname key/value using the format `smtp_host=<servername>`. Replace <servername> with the hostname of your SMTP server (Example 12-3).
   e. Save and close the file.

Example 12-3 smtp server hostname properties file

```
smtp_host=NA.relay.ibm.com
```

4. In the Business Integration view, expand CustomSample1Module, right-click on **Data Types** and select **New -> Business Object**.
5. Set the Name to `EmailCorrelationContext` and click **Finish**.
6. Create the business object (Figure 12-82).
   f. Click the Add Attribute button .
   g. Set the name of the attribute to `email_to`.
   h. Click the Add Attribute button .
   i. Set the name of the attribute to `bookname`.
   j. Save and close the business object.



email_to	string
bookname	string

Figure 12-82 Correlation context business object

7. In the assembly diagram right-click **Mediation1** and select **Add -> Interface**.
8. Select the **BookOrderService** interface and click **OK**.
9. Right click **Mediation1** and select **Add -> Reference**.
10. Select the **BookOrderService** interface and click **OK**.
11. Add an import into the assembly diagram using .
12. Wire Mediation1 to Import1. This will create a matching interface on the import.
13. Right-click Import1 and choose **Generate Binding** → **SCA Binding** from the context menu.

Note: We will not use this import to invoke any service provider. We need the Import to be present to successfully test the mediation flow component.

14. In the assembly diagram, right-click **Mediation1** and select **Generate Implementation**. Click **OK** to choose the default folder for implementation location.
15. In the Operation connections panel wire the **order** operation from the BookOrderService interface to the **order** operation on the BookOrderServicePartner reference.
16. In the Mediation flow editor add an XSL Transformation mediation primitive to the request flow using .
17. Wire the request flow (Figure 12-83).
    a. Wire **BookOrderService_order_Input** to the **in** terminal of XSLTransformation1.
    b. Wire the **out** terminal of XSLTransformation1 to **BookOrderServicePartner_order_Callout**.

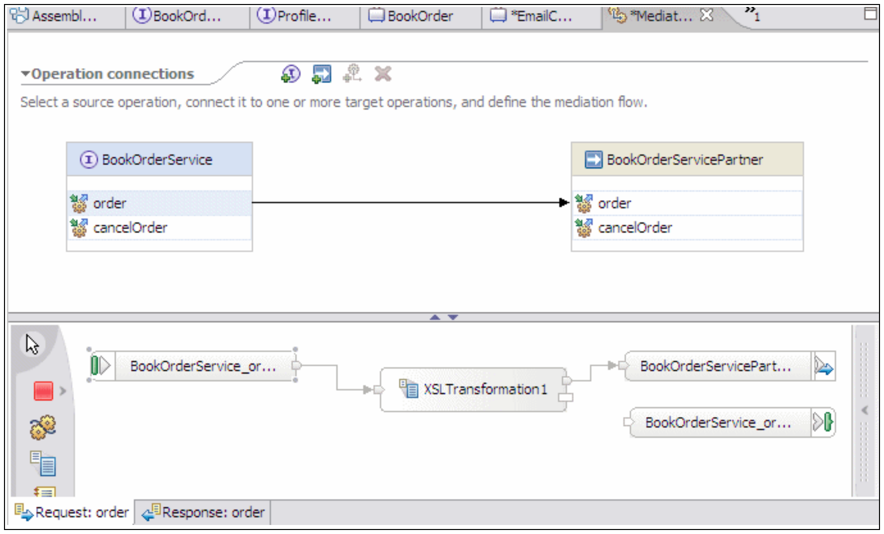
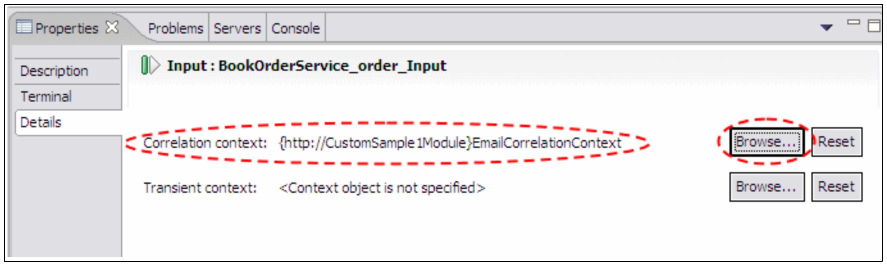


Figure 12-83 Operation connections and XSLT primitive

18. Specify the Correlation context business object (Figure 12-84).
    a. Click the **BookOrderService_order_Input**.
    b. Click the **Details** tab in the Properties view.
    c. On the Correlation Context line click **Browse**.
    d. Select the **EmailCorrelationContext** business object.
    e. Click **OK.**



Correlation context: {http://CustomSample1Module}EmailCorrelationContext

Transient context: <Context object is not specified>

Figure 12-84 Correlation context object

19. Create a new XSL Transformation mapping.
    a. Click the **XSLTransformation1** primitive to select it.
    b. Click the **Details** tab in the Properties view.
    c. In the Root drop-down menu select /.

d. Click the **New** button.

e. Leave the input and output messages unchanged (orderRequestMsg) and click **Finish**.

20. Define the XSL Mapping (Figure 12-85 on page 431).

a. Propagate the headers element by selecting the **headers** element on the source SMO, then selecting the **headers** element on the target SMO.

b. Right-click the **headers** element in the source panel and select **Match Mapping** from the context menu.

c. Propagate the body element by selecting the **body** element on the source SMO, then selecting the **body** element on the target SMO.

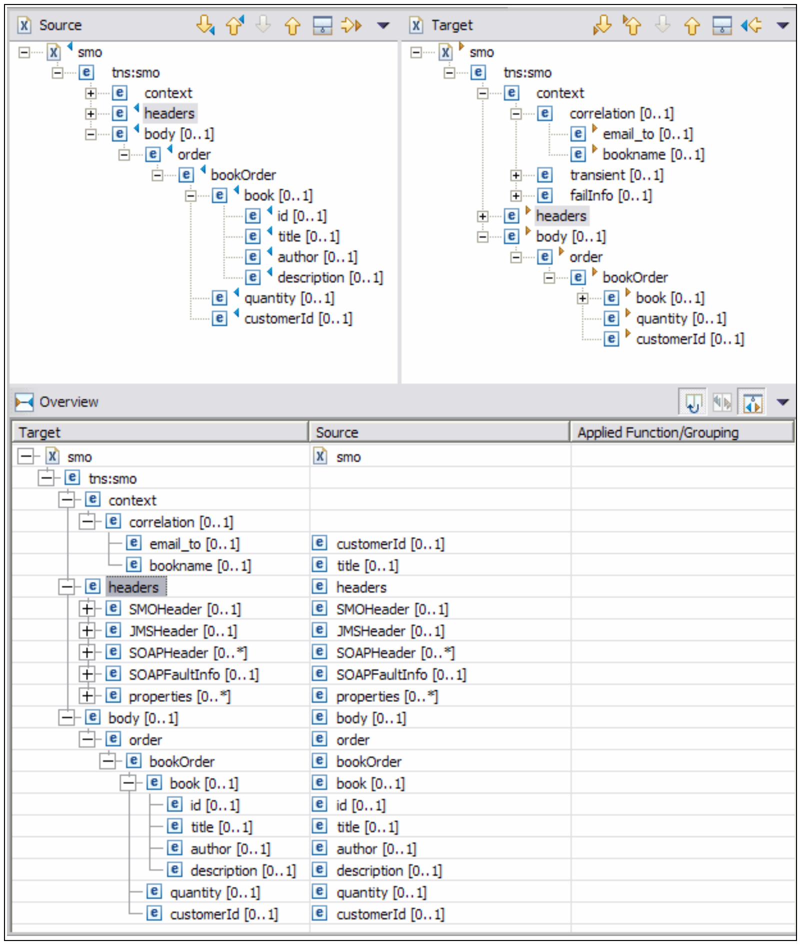
d. Right-click the **body** element in the source panel and select **Match Mapping** from the context menu.

e. Store the required values in the correlation context.

i. Drag **body -> order -> bookOrder -> book -> title** in the source SMO to **context -> correlation -> bookname** in the target SMO.

ii. Drag **body -> order -> bookOrder -> customerId** in the source SMO to **context -> correlation -> email_to** in the target SMO.

Note: In this sample we will use the customer's e-mail address as the customer id.


Source

```
smo
  tns:smo
    context
    headers
    body [0..1]
      order
        bookOrder
          book [0..1]
            id [0..1]
            title [0..1]
            author [0..1]
            description [0..1]
          quantity [0..1]
          customerId [0..1]
```

Target

```
smo
  tns:smo
    context
      correlation [0..1]
        email_to [0..1]
        bookname [0..1]
      transient [0..1]
      failInfo [0..1]
    headers
    body [0..1]
      order
        bookOrder
          book [0..1]
          quantity [0..1]
          customerId [0..1]
```

Overview

Target	Source	Applied Function/Grouping
smo	smo	
tns:smo		
context		
correlation [0..1]		
email_to [0..1]	customerId [0..1]	
bookname [0..1]	title [0..1]	
headers	headers	
SMOHeader [0..1]	SMOHeader [0..1]	
JMSHeader [0..1]	JMSHeader [0..1]	
SOAPHeader [0..*]	SOAPHeader [0..*]	
SOAPFaultInfo [0..1]	SOAPFaultInfo [0..1]	
properties [0..*]	properties [0..*]	
body [0..1]	body [0..1]	
order	order	
bookOrder	bookOrder	
book [0..1]	book [0..1]	
id [0..1]	id [0..1]	
title [0..1]	title [0..1]	
author [0..1]	author [0..1]	
description [0..1]	description [0..1]	
quantity [0..1]	quantity [0..1]	
customerId [0..1]	customerId [0..1]	

Figure 12-85 SMO Mapping

21. Save and close the XSL transformation.
22. Click the **Regenerate XSL** button on the mediation flow editor's Details tab in the Properties view.

At this point we have finished building our request flow and we have stored the information we need in the correlation context. We can now start building the response flow.

23. Click the **Response** tab in the Mediation Flow editor
24. Add a Custom mediation primitive to the response flow using .
25. Wire the response flow (Figure 12-86).

a. Wire **BookOrderServicePartner_order_CalloutResponse** to the **in** terminal of CustomMediation1.

b. Wire the **out** terminal of CustomMediation1 to **BookOrderService_order_InputResponse**.

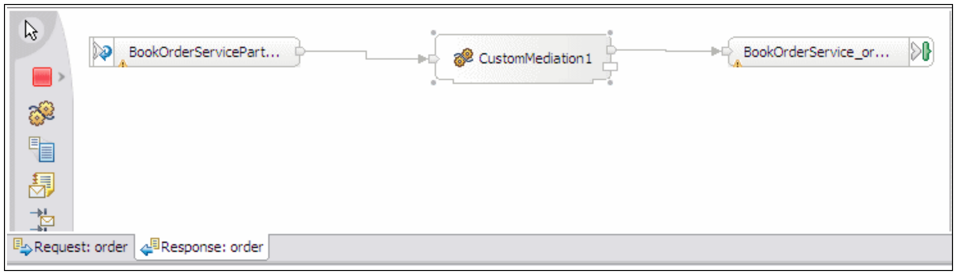
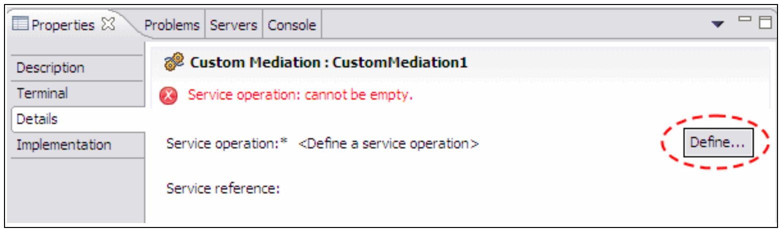


Figure 12-86 Response flow

26. Click on CustomMediation1 and select the **Details** tab in the Properties view.

27. Click **Define** (Figure 12-87).



Custom Mediation : CustomMediation1

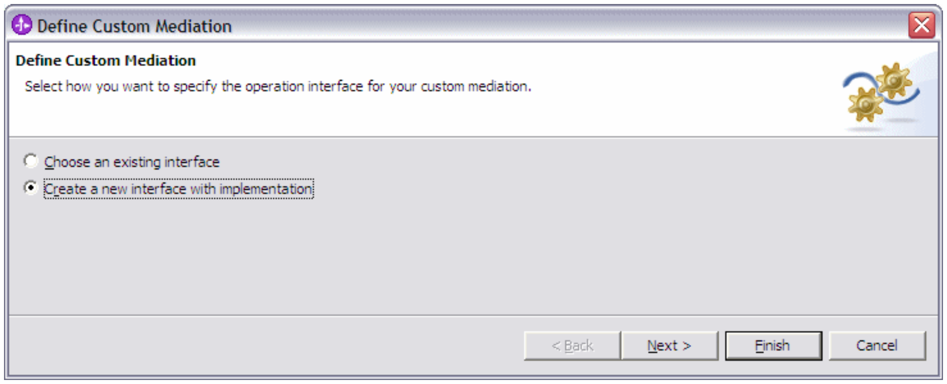
Service operation: cannot be empty.

Service operation:* <Define a service operation>

Service reference:

Figure 12-87 Define custom mediation

28. Make sure **Create a new interface with implementation** is selected and click **Next** (Figure 12-88 on page 433).

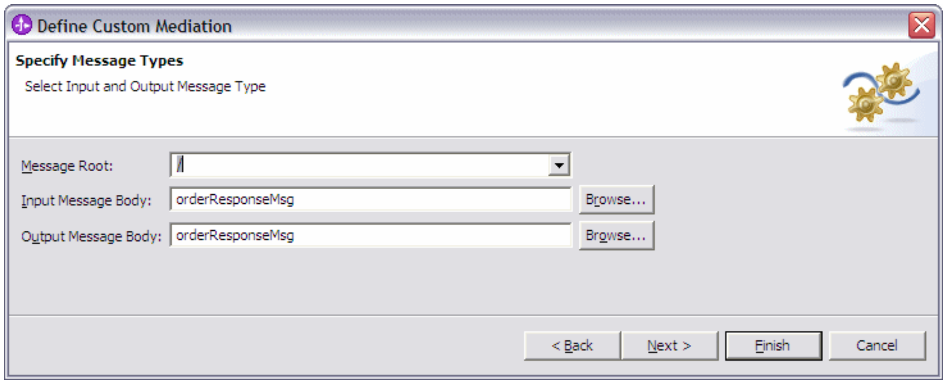

Define Custom Mediation

Select how you want to specify the operation interface for your custom mediation.

( ) Choose an existing interface
(•) Create a new interface with implementation

Figure 12-88 Define custom mediation

29. At the Specify Message Types dialog select **/** as the message root and leave the message types unchanged (Figure 12-89). Click **Next**.

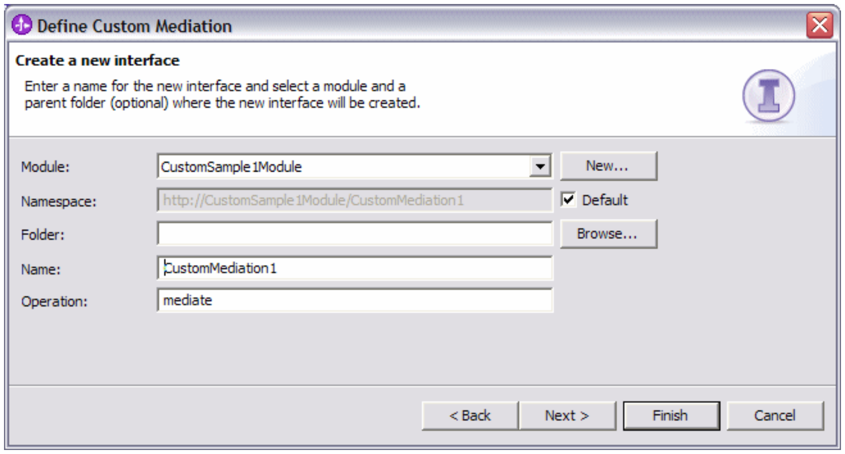

Specify Message Types

Select Input and Output Message Type

Message Root: /
Input Message Body: orderResponseMsg
Output Message Body: orderResponseMsg

Figure 12-89 Specify message types

30. At the Create a new interface screen verify the module name is CustomSample1Module and click **Next**. You may specify a different folder in which to create the new interface (Figure 12-90).


Define Custom Mediation

Create a new interface

Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.

Module: CustomSample1Module

Namespace: http://CustomSample1Module/CustomMediation1 [✓ Default]

Folder:

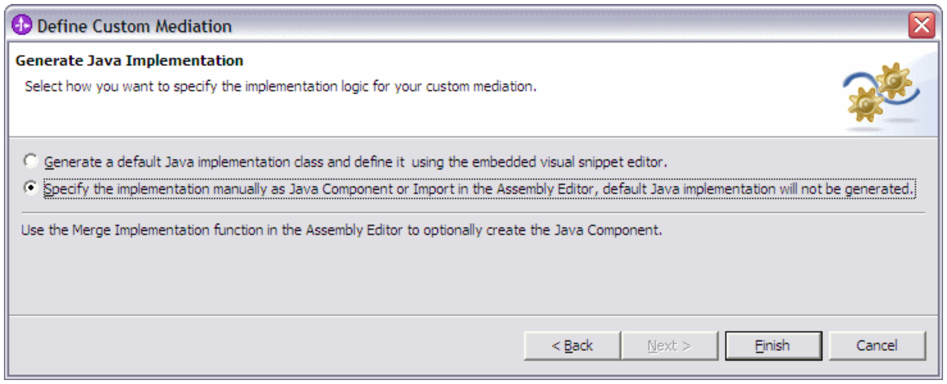
Name: CustomMediation1

Operation: mediate

Figure 12-90 Create new interface

31. At the Generate Java Implementation screen choose **Specify the implementation manually as Java Component or Import in the Assembly Editor, default Java implementation will not be generated** (Figure 12-91). Click **Finish**.

> **Note:** This creates a new Java SCA component which appears in the assembly diagram.


Define Custom Mediation

Generate Java Implementation

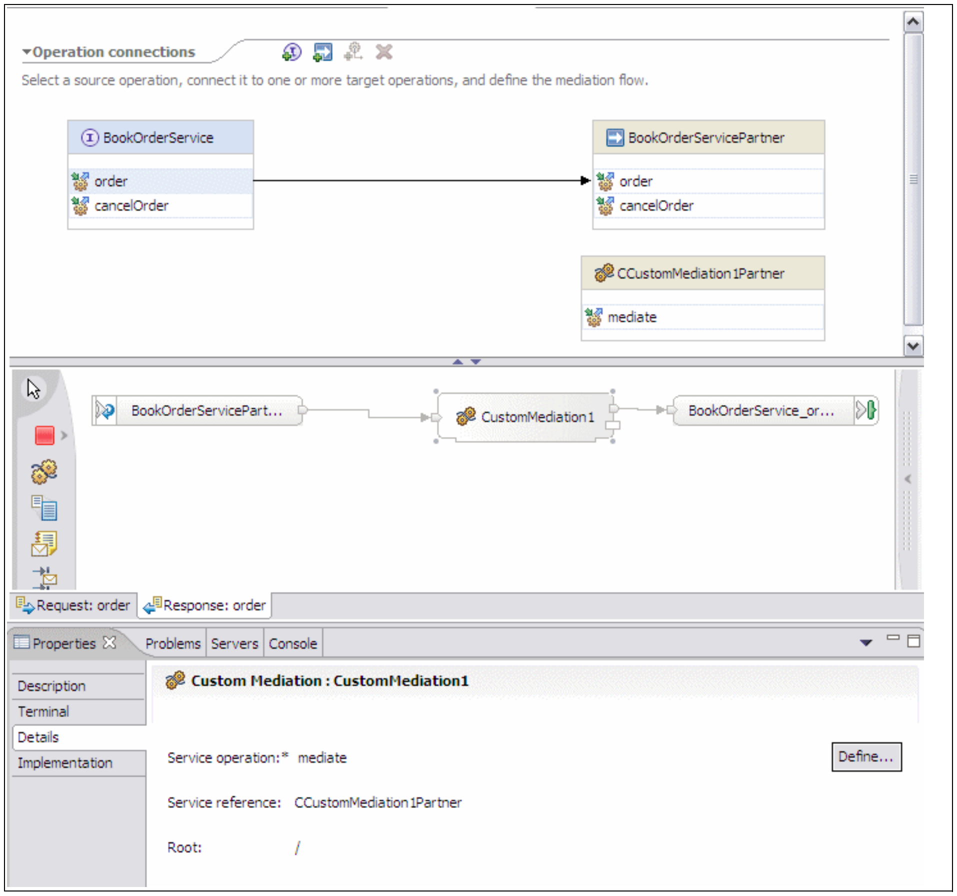
Select how you want to specify the implementation logic for your custom mediation.

- ○ Generate a default Java implementation class and define it using the embedded visual snippet editor.
- ● Specify the implementation manually as Java Component or Import in the Assembly Editor, default Java implementation will not be generated.

Use the Merge Implementation function in the Assembly Editor to optionally create the Java Component.

Figure 12-91 Generate Java implementation

32. Save the mediation flow. Review your mediation flow and compare it with Figure 12-92.



Custom Mediation : CustomMediation1

Service operation:* mediate

Service reference: CCustomMediation1Partner

Root: /

Figure 12-92 Mediation flow

33. Save the assembly diagram. In the assembly diagram, you will see there is an error on the Mediation1 mediation flow component. This is because you just generated a new Java SCA component in your module and your assembly needs to be synchronized to reflect this.
    a. Right-click **Mediation1** and select **Merge Implementation**.
    b. Click **OK** on the next two dialog boxes.
    c. Save the assembly diagram.
    d. Now you should have no errors and a new SCA component will be wired to your mediation flow component. This new component will implement the custom mediation logic (Figure 12-93).

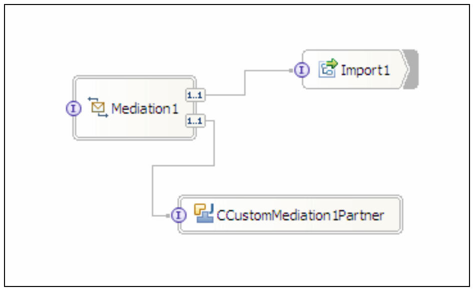


Figure 12-93 New SCA component created

34. Right-click the new Java SCA component **CustomMediation1Partner** and select **Generate Implementation** from the context menu.
35. In the Generate Implementation window click **OK**.
36. The Java editor opens up. Insert the required imports as in Example 12-4.

> **Note:** The entire code for this Java class can be found in the additional material supplied with this redbook in the following location:
>
> `\MediationPrimitives\Resources\Custom\CustomMediation1Partner_1Impl.java`

Example 12-4 Class imports

```
import java.util.Date;
import java.util.Properties;
import java.util.ResourceBundle;

import javax.mail.*;
import javax.mail.internet.*;

import commonj.sdo.DataObject;
import com.ibm.websphere.sca.ServiceManager;
import com.ibm.websphere.sibx.smobo.ContextType;
import com.ibm.websphere.sibx.smobo.ServiceMessageObject;
```

37. Locate the mediate method and replace its implementation with the code in Example 12-5

Example 12-5 mediate method implementation

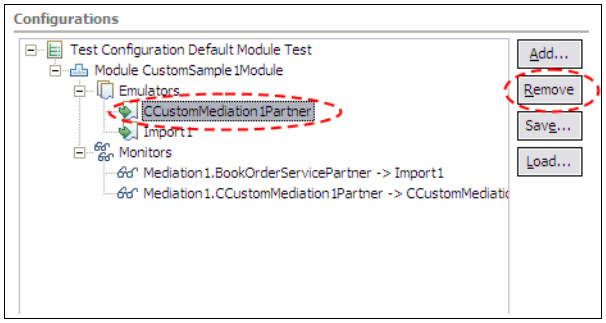
```
public DataObject mediate(DataObject input1) {

   /*
   * get the smtp server host name from the properties file
   */
   ResourceBundle bundle = ResourceBundle.getBundle("smtp_host");
```

```
String smtp_host = bundle.getString("smtp_host");

/*
* get the book name and email address from the correlation context
*/
ServiceMessageObject smo = (ServiceMessageObject)input1;
ContextType ctx = smo.getContext();
DataObject correlationCtx = (DataObject)ctx.getCorrelation();
String bookname = correlationCtx.getString("bookname");
String email_to = correlationCtx.getString("email_to");

/*
* get the confirmation ID from the response message
* note that the 'order' operation will respond with an
* 'orderResponse' message from which we can retrieve
* the confirmationId string
*/
DataObject body = (DataObject)smo.getBody();
DataObject response = (DataObject)body.getDataObject("orderResponse");
String confirmation = response.getString("confirmationId");

/*
* create properties for the mail session get
* the mail session instance and send the email
* using the data items retrieved from the smo
*/
Properties props = new Properties();
props.put("mail.smtp.host", smtp_host);
Session session = Session.getInstance(props, null);
session.setDebug(true);
try {
   MimeMessage msg = new MimeMessage(session);
   msg.setFrom(new InternetAddress("service@itsobooks.com"));
   msg.addRecipients(Message.RecipientType.TO,email_to);
   msg.setSubject("Your book order confirmation");
   msg.setSentDate(new Date());
   StringBuffer buffer =
      new StringBuffer("Hello from your bookstore!\n\n");
   buffer.append("Your book: ").append(bookname)
         .append(", has been ordered.\n");
   buffer.append("Your order confirmation number is: ")
         .append(confirmation);
   msg.setText(buffer.toString());
   Transport.send(msg);
} catch (MessagingException mex) {
   System.out.println(mex.getMessage());
}
return input1;
```

```
}
```

38. Save the Java implementation.
39. Save the assembly diagram.
40. Deploy the module to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add CustomSample1ModuleApp.
    d. Click **Finish**.
41. Right-click **Mediation1** on the assembly diagram and select **Test Component**.
42. Click on the **Configurations** tab of the Integration Test Client remove the CustomMediation1Partner emulator (Figure 12-94).

When testing a component, the Integration Test Client tries to emulate every reference out of the component but this particular one is the custom mediation logic we just defined in the mediate method.

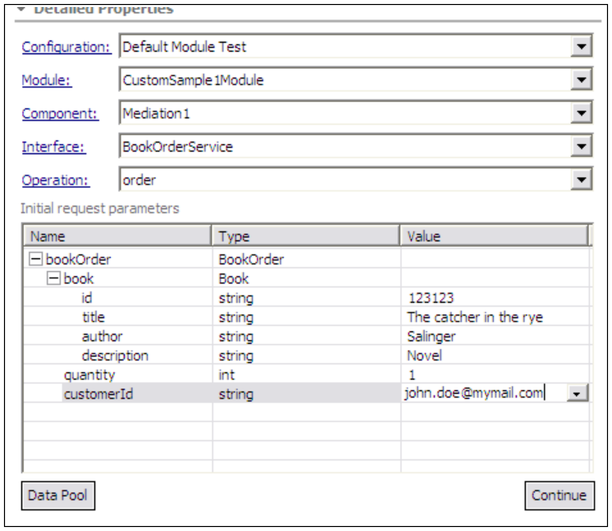


```
Test Configuration Default Module Test
  Module CustomSample1Module
    Emulators
      CCustomMediation1Partner
      Import1
    Monitors
      Mediation1.BookOrderServicePartner -> Import1
      Mediation1.CCustomMediation1Partner -> CCustomMediatic
```

Figure 12-94 Remove mediation partner emulator

43. Click the **Events** tab select **order** from the Operation drop down menu.
44. Populate the bookOrder data items (Figure 12-95).

Note: Make sure customerId is a valid E-mail address.



Detailed Properties

Configuration: Default Module Test
Module: CustomSample1Module
Component: Mediation1
Interface: BookOrderService
Operation: order

Initial request parameters

Name	Type	Value
bookOrder	BookOrder	
book	Book	
id	string	123123
title	string	The catcher in the rye
author	string	Salinger
description	string	Novel
quantity	int	1
customerId	string	john.doe@mymail.com

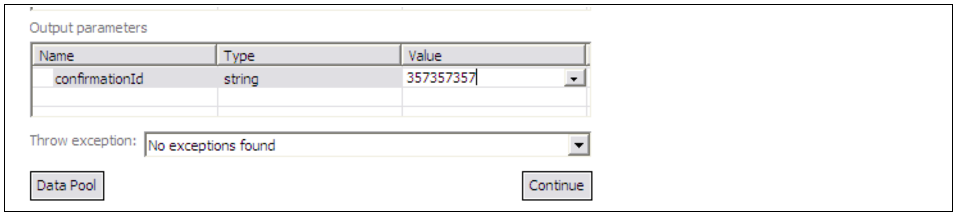
Figure 12-95 Invoking the book order service

45. Click **Continue**.

46. Select **WebSphere ESB Server v6.0** as the Deployment Location and click **Finish**.

You will need to emulate the response.

47. On the Output parameters panel populate confirmationId with a value of your choice (Figure 12-96). Click **Continue**.



Output parameters

Name	Type	Value
confirmationId	string	357357357

Throw exception: No exceptions found

Figure 12-96 Emulate confirmation response

48. At this point the Custom Mediation primitive performs the following tasks:

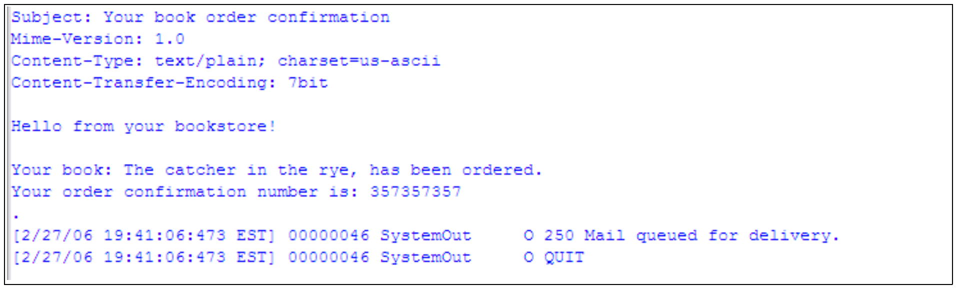
    a. Retrieves information from the properties file.

    b. Retrieves information from the correlation context.

    c. Retrieves the confirmationId from the response message.

    d. Sends an email using the JavaMail API.

e. All interaction with the SMTP server is logged to the Console view. (Figure 12-97). You should also receive an email at the email address you specified.



```
Subject: Your book order confirmation
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Hello from your bookstore!

Your book: The catcher in the rye, has been ordered.
Your order confirmation number is: 357357357
.
[2/27/06 19:41:06:473 EST] 00000046 SystemOut     O 250 Mail queued for delivery.
[2/27/06 19:41:06:473 EST] 00000046 SystemOut     O QUIT
```

Figure 12-97 Mail session in Console view

49. Once the testing is complete remove the project from the server.

Congratulations, You built a request response flow that stores information in the correlation context using an XSL Transformation primitive during the request flow, and uses a Custom mediation primitive on the response flow to retrieve information from various sources and send an email using the JavaMail API.

13 Configuring modules to provide quality of service

This chapter describes how to add quality of service functions to mediation modules. It describes three step-by-step examples

- 13.1, “CEI events” on page 442

  Describes how to add CEI events to a mediation flow, and how to browse CEI events.

- 13.2, “Security” on page 450

  Describes how to apply security to mediation modules.

- 13.3, “Transactions” on page 460

  Describes how to add transactional scopes to mediation modules.

These development examples assume you have configured your WebSphere Integration Developer workspace as described in Chapter 10, “Preparing for the development examples” on page 271.

Each of the development examples in this section can be imported as Project Interchange files (except for the transactions example) from the additional material supplied with this redbook in the `\QualityOfService\Solutions` directory.

13.1 CEI events

Common Event Infrastructure (CEI) provides basic event management services, such as event generation, transmission, persistence, and consumption.

This sample explores the CEI events by specifying event logging in a simple mediation flow, executing the flow and viewing the logged messages in the database.

This sample involves:

- Building a mediation module containing an export with a Web service bindings.
- Implementing the request flow using a Stop mediation primitive.
- Use the Event Monitor to enable CEI events.
- Testing the module using the Integration Test Client.
- Use the CBE Event Browser to view events
- View events in a Cloudscape database.

The completed sample will create a CEI event when the foo method is called on the FooInterface.

1. Create a new mediation module.
   a. In the Business Integration view, right-click and select **New -> Mediation Module**.
   b. Set the Module Name to `CEISample1Module` and click **Finish**.
2. In the Business Integration view, in the CEISample1Module, right-click on **Interfaces** and select **New -> Interface**.
3. Name the interface `FooInterface` and click **Finish**.
4. Create the interface (Figure 13-1 on page 443).
   a. Click on the **Add One Way Operation** button .
   b. Name the operation `foo`.
   c. Click on the Add Input button .
   d. Name the input `dummy` and leave the type as string.

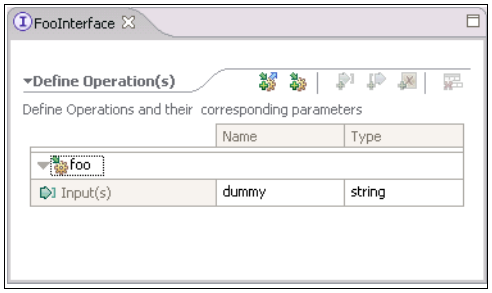


Figure 13-1 FooInterface

5. Save and close the Interface editor.
6. Open the module in the Assembly Editor by double-clicking .
7. Create an export by dragging FooInterface onto the assembly diagram.
8. Choose **Export with WebService Binding** and click **OK**.
9. When asked if the bindings should be created automatically click **Yes**.
10. Choose transport **soap/http** in the Select Transport dialog and click **OK**.
11. Rename Export1 to `FooExport`.
12. Rename the mediation flow component Mediation1 to `FooMediation`.
13. Wire FooExport to FooMediation (Figure 13-2).

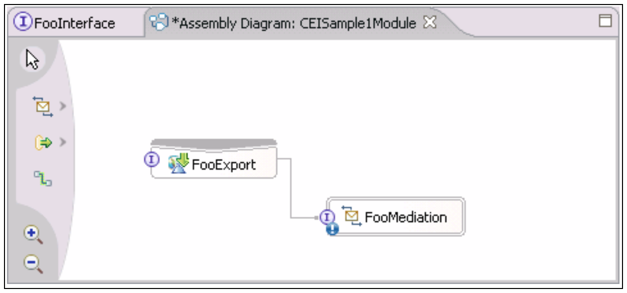


Figure 13-2 CEISample1Module

14. Right-click on **FooMediation** and choose **Generate Implementation** and click **OK** to generate the implementation in the default location.
15. In the mediation flow editor, under Operation connections select the **foo** operation to display the mediation request flow.
16. Add a Stop mediation primitive to the request flow using 🟥.
17. Wire **FooInterface_foo_Input** to the **in** terminal of the Stop (Figure 13-3 on page 444).

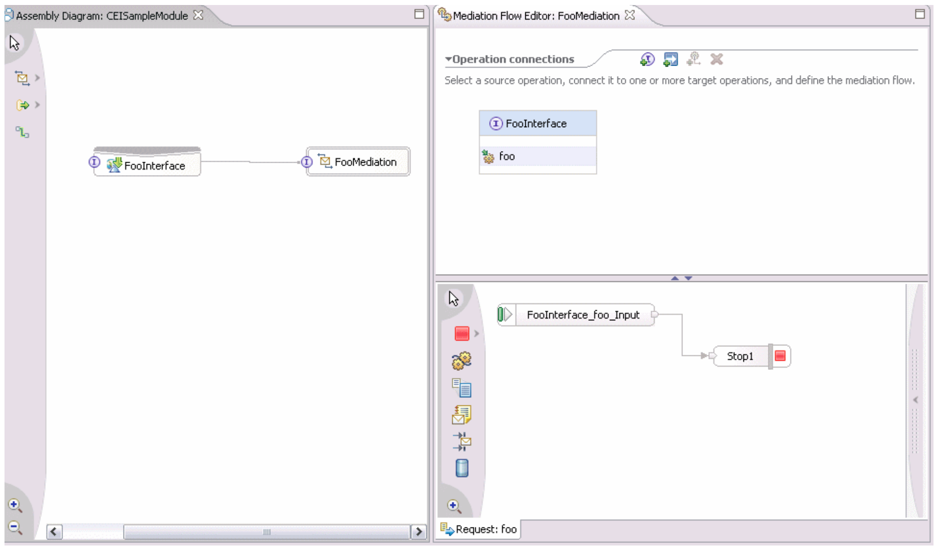

Assembly Diagram: CEISampleModule
FooInterface
FooMediation
Mediation Flow Editor: FooMediation
Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
FooInterface
foo
FooInterface_foo_Input
Stop1
Request: foo

Figure 13-3 Minimal mediation flow for the CEI sample

18. Save and close the Mediation Flow editor.

Now that we have a mediation flow we are to ready to specify a event that will be created and logged.

19. In the assembly diagram, select **FooMediation**
20. In the Properties view, select the **Details** tab.
21. Expand **FooInterface** and select the **foo** operation (Figure 13-4 on page 445).

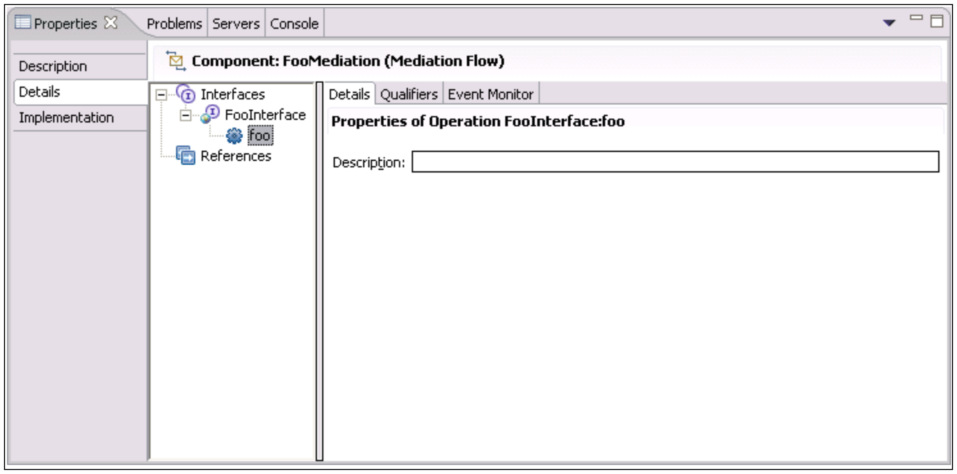
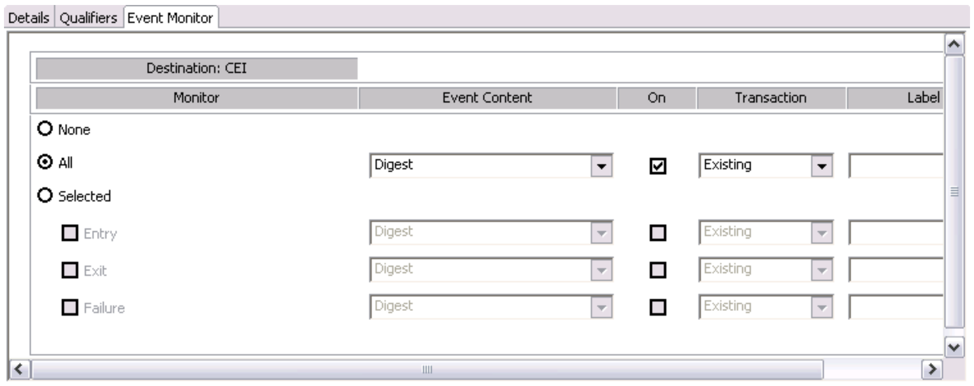


Figure 13-4 Foo operation in properties view

In WebSphere Integration Developer you have to specify which operations (associated with elements of the assembly editor, such as imports, exports, mediation flow components and Java components) should create events. After you selected an operation applicable for creating custom events you have to change the setting from *None* either to *All* or to *Selected*, to specify which of the predefined events should be generated at runtime.

22. Select the **Event Monitor** tab and change the radio button from **None** to **All** (Figure 13-5).

Note: When you save you will notice that the mediation component in the assembly editor shows a little yellow flag in order to indicate that there is a custom event specified.



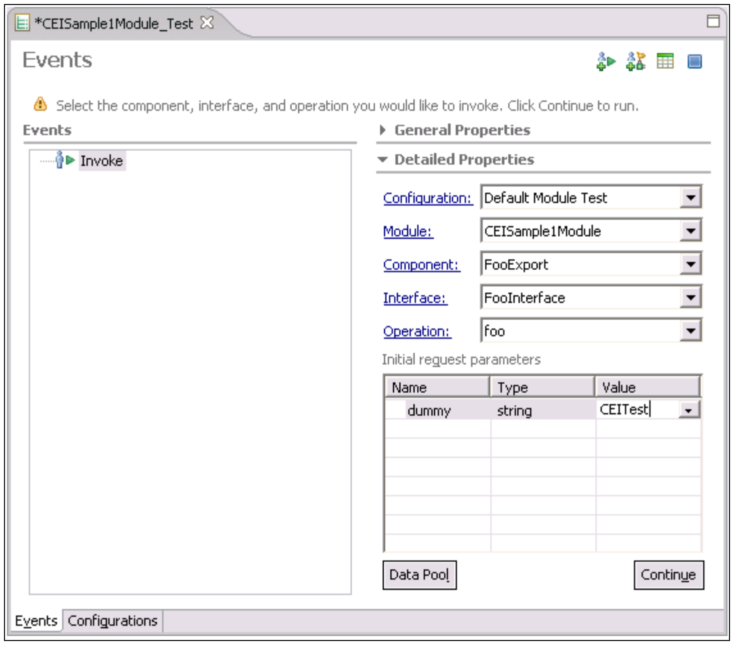
Destination: CEI				
Monitor	Event Content	On	Transaction	Label
○ None				
◉ All	Digest	☑	Existing	
○ Selected				
☐ Entry	Digest	☐	Existing	
☐ Exit	Digest	☐	Existing	
☐ Failure	Digest	☐	Existing	

Figure 13-5 Event Monitor tab

23. Save the module.

Having specified the event the module can be deployed to the test environment.

24. Deploy the module to the server.
    a. Switch to the Servers view.
    b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
    c. Add CEISample1Module.
    d. Click **Finish**.
25. Right-click on the **CEISample1Module** module and select **Test -> Test Module**.
26. Ensure the Component selected is FooExport.
27. Enter a value for the dummy request parameter (Figure 13-6 on page 447).


*CEISample1Module_Test

Events

Select the component, interface, and operation you would like to invoke. Click Continue to run.

Events

Invoke

General Properties

Detailed Properties

Configuration: Default Module Test

Module: CEISample1Module

Component: FooExport

Interface: FooInterface

Operation: foo

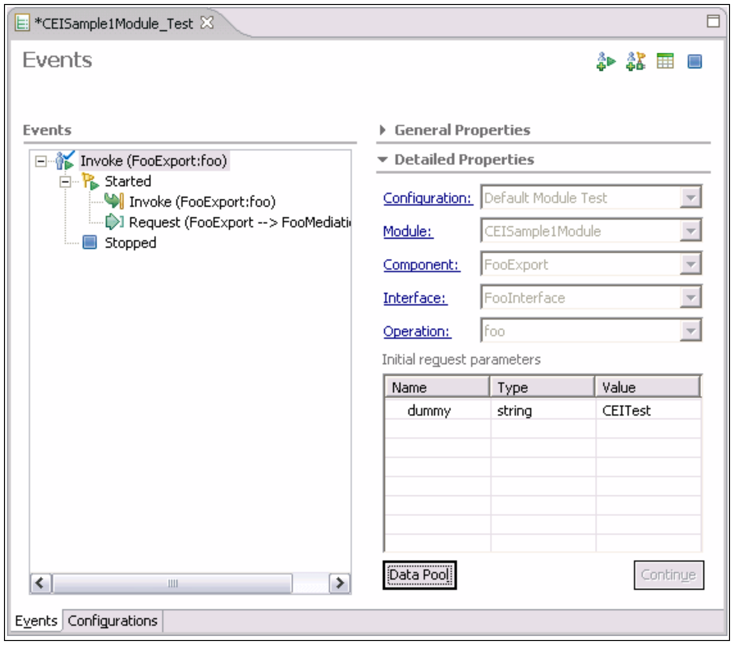
Initial request parameters

Name	Type	Value
dummy	string	CEITest

Figure 13-6 Testing CEISample1Module

28. Click **Continue**.
29. In the Deployment Location dialog select your WebSphere Enterprise Service Bus server and click **Finish**.

The Integration Test Client will execute the mediation flow and result in an output as shown in Figure 13-7 on page 448.



- Invoke (FooExport:foo)
  - Started
    - Invoke (FooExport:foo)
    - Request (FooExport --> FooMediati
  - Stopped

Configuration: Default Module Test
Module: CEISample1Module
Component: FooExport
Interface: FooInterface
Operation: foo

Initial request parameters

Name	Type	Value
dummy	string	CEITest

Figure 13-7 Executing the mediation flow component with the testing facility

Having successfully executed the mediation flow you can now go and find the logged events for this execution. With the installation of WebSphere Integration Developer and the WebSphere Enterprise Service Bus test environment a default CEI repository is created during the installation.

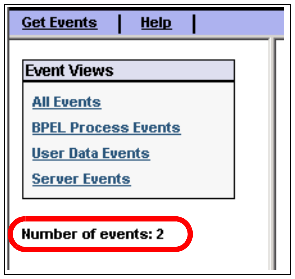
A common way to view CEI events is to use the CBE Event Browser.

30. Start a Web browser and enter the following URL
    `http://localhost:9061/ibm/console/cbebrowser/events`

Attention: The port number 9061 may vary depending on your install of WebSphere Enterprise Service Bus.

The number of events displayed is currently 0.

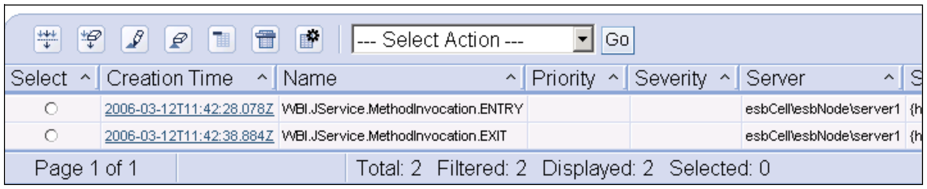
31. Click the **Get Events** button. The number of events should increase to 2, as shown in Figure 13-8 on page 449.



Number of events: 2

Figure 13-8 Number of events

32. In the Event Views box, click **All Events**. This will list the two events (Figure 13-9).

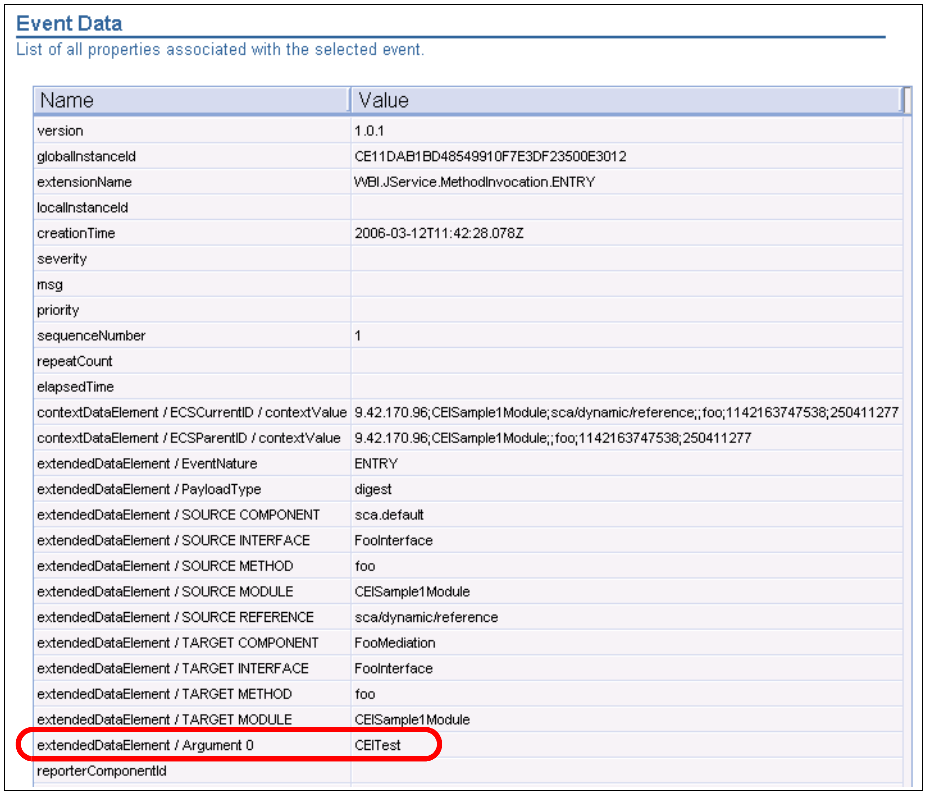


Select	Creation Time	Name	Priority	Severity	Server	S
○	2006-03-12T11:42:28.078Z	WBI.JService.MethodInvocation.ENTRY			esbCell\esbNode\server1	{h
○	2006-03-12T11:42:38.884Z	WBI.JService.MethodInvocation.EXIT			esbCell\esbNode\server1	{h

Page 1 of 1 Total: 2 Filtered: 2 Displayed: 2 Selected: 0

Figure 13-9 All events

33. Click on the first event (using the link in the Creation Time column) to view it. In the Event Data you can see the input data we provided (Figure 13-10 on page 450).


Event Data

List of all properties associated with the selected event.

Name	Value
version	1.0.1
globalInstanceId	CE11DAB1BD48549910F7E3DF23500E3012
extensionName	WBI.JService.MethodInvocation.ENTRY
localInstanceId	
creationTime	2006-03-12T11:42:28.078Z
severity	
msg	
priority	
sequenceNumber	1
repeatCount	
elapsedTime	
contextDataElement / ECSCurrentID / contextValue	9.42.170.96;CEISample1Module;sca/dynamic/reference;;foo;1142163747538;250411277
contextDataElement / ECSParentID / contextValue	9.42.170.96;CEISample1Module;;foo;1142163747538;250411277
extendedDataElement / EventNature	ENTRY
extendedDataElement / PayloadType	digest
extendedDataElement / SOURCE COMPONENT	sca.default
extendedDataElement / SOURCE INTERFACE	FooInterface
extendedDataElement / SOURCE METHOD	foo
extendedDataElement / SOURCE MODULE	CEISample1Module
extendedDataElement / SOURCE REFERENCE	sca/dynamic/reference
extendedDataElement / TARGET COMPONENT	FooMediation
extendedDataElement / TARGET INTERFACE	FooInterface
extendedDataElement / TARGET METHOD	foo
extendedDataElement / TARGET MODULE	CEISample1Module
extendedDataElement / Argument 0	CEITest
reporterComponentId	

Figure 13-10 Event Data showing the input arguments we provided

34.This completes the testing. Remove the project from the server.

Congratulations, you have successfully showed how CEI events can be defined at development time for mediation flow components on an operation level.

13.2 Security

This sample demonstrates how to enabled security within a mediation module.

There are two quality-of-service qualifiers relevant for security in SCA components in WebSphere Enterprise Service Bus. These qualifiers for security are:

- Security permission (the required J2EE role to invoke an operation).
- Security identity (the J2EE role under which the component will be executed - regardless of the invoking J2EE role).

This sample involves:

- Importing a mediation module containing a stand-alone reference and a Java component.
- Enabling security on a Java component.
- Testing security by using an SCA client to access the module.

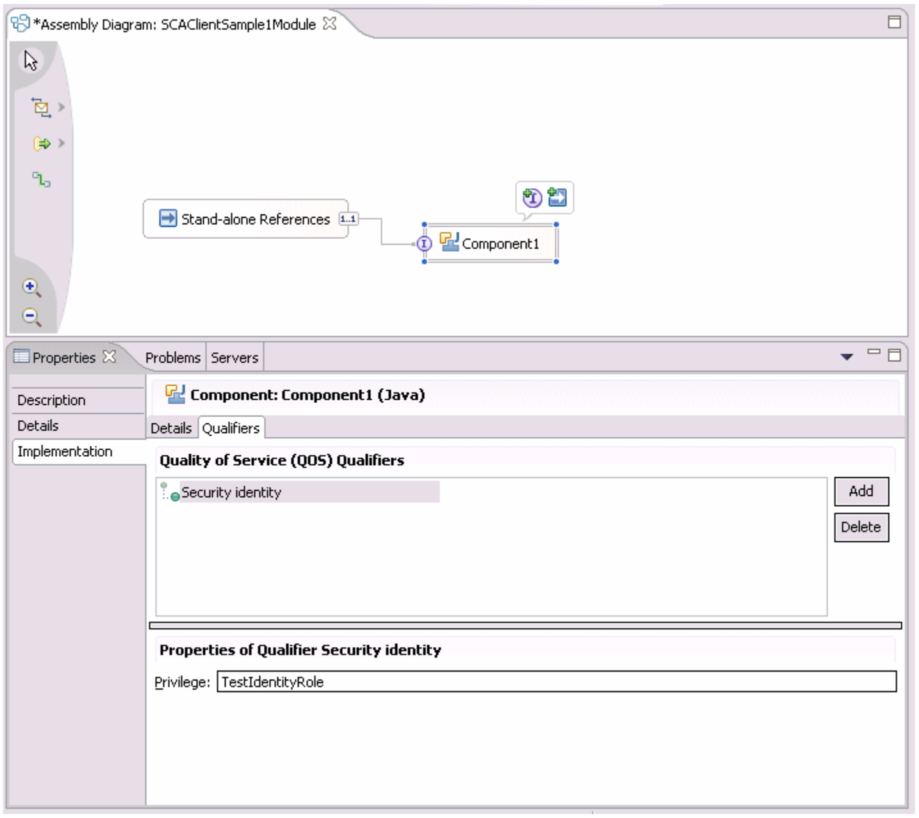
The completed sample uses a client to attempt to make a book order to a secured service (mediation module). The client receives an exception warning that permission is denied.

Note: It is beyond the scope of this redbook to configure a complete end-to-end sample with security, but we want to explore briefly what gets generated in order to give an idea of how to proceed after the declaration has been done.

Usually, you would not be required to deal with the generated J2EE artifacts. In this case we make an exception, because we do not want to configure a complete security infrastructure.

Perform the following:

1. Import the SCA client module and the SCA client using Project Interchange:
   a. Click **File** -> **Import**, select **Project Interchange**, and click **Next**.
   b. Browse to SCAClient.zip which you can find in the additional material supplied with this redbook in the \Clients\Solutions directory.
   c. Click **Select All** then click **Finish**.
2. In the Business Integration view, expand **SCAClientSample1Module** and open the assembly editor.
3. Assign a security identity qualifier to the mediation (Figure 13-11 on page 452).
   a. In the assembly editor select the SCA component **Component1**.
   b. In the Properties view select the **Implementation** tab.
   c. Select the **Qualifiers** tab from the Properties view.
   d. Press the **Add** button.
   e. From the Add Qualifier dialog select **Security identity** as the Quality of Service qualifier and click **OK**.
   f. From the Properties view select **Security identity** and enter the Privilege name of `TestIdentityRole`.



Component: Component1 (Java)

Quality of Service (QOS) Qualifiers

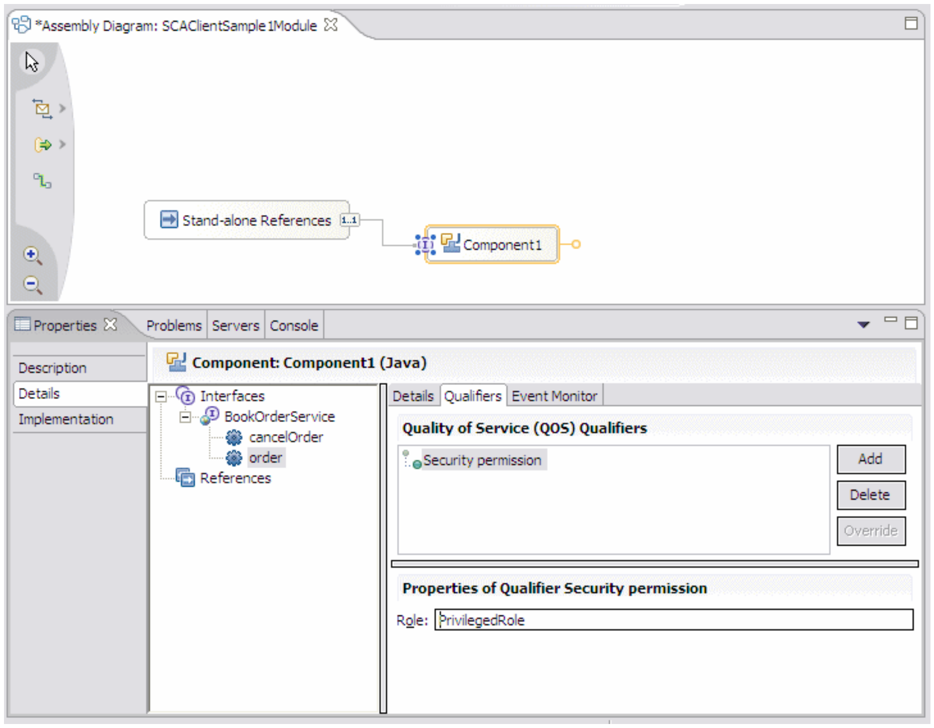
Security identity

Properties of Qualifier Security identity

Privilege: TestIdentityRole

Figure 13-11 Adding a security identity to a mediation implementation

4. Assign a security permission qualifier to the order operation (Figure 13-12 on page 453).
   a. In the assembly editor select the Java component **Component1**.
   b. In the Properties view select the **Details** tab.
   c. Expand the Interfaces tree and select the **order** operation on the BookOrderService interface.
   d. Select the **Qualifiers** tab and click the **Add** button.
   e. From the Add Qualifier dialog, select **Security permission** and click **OK**.
   f. In the Properties view select **Security permission** and enter the Role `PrivilegedRole`.
   g. Save the module.



Component: Component1 (Java)

Quality of Service (QOS) Qualifiers

Security permission

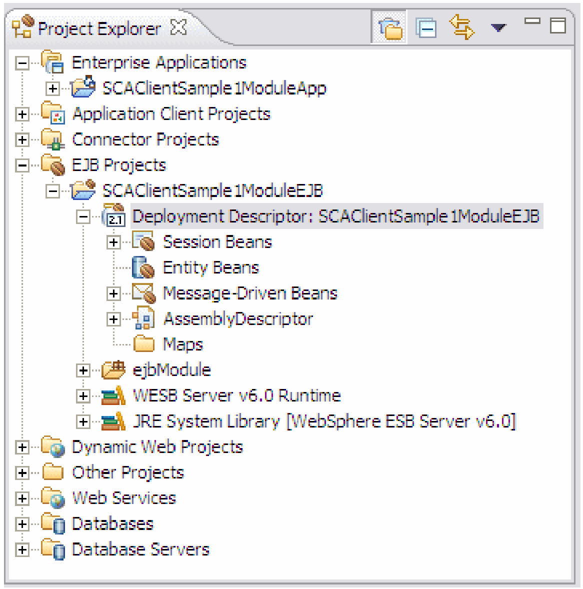
Properties of Qualifier Security permission

Role: PrivilegedRole

Figure 13-12 Specifying a security permission qualifier for operations

Now let’s go and locate the output of the generation process. We need to switch to the J2EE perspective to view the required resources.

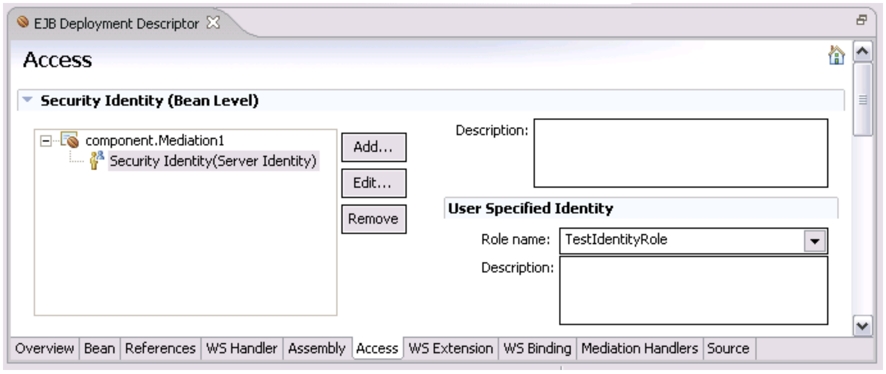
5. Select from the menu bar **Window -> Open Perspective -> Other**.
6. From the Select Perspective dialog select **J2EE** and click **OK**.
7. Open the deployment descriptor in the SCAClientSample1ModuleEJB, under EJB Projects (Figure 13-13 on page 454).



```
Project Explorer
⊟ Enterprise Applications
    ⊞ SCAClientSample1ModuleApp
⊞ Application Client Projects
⊞ Connector Projects
⊟ EJB Projects
    ⊟ SCAClientSample1ModuleEJB
        ⊟ Deployment Descriptor: SCAClientSample1ModuleEJB
            ⊞ Session Beans
              Entity Beans
            ⊞ Message-Driven Beans
            ⊞ AssemblyDescriptor
              Maps
        ⊞ ejbModule
        ⊞ WESB Server v6.0 Runtime
        ⊞ JRE System Library [WebSphere ESB Server v6.0]
⊞ Dynamic Web Projects
⊞ Other Projects
⊞ Web Services
⊞ Databases
⊞ Database Servers
```

Figure 13-13 J2EE deployment descriptor

8. In the EJB Deployment Descriptor editor, select the **Access** tab to view the associated security identity (Figure 13-13).



EJB Deployment Descriptor

Access

Security Identity (Bean Level)

```
⊟ component.Mediation1
      Security Identity(Server Identity)
```

Add... Edit... Remove

Description:

User Specified Identity

Role name: TestIdentityRole

Description:

Overview | Bean | References | WS Handler | Assembly | Access | WS Extension | WS Binding | Mediation Handlers | Source

Figure 13-14 Run-as specification in the deployment description

As far as the security permission qualifier is concerned the generation process does not map the role directly to the method level of the components in the deployment descriptor, since Enterprise Java Beans have generic interfaces.

Therefore only a role reference gets generated, which is verified in the generated code. Example 13-1 shows the relevant part of the deployment descriptor source.

Example 13-1 Generated J2EE role reference in the deployment descriptor

```
...
   <security-role-ref>
      <description>PrivilegedRole</description>
      <role-name>PrivilegedRole</role-name>
      <role-link>PrivilegedRole</role-link>
   </security-role-ref>
</session>
...
```

9. Deploy the module to the server.
   a. Switch to the Servers view.
   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
   c. Add SCAClientSample1Module.
   d. Click **Finish**.

To test the security we have enabled on the module we must enable global security on our WebSphere Enterprise Service Bus server.

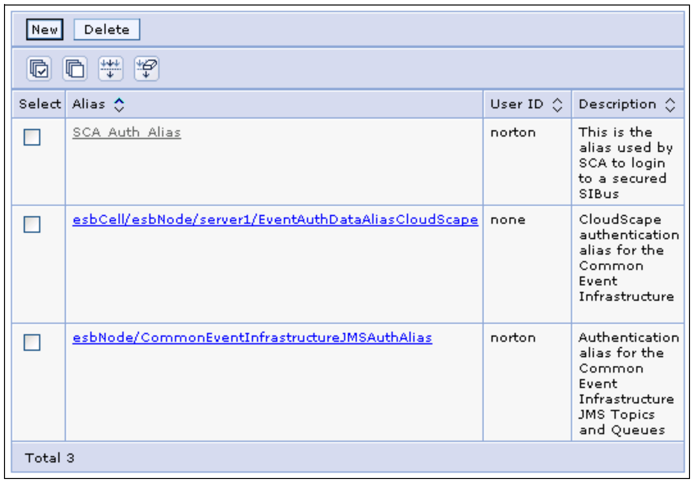
10. In the Servers view, right-click on the WebSphere Enterprise Service Bus server and select **Run administrative console**.
11. Log into the console.
12. Click **Security -> Global Security**.
13. Under User Registries click **Local OS**.
14. Enter a user ID and password for accessing the server.

Note: The username and password you enter must be the username and password you used to log onto the machine.

15. Click **OK**. You are returned to the Global Security properties panel.
16. Under Authentication, expand **JAAS configuration** and click **J2C Authentication data**.

You will see three entries (Figure 13-15 on page 456).

The entries to authenticate with the messaging engine (SCA_Auth_Alias) and the CEI topics and queues (esbNode/CommonEventInfrastructureJMSAuthAlias) currently use a user Id of wid. We need to change this to our user ID.

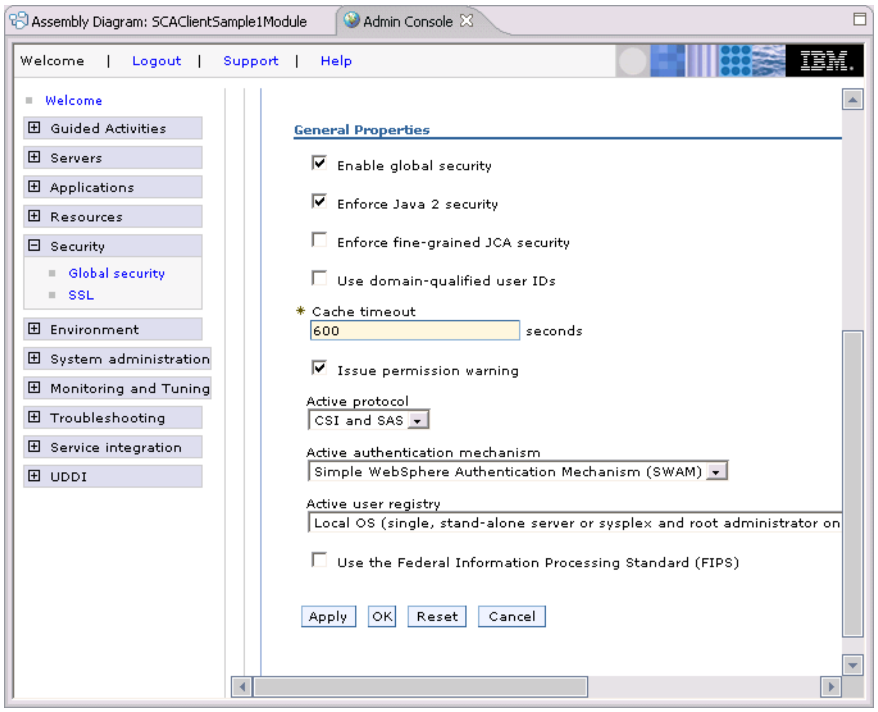


Select	Alias	User ID	Description
☐	SCA_Auth_Alias	norton	This is the alias used by SCA to login to a secured SIBus
☐	esbCell/esbNode/server1/EventAuthDataAliasCloudScape	none	CloudScape authentication alias for the Common Event Infrastructure
☐	esbNode/CommonEventInfrastructureJMSAuthAlias	norton	Authentication alias for the Common Event Infrastructure JMS Topics and Queues

Total 3

Figure 13-15 J2C authentication data

17. Click on **SCA_Auth_Alias**.
18. Enter your username and password and click **OK**.
19. Click on **esbNode/CommonEventInfrastructureJMSAuthAlias**.
20. Enter your username and password and click **OK**.
21. Return to the Global Security panel.
22. Under General Properties, check the **Enable global security** check box. The Enforce Java 2 security check box will also become checked.
23. Ensure the Active User registry drop down menu is set to Local OS (Figure 13-16 on page 457). Click **OK**.



General Properties

- [✓] Enable global security
- [✓] Enforce Java 2 security
- [ ] Enforce fine-grained JCA security
- [ ] Use domain-qualified user IDs

* Cache timeout: 600 seconds

- [✓] Issue permission warning

Active protocol: CSI and SAS

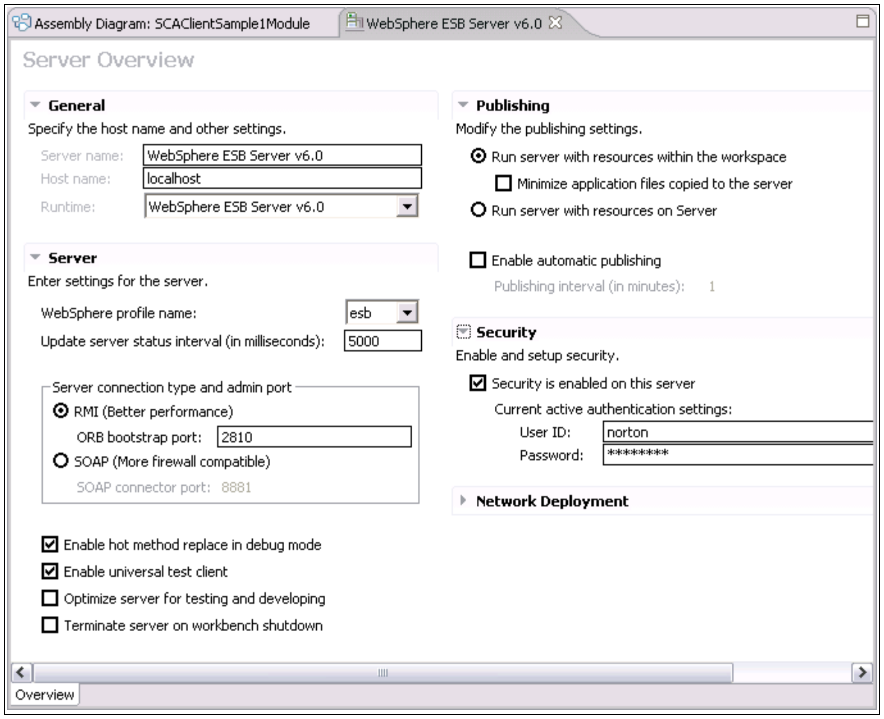
Active authentication mechanism: Simple WebSphere Authentication Mechanism (SWAM)

Active user registry: Local OS (single, stand-alone server or sysplex and root administrator on

- [ ] Use the Federal Information Processing Standard (FIPS)

Figure 13-16 Enabling security

24. Save the changes by clicking the **Save** link and confirm by pressing the **Save** button. When saved, close the administrative console.
25. In the Servers view double click on your WebSphere Enterprise Service Bus server to open the Server Overview panel.
26. Expand the **Security** section and check the **Security is enabled on this server** check box.
27. Enter the username and password you just specified into the relevant fields and save the changes (Figure 13-17 on page 458).


Assembly Diagram: SCAClientSample1Module
WebSphere ESB Server v6.0
Overview

Server Overview

General
Specify the host name and other settings.
Server name: WebSphere ESB Server v6.0
Host name: localhost
Runtime: WebSphere ESB Server v6.0

Server
Enter settings for the server.
WebSphere profile name: esb
Update server status interval (in milliseconds): 5000

Server connection type and admin port
[✓ RMI (Better performance)]
ORB bootstrap port: 2810
SOAP (More firewall compatible)
SOAP connector port: 8881

[✓ Enable hot method replace in debug mode]
[✓ Enable universal test client]
Optimize server for testing and developing
Terminate server on workbench shutdown

Publishing
Modify the publishing settings.
[✓ Run server with resources within the workspace]
Minimize application files copied to the server
Run server with resources on Server

Enable automatic publishing
Publishing interval (in minutes): 1

Security
Enable and setup security.
[✓ Security is enabled on this server]
Current active authentication settings:
User ID: norton
Password: *********

Network Deployment

Figure 13-17 Server overview panel

When the enterprise application gets deployed on the server, the J2EE roles are mapped and resolved to the local security infrastructure.

28.Now restart the server for the security changes to be persisted.

Note: When restarting the server you may find that a error occurs indicating that a server may already be running. If this occurs the server stop has failed and you will need to kill the java process using Task Manager or the kill command.

Note: When using the global security and starting the server from WebSphere Integration Developer you will find that although you see the message *server open for e-business*, the Servers view still shows the server as starting. This prevents you from accessing the server from WebSphere Integration Developer but it can still be accessed through the administrative console at:

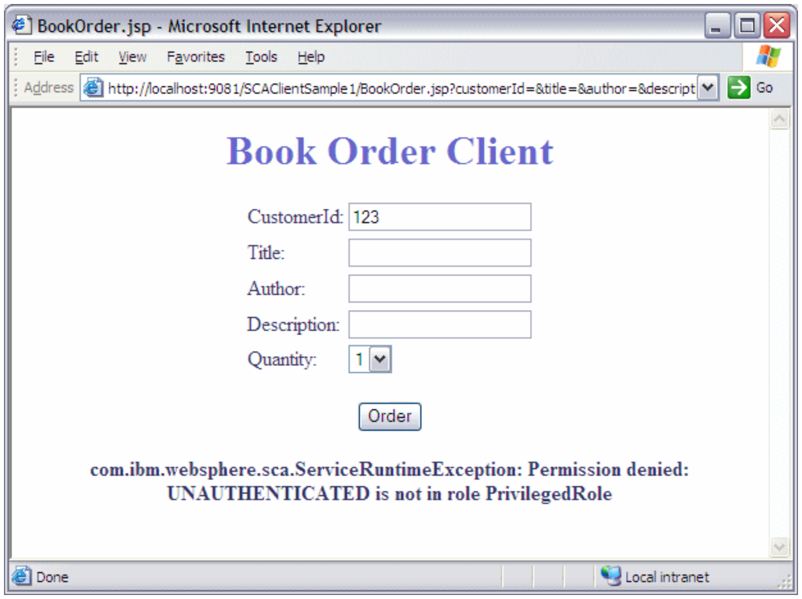
```
http://localhost:9061/ibm/console
```

29.Open a browser and enter the URL:

`http://localhost:9081/SCAClientSample1/BookOrder.jsp.`

30. Enter values into the fields and click the **Order** button.

You should see an exception is returned stating that permission has been denied (Figure 13-18 on page 459).


BookOrder.jsp - Microsoft Internet Explorer

Address http://localhost:9081/SCAClientSample1/BookOrder.jsp?customerId=&title=&author=&descript

Book Order Client

CustomerId:	123
Title:	
Author:	
Description:	
Quantity:	1

Order

com.ibm.websphere.sca.ServiceRuntimeException: Permission denied: UNAUTHENTICATED is not in role PrivilegedRole

Done | Local intranet

Figure 13-18 Sample output of an unauthorized access

31. Once testing is complete enter the URL `http://localhost:9061/ibm/console` into your browser and log in to the console.
32. From here disable global security by clicking **Security -> Global Security** and unchecking the **Enable global security** check box and the **Enforce Java 2 security** check box.
33. Click **OK** and save the changes.
34. In WebSphere Integration Developer in the Servers view, double click on your WebSphere Enterprise Service Bus server. Expand **Security** and uncheck **Security is enabled on this server**. Save and close the editor.
35. Restart the server and remove the deployed module.

Congratulations, you have successfully demonstrated how to enable security on a mediation module.

For more information

- For a detailed walkthrough (including authentication) on the application of security qualifiers see the following article:

  Defining a J2EE role on Service Component Architecture components with WebSphere Integration Developer 6.0.1

  `http://www-128.ibm.com/developerworks/websphere/library/techarticles/0602_charpentier/0602_charpentier.html`

- For more information on the security model of WebSphere Process Server (mostly applicable to WebSphere Enterprise Service Bus):

  WebSphere Process Server security overview

  `http://www-128.ibm.com/developerworks/websphere/library/techarticles/0602_khangoankar/0602_khangaonkar.html`

- For general information on J2EE security in WebSphere Application Server V6 and how to setup security refer to the redbook *WebSphere Application Server V6 Security Handbook*, SG24-6316*.*

13.3 Transactions

This sample shows how to control transactional behavior in mediation modules and mediation primitives.

A transaction is used to group units of work together. If an exception occurs during a transaction every unit of work performed within that transaction will be rolled-back, otherwise they are all committed.

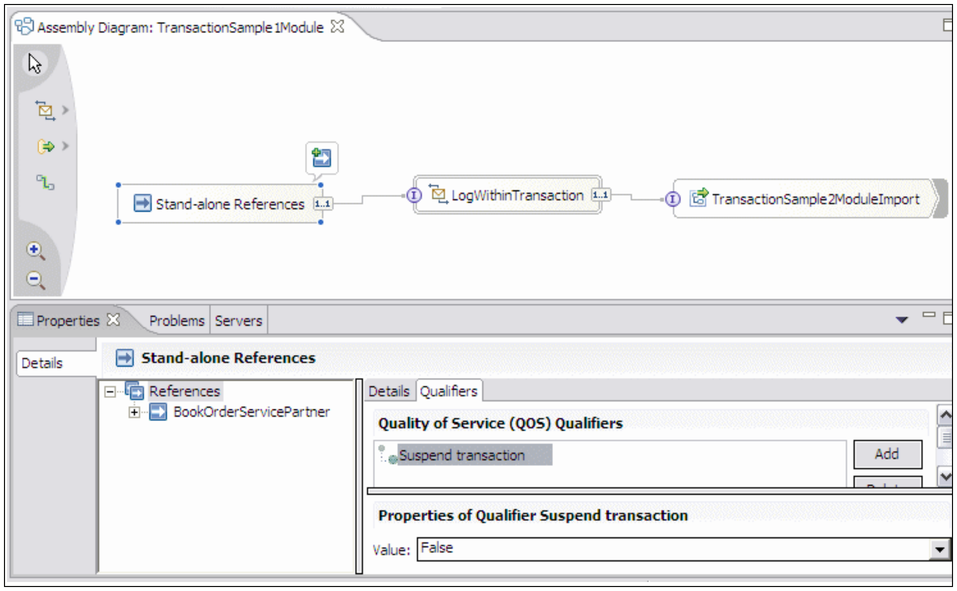
This sample involves:

- Importing a mediation module.
- Specifying transaction qualifiers on a mediation module.
- Defining transaction scope on a Message Logger mediation primitive.
- Testing a completed transaction.
- Testing a rolled-back transaction.
- Viewing whether messages where logged to a Cloudscape database.

The completed sample will demonstrate ordering a book, in one instance the order is successful and a confirmation ID is returned. In the other case the transaction is rolled-back and no confirmation ID is returned.

Note: Rather than building modules from scratch, we will concentrate on reviewing the transaction settings and test the transactional behavior.

1. Import existing resources:
   a. From the menubar select **File -> Import**. Click **Project Interchange** and click **Next**.
   b. Browse to TransactionSampleResources.zip which is located in the additional material supplied with this redbook in the \QualityOfService\Resources\Transactions directory.
   c. Click **Select All** then click **Finish**.
   d. Switch to the J2EE perspective
      i. From the menubar select **Window -> Open Perspective -> Other**.
      ii. Select **J2EE** and click **OK**.
   e. Expand the **EJB Projects** folder.
   f. You will see an error on the SCATranSample1EJB project.
   g. Expand **SCATranSample1EJB** → **Deployment Descriptor** → **Session Beans**.
   h. Right-click **Default Session** and select **Deploy**. This will resolve the error.
2. Review the transaction settings of TransactionSample1Module.
   a. Switch back to the Business Integration perspective.
   b. Open the Assembly Diagram of TransactionSample1Module using .
   c. Select **Stand-alone References**. In the Properties view, select the **Qualifiers** tab.
   d. Select the **Suspend transaction** qualifier. The value should show false, indicating that the clients transaction is not suspended here Figure 13-19 on page 462.



Stand-alone References

Quality of Service (QOS) Qualifiers

Suspend transaction

Properties of Qualifier Suspend transaction

Value: False

Figure 13-19 Suspend transaction set to false

e. In the Assembly Diagram, select the **LogWithinTransaction** mediation flow component. In the Properties view review the transaction settings for the Interface, Reference, and the Implementation.

f. Now, select the import and review the quality of service settings of its interface.

3. Review the transaction settings of TransactionSample2Module using the Assembly Diagram.

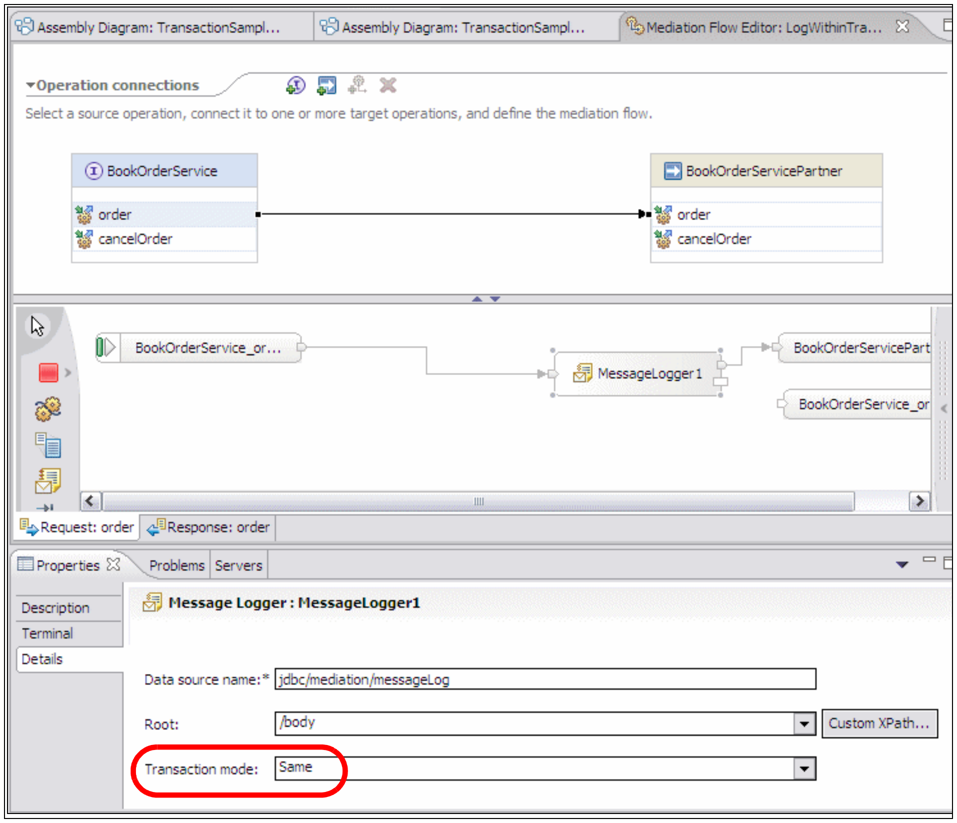
4. Review the transaction settings of the Message Logger mediation primitives in the mediation flow components.

   a. In TransactionSample1Module open the LogWithinTransaction mediation flow.

   b. Select the **MessageLogger1** mediation primitive.

   c. In the Properties view select the **Details** tab.

   d. The transaction mode is set to Same, indicating that the database access to log the message body is performed in the same transaction used in the SCA layer (Figure 13-20 on page 463).



Message Logger : MessageLogger1

Data source name:* jdbc/mediation/messageLog

Root: /body

Transaction mode: Same

Figure 13-20 Message logger primitive transaction settings

a. In TransactionSample2Module open the LogWithinTransaction2 mediation flow.

b. Select the **MessageLogger1** primitive.

c. In the Properties view select the **Details** tab.

d. The transaction mode is also set to Same here.

Note: The filter primitive checks if the book title is *rollback*. If yes, the message is passed to the *fail1* primitive, which throws an exception. We added this logic for testing purposes. If the book title sent is any string but *rollback*, the transaction will commit. If it is *rollback* the transaction will rollback, because the exception is thrown before the transaction is completed.

For an overview of the transaction settings in all the components of the two modules used, see Figure 13-21.

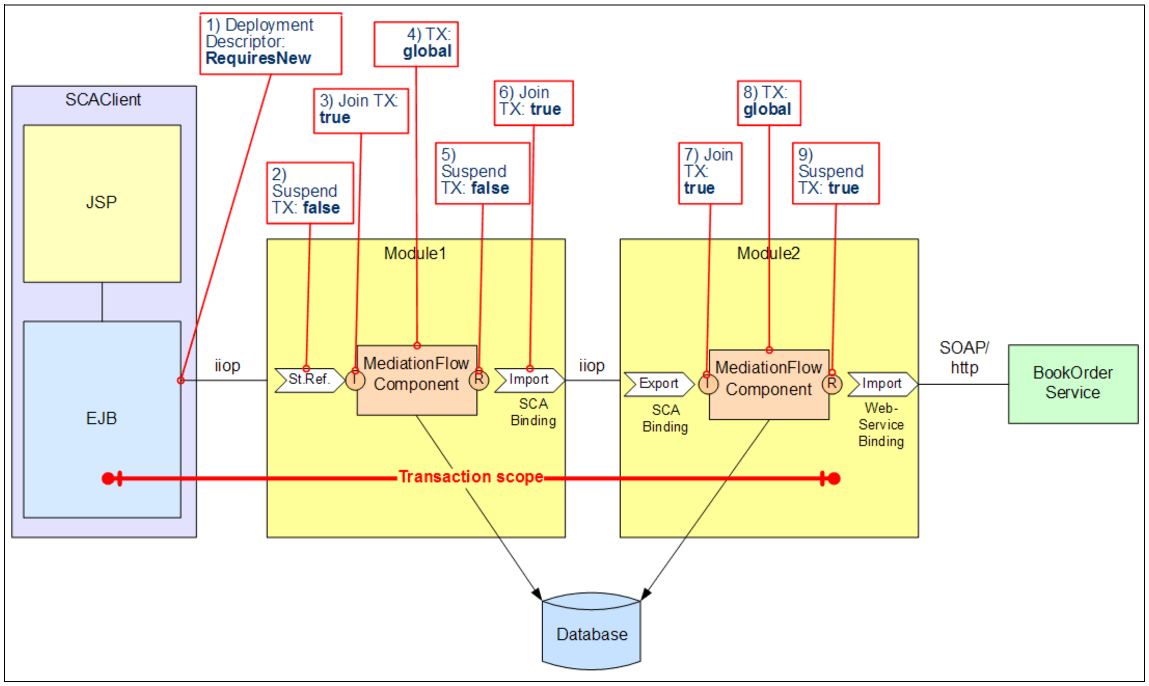


Figure 13-21 Transaction settings overview

We wanted to achieve either all mediation steps are executed, including the logging of the message, or none of them.

We start a transaction in the clients EJB. As we set Suspend transaction to false in the stand-alone reference, this transaction is used within Module1. We do not suspend the transaction at the reference of the mediation flow component within it and the import still joins the transaction, so the transaction context is passed over to the second module. At the reference of the second module's mediation flow component we then suspend the transaction. Therefore sending the SOAP request to the Web service is outside of the transaction scope.

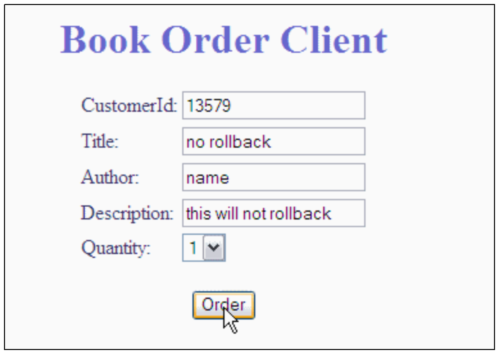
For the implementation of both mediation flow components we set the transaction to global. That results in the database accesses in the message logger primitives to take part in the transaction. So either the message is logged twice, or not at all.

Note: In our test client we call the EJB directly from a JSP, but this is not good practice. When you develop a client, it should use the *Model, View, Controller* pattern.

5. Deploy the modules and Web service to the server.
   a. Switch to the Servers view.
   b. Right-click on your WebSphere Enterprise Service Bus server and select **Add and remove projects**.
   c. Add BookOrderServiceEAR, TransactionSample1ModuleApp and TransactionSample2ModuleApp.
   d. Click **Finish**.
6. Test the transactional behavior.
   e. Open a Web browser and enter the following URL:

      `http://localhost:9081/SCATranClientSample1Web/BookOrder.jsp`

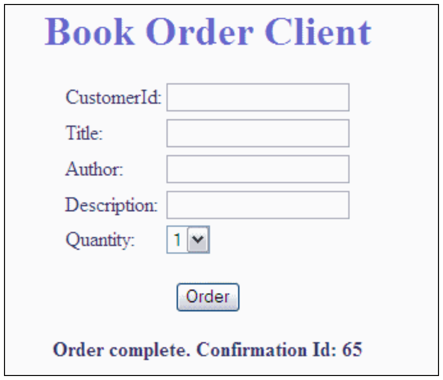
   f. In the BookOrder.jsp enter some test data. For the Title use the string `no rollback` (Figure 13-22).


Book Order Client

CustomerId:	13579
Title:	no rollback
Author:	name
Description:	this will not rollback
Quantity:	1

Figure 13-22 Start first test

   g. Click the **Order** button.
   h. The Web browser should display the order number created in the BookOrder Web service (Figure 13-23).

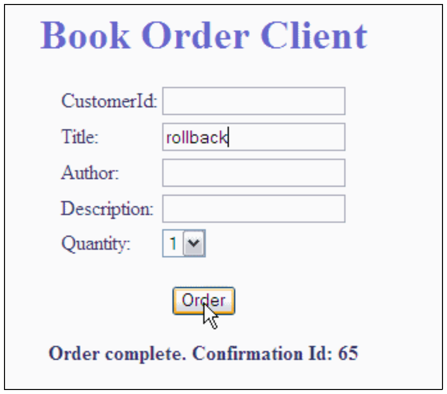

Book Order Client

CustomerId:	
Title:	
Author:	
Description:	
Quantity:	1

Order complete. Confirmation Id: 65

Figure 13-23 Result of first test

i. While the request was sent from the JSP to the Web service the message was logged twice to the database. Now enter the Title `rollback` and click on the **Order** button (Figure 13-24 on page 466).

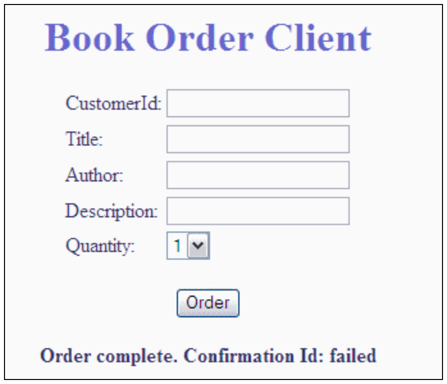

Book Order Client

CustomerId:	
Title:	rollback
Author:	
Description:	
Quantity:	1

Order complete. Confirmation Id: 65

Figure 13-24 Start second test with rollback

j. The text in the browser now indicates that no Confirmation Id was sent back (Figure 13-25 on page 467).


Book Order Client

CustomerId:	
Title:	
Author:	
Description:	
Quantity:	1

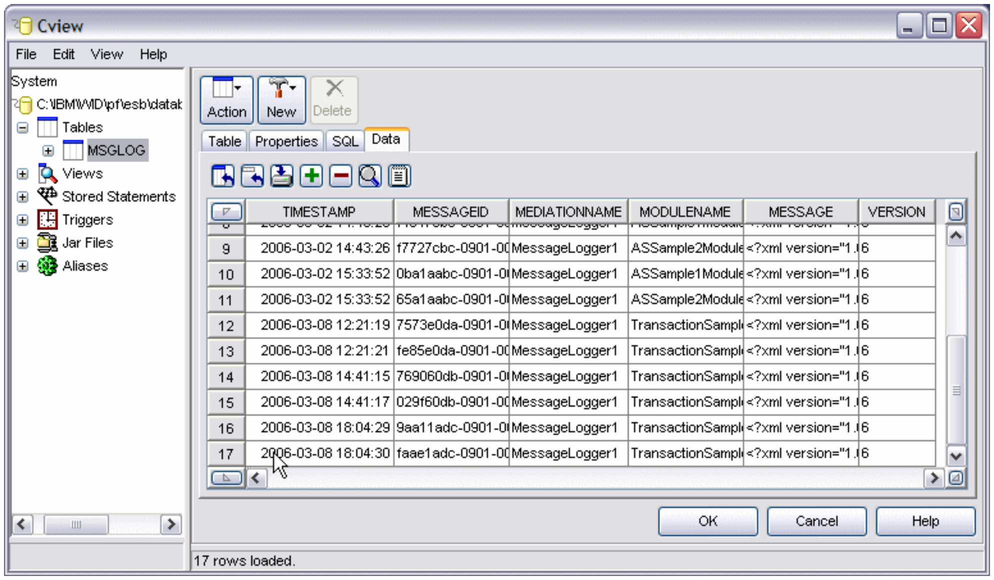
Order complete. Confirmation Id: failed

Figure 13-25 Result of second test

k. The request was prepared to be logged twice to the database, but as a exception was thrown, before the transaction was committed, the database entries should be rolled back.

Lets check the messages that have been logged to the database.

7. Stop the server.
8. Run the utility cview.bat, which is available in <WID_INSTALL>/runtimes/bi_v6/cloudscape/bin/embedded directory.
9. Click on **File -> Open** and open the Cloudscape database **EsbLogMedDB** which is in the directory <WID_INSTALL>/pf/esb/databases.
10. This will open the database. Expand **Tables** and select **MSGLOG.**
11. Click on the **Data** tab to show the records in the table (Figure 13-26 on page 468).

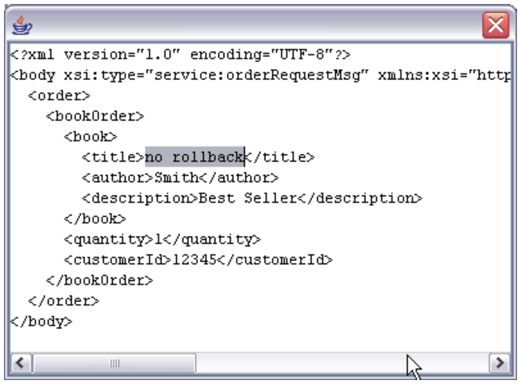


	TIMESTAMP	MESSAGEID	MEDIATIONNAME	MODULENAME	MESSAGE	VERSION
9	2006-03-02 14:43:26	f7727cbc-0901-00	MessageLogger1	ASSample2Module	<?xml version="1.	6
10	2006-03-02 15:33:52	0ba1aabc-0901-0	MessageLogger1	ASSample1Module	<?xml version="1.	6
11	2006-03-02 15:33:52	65a1aabc-0901-0	MessageLogger1	ASSample2Module	<?xml version="1.	6
12	2006-03-08 12:21:19	7573e0da-0901-0	MessageLogger1	TransactionSampl	<?xml version="1.	6
13	2006-03-08 12:21:21	fe85e0da-0901-00	MessageLogger1	TransactionSampl	<?xml version="1.	6
14	2006-03-08 14:41:15	769060db-0901-0	MessageLogger1	TransactionSampl	<?xml version="1.	6
15	2006-03-08 14:41:17	029f60db-0901-00	MessageLogger1	TransactionSampl	<?xml version="1.	6
16	2006-03-08 18:04:29	9aa11adc-0901-0	MessageLogger1	TransactionSampl	<?xml version="1.	6
17	2006-03-08 18:04:30	faae1adc-0901-00	MessageLogger1	TransactionSampl	<?xml version="1.	6

17 rows loaded.

Figure 13-26 Log database entries

12. Verify that there are only two entries in the timeframe of the two test runs.
13. Select the latest message in the **MESSAGE** column and click the **Text Editor** button.
14. You should see the book title you entered for the first test (Figure 13-27).



```
<?xml version="1.0" encoding="UTF-8"?>
<body xsi:type="service:orderRequestMsg" xmlns:xsi="http
  <order>
    <bookOrder>
      <book>
        <title>no rollback</title>
        <author>Smith</author>
        <description>Best Seller</description>
      </book>
      <quantity>1</quantity>
      <customerId>12345</customerId>
    </bookOrder>
  </order>
</body>
```

Figure 13-27 Message content logged in the database

15. Also look at the content of the second last message and verify that the book title is the one from the first test.

Note: The request message from the second test were also prepared to be written to the database twice, but as the log primitives participate in the global transaction started by the client and the transaction was rolled back they were not finally written.

16. Close the cview.bat utility.
17. Start the server and remove all projects from the server.

Part 5

Appendixes

A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG247212
```

Alternatively, you can go to the IBM Redbooks Web site at:

```
ibm.com/redbooks
```

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247217.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
SG247212.zip	Zipped Code Samples

This ZIP file contains all of the resources required to complete the development examples in this redbook. It contains resources that you will need to import for certain development examples. It also contains solutions to each development example, stored in Project Interchange ZIP files.

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.

B

Hints and tips

This chapter lists a few obstacles the team ran into while creating the samples for this redbook and workarounds for them accordingly. You may not necessarily observe these behaviors in subsequent releases of the product.

This section covers the following hints:

- Force complete regeneration
- Update of business objects
- Renaming of resources
- Testing of a mediation flow component standalone
- Incompatible target runtimes

Resolving obstacles with WebSphere Integration Developer

Each hint in this section describes the scenario where we encountered an issue, where the behavior occurred, and how we resolved the issue.

Force complete regeneration

You have developed a module and added the project to the server.

Observed behavior

You observe a stack trace in the Console view when executing the mediation module warning that a WSDL file cannot be found.

Resolution

Force a clean regeneration and deployment:

1. Remove the project from the server
2. Run a project *clean* (**Project** -> **Clean**)
3. Add the project again to the server.

Note: In some cases you may need to stop the server after step 1) and start the server after step 2).

Update of business objects

You have developed a module and added the project to the server. Now you change the business objects used by the module (for example, you add attributes).

Observed behavior

The changes are not reflected in your test environment even if you redeploy the project.

Resolution

The objects are cached so you will need to restart the server.

Renaming of resources

When you have developed a complete mediation module and you change the name of an element such as an interface.

Observed behavior

Sometimes the changes are not propagated to all references of that element, so a number of errors appear in the Problems view.

Resolution

Go through all references of the changed element manually and make sure that the references are updated properly. This can be achieved by removing references and rewiring the module. Run a project clean (**Project** -> **Clean**).

Testing of a mediation flow component standalone

You create a mediation flow component with a reference containing a business object. The reference is not wired to an import. You want to test this component with the Integration Test Client.

Observed behavior

You experience an exception in the Integration Test Client (Example B-1).

Example: B-1 Exception reported by the Integration Test Client

```
com.ibm.wsspi.sibx.mediation.flow.MediationRuntimeException: CWSXM1025E: An
unexpected exception occurred during flow invocation: index=0, size=0
   at
com.ibm.wsspi.sibx.mediation.flow.ejb.MediationFlowBean.invokeRequestFlow(Media
tionFlowBean.java:200)
...
```

Resolution

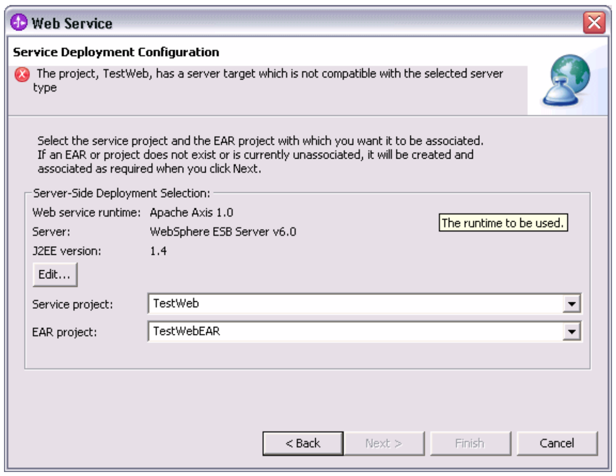
Add an import to the Assembly Editor, wire it to the reference of the mediation flow and generate a binding. The Integration Test Client should run successfully and can be used to emulate the import.

Incompatible target runtimes

You create a new Web project, for example for creating a client application that accesses a mediation module. You have multiple test environments and you did not explicitly set **WebSphere ESB** to be the default test environment.

Observed behavior

When you try to generate the Java client from the WSDL file you can run into problems and the wizards complain that the associated test environment is not compatible with the target one (Figure B-1 on page 478).


Web Service

Service Deployment Configuration

The project, TestWeb, has a server target which is not compatible with the selected server type

Select the service project and the EAR project with which you want it to be associated.
If an EAR or project does not exist or is currently unassociated, it will be created and associated as required when you click Next.

Server-Side Deployment Selection:

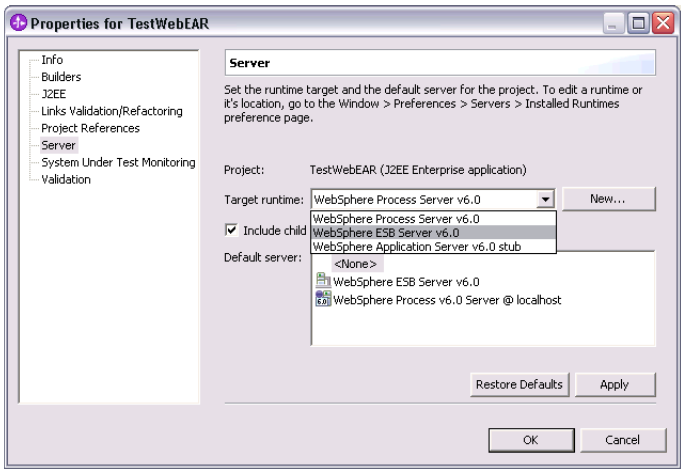
Web service runtime:	Apache Axis 1.0
Server:	WebSphere ESB Server v6.0
J2EE version:	1.4
Service project:	TestWeb
EAR project:	TestWebEAR

Figure B-1 Mismatch of associated test server environment

Resolution

By default (sometimes only visible when you expand the Advanced section in a wizard) the WebSphere Process Server test environment gets associated with a new project in WebSphere Integration Developer. Therefore you need to change the associated test environment:

1. Go the J2EE perspective
2. In the Project Explorer view open the tree of **Enterprise Applications** and right-click on your project name.
3. Choose **Properties** and change the target runtime for your project under the **Server** category to **WebSphere ESB**.


Properties for TestWebEAR
Info
Builders
J2EE
Links Validation/Refactoring
Project References
Server
System Under Test Monitoring
Validation
Server
Set the runtime target and the default server for the project. To edit a runtime or it's location, go to the Window > Preferences > Servers > Installed Runtimes preference page.

Project: TestWebEAR (J2EE Enterprise application)

Target runtime: WebSphere Process Server v6.0

New...
WebSphere Process Server v6.0
WebSphere ESB Server v6.0
WebSphere Application Server v6.0 stub
Include child
Default server:
<None>
WebSphere ESB Server v6.0
WebSphere Process v6.0 Server @ localhost
Restore Defaults
Apply
OK
Cancel

Figure B-2 Changing the projects target runtime

Abbreviations and acronyms

BO	Business Object
CBE	Common Base Events
CEI	Common Event Infrastructure
EAI	Enterprise Application Integration
EAR	Enterprise Archive
EIS	Enterprise Information System
EJB	Enterprise Java Beans
ESB	Enterprise Service Bus
J2C	J2EE Connector Architecture
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
MQI	Message Queuing Interface
OAM	Object Authority Manager
QA	Quality Assurance
RSDP	Rational Software Development Platform
SCA	Service Component Architecture
SCDL	Service Component Definition Language
SDO	Service Data Object
SIT	System Integration Test
SMO	Service Message Object
SOA	Service-oriented architecture
UTE	Unit Test Environment
WSDL	Web services Description Language

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 484. Note that some of the documents referenced here may be available in softcopy only.

- *Rational Application Developer V6 Programming Guide*, SG24-6449
- *WebSphere Version 6 Web Services Handbook Development and Deployment,* SG24-6461
- *WebSphere Application Server V6: System Management and Configuration Handbook*, SG24-6451
- *WebSphere Application Server Network Deployment V6: High Availability Solutions,* SG24-6688
- *WebSphere Application Server V6 Security Handbook*, SG24-6316

Other publications

These publications are also relevant as further information sources:

- *Patterns for e-business: A Strategy for Reuse,* by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos, ISBN 1931182027

Online resources

These Web sites and URLs are also relevant as further information sources:

- WebSphere Enterprise Service Bus home page:

  `http://www.ibm.com/software/integration/wsesb/`
- WebSphere Application Server home page:

  `http://www.ibm.com/software/webservers/appserv/was/`
- WebSphere Process Server home page:

http://www.ibm.com/software/integration/wps/

- WebSphere MQ home page:

  http://www.ibm.com/software/integration/wmq/

- WebSphere Message Broker home page:

  http://www.ibm.com/software/integration/wbimessagebroker/v6/

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

IBM Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

IBM Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(0.2”spine)
0.17”<->0.473”
90<->249 pages

(0.1”spine)
0.1”<->0.169”
53<->89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM

Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(2.5" spine)
2.5"<->nnn.n"
1315<-> nnnn pages

IBM

Redbooks

Getting Started with WebSphere Enterprise Service Bus V6

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages

Getting Started with WebSphere Enterprise Service Bus V6

Build ESB solutions using SCA and Web services

Implement mediation flows in WebSphere Integration Developer

Learn by example with practical scenarios

IBM WebSphere Enterprise Service Bus is a flexible connectivity infrastructure for integrating applications and services, designed to enable the development of a service-oriented architecture (SOA).

This IBM Redbook guides you through the capabilities and product features of WebSphere Enterprise Service Bus V6.0. It also contains many step-by-step examples of building resources for WebSphere Enterprise Service Bus using WebSphere Integration Developer.

Part 1 of this book introduces WebSphere Enterprise Service Bus and positions it among IBM's other SOA and Enterprise Service Bus product offerings.

Part 2 describes how to install and configure both WebSphere Enterprise Service Bus and WebSphere Integration Developer, and explains how to perform key concepts and tasks using these products.

Part 3 explains the administration and testing capabilities, including step-by-step examples.

Part 4 provides a wealth of development examples showing step-by-step how to develop solutions using mediation primitives, integrate with services, and deliver qualities of service.

SG24-7212-00 ISBN